

# Data Protection Assurance by Design: Support for Conflict Detection, Requirements Traceability and Fairness Analysis

by  
M.Sc. Qusai Ramadan

Approved Dissertation thesis for the partial fulfillment of the requirements for a  
Doctor of Natural Sciences (Dr. rer. nat.)  
Fachbereich 4: Informatik  
Universität Koblenz-Landau

Chair of PhD Board:	Prof. Dr. Jan Jürjens
Chair of PhD Commission:	Prof. Dr. Karin Harbusch
Examiner and Supervisor:	Prof. Dr. Jan Jürjens
Further Examiner:	Prof. Dr. Andreas Mauthe
Co-Supervisor:	Dr. Daniel Strüber

Date of the doctoral viva: June 19,2020



# Erklärung

Hiermit erkläre ich gemäß § 10 Abs. 3 Punkt 4 der Promotionsordnung des Fachbereichs 4: Informatik der Universität Koblenz Landau,

- dass ich die vorliegende Dissertation mit dem Titel “Data Protection Assurance by Design: Support for Conflict Detection, Requirements Traceability and Fairness Analysis” selbst angefertigt und alle benutzten Hilfsmittel in der Arbeit angegeben habe,
- dass ich die Dissertation oder Teile der Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe, und
- dass ich weder diese noch eine andere Abhandlung bei einer anderen Hochschule als Dissertation eingereicht habe,
- dass ich meine individuellen Beiträge an kooperativ erzielten Forschungsergebnissen in der Dissertation an den entsprechenden Stellen gekennzeichnet habe und dass meine Koautoren diese Einschätzung meines Beitrages teilen.

Koblenz, den 23. Juni 2020

---

*Qusai Ramadan (Unterschrift)*



# Abstract

Data-minimization and fairness are fundamental data protection requirements to avoid privacy threats and discrimination. Violations of data protection requirements often result from: First, conflicts between security, data-minimization and fairness requirements. Second, data protection requirements for the organizational and technical aspects of a system that are currently dealt with separately, giving rise to misconceptions and errors. Third, hidden data correlations that might lead to influence biases against protected characteristics of individuals such as ethnicity in decision-making software. For the effective assurance of data protection needs, it is important to avoid sources of violations right from the design modeling phase. However, a model-based approach that addresses the issues above is missing.

To handle the issues above, this thesis introduces a model-based methodology called MoPrivFair (**Model-based Privacy & Fairness**). MoPrivFair comprises three sub-frameworks: First, a framework that extends the SecBPMN2 approach to allow detecting conflicts between security, data-minimization and fairness requirements. Second, a framework for enforcing an integrated data-protection management throughout the development process based on a business processes model (i.e., SecBPMN2 model) and a software architecture model (i.e., UMLsec model) annotated with data protection requirements while establishing traceability. Third, the UML extension UMLfair to support individual fairness analysis and reporting discriminatory behaviors. Each of the proposed frameworks is supported by automated tool support.

We validated the applicability and usability of our conflict detection technique based on a health care management case study, and an experimental user study, respectively. Based on an air traffic management case study, we reported on the applicability of our technique for enforcing an integrated data-protection management. We validated the applicability of our individual fairness analysis technique using three case studies featuring a school management system, a delivery management system and a loan management system. The results show a promising outlook on the applicability of our proposed frameworks in real-world settings.



# Kurzfassung

Moderne Softwaresysteme verwenden personenbezogene Daten, um automatisiert Entscheidungen zu treffen. Um Bedrohungen der Privatsphäre und Diskriminierung effektiv zu verhindern, sind relevante Datenschutzanforderungen, insbesondere bezüglich Datenminimierung und Fairness, zum Gegenstand der Anforderungsanalyse geworden. Solche Datenschutzanforderungen sind momentan drei wesentlichen Arten von Bedrohungen ausgesetzt: 1. Inhärente Konflikte zwischen Sicherheits-, Datenminimierungs- und Fairness-Anforderungen. 2. Die fehleranfällige separate Behandlung von Datenschutzanforderungen für die organisatorischen und technischen Belange eines Systems. 3. Versteckte Korrelationen zwischen Daten, die zu einer unbeabsichtigten Diskriminierung aufgrund von geschützten Personenmerkmalen, wie z.B. der ethnischen Zugehörigkeit einer Person, führen können. Um die Datenschutzanforderungen wirksam zu gewährleisten, ist es wichtig, relevante Schwachstellen bereits während der Entwurfsmodellierung zu erkennen und geeignete Lösungen zu finden. Momentan fehlt jedoch ein Ansatz, der die genannten Probleme auf der Modell-Ebene adressiert.

Diese Arbeit stellt eine modellbasierte Methodik vor, um die oben genannten Probleme zu lösen. Die vorgeschlagene Methodik heißt *MoPrivFair*: **Model-based Privacy & Fairness**. *MoPrivFair* umfasst drei Teilansätze: Erstens eine Erweiterung des SecBPMN2-Ansatzes zur Erkennung von Konflikten zwischen Sicherheits-, Datenminimierungs- und Fairness-Anforderungen. Der Ansatz basiert auf einem Katalog von Konflikt-Mustern, der eine systematische Abdeckung der möglichen Konflikte zwischen den verschiedenen Anforderungstypen bietet. Zweitens ein Framework zur Durchsetzung eines integrierten Datenschutzmanagements während des gesamten Entwicklungsprozesses auf der Grundlage eines Geschäftsprozessmodells (basierend auf SecBPMN2) und eines Softwarearchitekturmodells (basierend auf UMLsec). Annotationen beider Modelle mit Datenschutzanforderungen und eine automatische Transformation gewährleisten die Nachverfolgbarkeit zwischen technischen und organisatorischen Belangen. Drittens die UML-Erweiterung *UMLfair* für die Analyse von Fairnessanforderungen und diskriminierendem Verhalten. Jeder der drei vorgeschlagenen Ansätze wird durch geeignete Softwarewerkzeuge unterstützt.

Im Rahmen der Arbeit wurde die Anwendbarkeit und Verwendbarkeit des Konflikterkennungsverfahrens anhand einer Fallstudie zum Gesundheitsmanagement und einer experimentellen Anwenderstudie validiert. Basierend auf einer Fallstudie zum Flugverkehrsmanagement untersuchten wir die Anwendbarkeit unseres Verfahrens zur Durchsetzung eines integrierten Datenschutzmanagements. Anhand von drei Fallstudien mit einem Schulverwaltungssystem, einem Lieferverwaltungssystem und einem Kreditverwaltungssystem wurde die Anwendbarkeit des Fairness-Analyseverfahrens untersucht. Die Ergebnisse zeigen einen vielversprechenden Ausblick für die Verwendung der Ansätze in der Praxis.



# Acknowledgements

I have been truly fortunate to work with and be inspired by several outstanding people. Having arrived at the end of this challenging, but gratifying journey, I want to thank these people who have made this thesis possible.

Firstly, I am very grateful to Prof. Dr. Jan Jürjens, my main supervisor, not only for his supervision, guidance, and continuous support but for believing in me and pushing me to do more than I thought I could. Through the course of the last four years, I have learned a lot from Jan. In fact, Jan has been a great example to me, and it has been an honor to work under his supervision.

I want to express my appreciation to my co-supervisor Dr. Daniel Strüber for his constructive suggestions that helped me to improve many important details in my thesis. Thank you Daniel for the stimulating ideas and the insightful suggestions.

I wish to thank Prof. Dr. Andreas Mauthe for immediately agreeing to be my second referee, and for his feedback which enhanced several parts of this thesis. My sincere thanks also go to the Ph.D. committee.

I am very thankful to Prof. Dr. Steffen Staab who co-supervised my work on fairness analysis. The discussion with him contributed to improving many essential details in the second and fourth chapters of this thesis.

I am very thankful to my collaborator and co-author, Dr. Mattia Salnitri (Polytechnic University of Milan). It has been a real pleasure working with you, and I hope we continue our collaboration, which has been very fruitful.

I would like to express my thanks to the members of the research group of Prof. Dr. Jan Jürjens, namely Dr. Amir Shayan Ahmadian, Dr. Jens Bürger, Julian Flake, Katharina Großer, Dr. Marco Konersmann, Matthias Lohr, Sven Peldszus and Joachim Stocker for the stimulating discussions during our seminars, off-site meetings but also during the social events. I owe special thanks to Dr. Volker Riedi-

ger for the feedback he has provided me since the very beginning of my research, as well as for his advice on choices I had to make.

My sincere thanks also go to my friends Mahmood Al-Doori, Ahmed Al-Dulaimy and Dr. Zeyd Boukhers for the great academic discussions during the coffee breaks. I wish also to thank the secretaries in our department, namely Sabine Hülstrunk, Silke Werger, and Anne Heibel for their help and support.

I am very thankful to Prof. Dr. Paolo Giorgini and the development team at the University of Trento for providing me with access to the source code of the STS tool, which I extended to automate my work on conflict detection.

Thank you Ahed, my true love, for the great patience, understanding, and unconditional love. You were always around at times I thought that it is impossible to continue. Thank you for always believing in me, and for standing by me through the course of these years, I know it has not been easy. In fact, without you this thesis would not be possible. I appreciate my daughters, my little princesses, Siwar Al-Dahab and Kenda for being the inspiration source. I consider myself extremely fortunate to have such a lovely and caring family. Ahed, Siwar, and Kenda you are the best, and this achievement is dedicated to you.

From my heart thanks to my beloved father and mother for their love and being always so proud of me. I am forever indebted to both of you for all you have done for me. I also wish to thank my parents-in-law for their love and support; I feel lucky to have you in my life.

Finally, I wish to acknowledge the financial supported of: (i) The Deutscher Akademischer Austauschdienst (DAAD); and (ii) the project "Engineering Responsible Information Systems" financed by the University of Koblenz-Landau.

# Contents

<b>Abstract</b>	<b>v</b>
<b>Kurzfassung</b>	<b>vii</b>
<b>Acknowledgements</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Challenges and Research Directions . . . . .	6
1.1.1 Conflict detection . . . . .	7
1.1.2 Integrating business process and software modeling . . . . .	9
1.1.3 Fairness analysis . . . . .	10
1.2 Research Contribution . . . . .	11
1.3 Research Methodology . . . . .	14
1.4 Thesis Outline . . . . .	15
1.5 List of Publications . . . . .	16
<b>2 BPMN-based Framework for Detecting Conflicts between Data Protection Requirements</b>	<b>19</b>
2.1 Introduction . . . . .	20
2.1.1 Problem statement and research questions . . . . .	22
2.1.2 Contribution . . . . .	23
2.2 Background . . . . .	24
2.2.1 Data-minimization concepts . . . . .	25
2.2.2 Overview of data-minimization: legal aspects and standards . . . . .	25
2.2.3 Fairness concepts . . . . .	27
2.2.4 Overview of discrimination from the legal aspects . . . . .	28
2.2.5 BPMN-based data-protection engineering . . . . .	31
2.2.6 SecBPMN2 security concepts . . . . .	32
2.2.7 SecBPMN2-Q . . . . .	34
2.3 Running Example . . . . .	34
2.4 Modeling Security-, Data-Minimization- and Fairness-Requirements	37
2.4.1 Data-minimization and fairness annotations . . . . .	37
2.4.2 The underlying background of the linkage constraints . . . . .	46
2.5 Framework for Detecting Conflicts . . . . .	48

2.6	Alignment Checking . . . . .	54
2.6.1	Modeling SecBPMN2-Q patterns . . . . .	55
2.6.2	Automated alignment checking . . . . .	57
2.7	Conflict Detection . . . . .	58
2.7.1	Automated conflict detection using anti-patterns . . . . .	58
2.7.2	Catalog of domain-independent conflicts: an overview . . . . .	67
2.7.3	Full textual description of the proposed catalog of conflicts . . . . .	71
2.8	Tool Support . . . . .	86
2.8.1	Algorithm for alignment checking . . . . .	87
2.8.2	Algorithm for conflict detection . . . . .	89
2.9	Case Study . . . . .	92
2.10	User Experiment . . . . .	93
2.10.1	Set-up of the experiment . . . . .	94
2.10.2	Results of the experiment . . . . .	95
2.10.3	Threats to validity . . . . .	97
2.11	Limitations and Future Work . . . . .	98
2.12	Related Work . . . . .	100
2.12.1	Conflicts between data protection requirements . . . . .	100
2.12.2	Model-based conflict detection approaches . . . . .	100
2.12.3	Other conflict detection approaches . . . . .	101
2.12.4	Data-minimization-aware approaches . . . . .	102
2.12.5	Fairness-aware approaches . . . . .	103
2.13	Conclusion . . . . .	103
<b>3</b>	<b>Integrating BPMN- and UML-based Data-Protection Engineering</b>	<b>105</b>
3.1	Introduction . . . . .	106
3.1.1	Problem statement and research questions . . . . .	107
3.1.2	Contribution . . . . .	108
3.2	Background . . . . .	109
3.2.1	SecBPMN2 (Revisit) . . . . .	109
3.2.2	UMLsec profile . . . . .	111
3.3	Framework for Integrating BPMN- and UML-based Data-Protection Engineering . . . . .	116
3.4	SecBPMN2 to UMLsec Transformation . . . . .	120
3.4.1	Mapping schema from SecBPMN2 to UMLsec elements . . . . .	120
3.4.2	Transformation rules from SecBPMN2 to UMLsec . . . . .	123
3.5	Tool Support . . . . .	133
3.6	Case Study . . . . .	134
3.6.1	Applying the integration framework . . . . .	135
3.6.2	Transformation results . . . . .	139
3.7	Discussion and Future Work . . . . .	142
3.7.1	Threats to validity and limitations . . . . .	143
3.7.2	Future Work . . . . .	143

---

3.8	Related Work . . . . .	144
3.8.1	Automated transformation . . . . .	144
3.8.2	Manual transformation . . . . .	145
3.9	Conclusion . . . . .	145
<b>4</b>	<b>Individual Fairness Analysis based on Software Design Models</b>	<b>147</b>
4.1	Introduction . . . . .	148
4.1.1	Problem statement and research question . . . . .	149
4.1.2	Contribution . . . . .	150
4.2	Running Example . . . . .	151
4.3	Framework for Analyzing Individual Fairness . . . . .	153
4.4	Annotating the UML Model with Fairness Information . . . . .	158
4.5	Generating Temporal Logic Claims . . . . .	160
4.5.1	Algorithm for generating temporal logic claims . . . . .	164
4.6	Model Checking . . . . .	167
4.7	Reporting on Individual Fairness . . . . .	168
4.7.1	Algorithm for reporting on individual fairness . . . . .	169
4.8	Optional: Generating Proxies From a Database . . . . .	170
4.9	Tool Support . . . . .	171
4.10	Case Studies . . . . .	172
4.11	Discussion . . . . .	175
4.11.1	Using the framework at run-time . . . . .	175
4.11.2	Threats to validity, limitations and future work . . . . .	176
4.12	Related Work . . . . .	177
4.12.1	Software model-based development . . . . .	177
4.12.2	Discrimination detection approaches . . . . .	178
4.13	Conclusion . . . . .	179
<b>5</b>	<b>Conclusions, Limitations, and Outlook</b>	<b>181</b>
5.1	Conclusion . . . . .	181
5.1.1	Evaluation results . . . . .	183
5.1.2	Limitations . . . . .	185
5.2	Outlook . . . . .	187
5.2.1	Conflict resolution . . . . .	187
5.2.2	Tracing requirements in legacy and evolutionary systems . . . . .	187
5.2.3	Support group fairness analysis based on software models . . . . .	187
<b>A</b>	<b>Conflict Detection: A Walk Through Artifacts and Tool Support</b>	<b>189</b>
<b>B</b>	<b>SecBPMN2 to UMLsec: A Walk Through Artifacts and Tool Support</b>	<b>191</b>
<b>C</b>	<b>Curriculum Vitae</b>	<b>199</b>
	<b>Bibliography</b>	<b>201</b>



## List of Figures

1.1	The MoPrivFair ( <b>Model-based Privacy &amp; Fairness</b> ) methodology. . .	5
1.2	Mapping between the research questions and the frameworks of the MoPrivFair methodology. . . . .	8
1.3	Mapping between the research papers that are written by the author of this thesis and the frameworks of the MoPrivFair methodology. . .	16
2.1	The highlighted part (dashed lines) denotes how Chapter 2 contributes to the overall workflow of the MoPrivFair methodology. . .	19
2.2	Running example: Specifying security, data-minimization, and fairness requirements in a healthcare business process. . . . .	35
2.3	All variants of SecBPMN's <i>non-repudiation</i> annotation. . . . .	37
2.4	Meta-model of our BPMN extension. . . . .	38
2.5	All variants of our data-minimization and fairness annotations. . . .	41
2.6	The proposed BPMN-based framework for conflict detection. . . . .	49
2.7	Requirements specified as SecBPMN2-Q patterns. . . . .	55
2.8	Conflicts C1–C5 between <i>non-repudiation</i> and <i>anonymity</i> as anti-patterns. . . . .	60
2.9	Potential conflicts between <i>non-repudiation</i> and <i>anonymity</i> as anti-patterns. . . . .	62
2.10	Conflicts and potential conflicts between <i>fairness</i> , <i>undetectability</i> , and <i>anonymity</i> as constrained anti-patterns. . . . .	64
2.11	Example 2: A simplified version of our running example model with different security and data-minimization annotations. . . . .	71
2.12	Potential conflicts PC1 and PC2 between <i>accountability</i> and <i>anonymity</i> as anti-patterns. . . . .	74
2.13	Conflicts C1 and C2 between <i>authenticity</i> and <i>anonymity</i> as anti-patterns. . . . .	75
2.14	Conflicts C1 and C2 between <i>auditability</i> and <i>anonymity</i> as anti-patterns. . . . .	76
2.15	Potential conflicts PC1 and PC2 between <i>non-delegation</i> and <i>anonymity</i> as anti-patterns. . . . .	78
2.16	Potential conflict PC1 between <i>binding-of-duties</i> and <i>unlinkability</i> as anti-patterns. . . . .	79
2.17	Conflict C1 between <i>anonymity</i> and <i>unobservability</i> as anti-patterns. .	80

2.18	Potential conflict C1 between <i>confidentiality</i> and <i>anonymity</i> as anti-patterns. . . . .	82
2.19	Potential conflicts PC1 and PC2 between <i>availability</i> and <i>anonymity</i> as anti-patterns. . . . .	84
2.20	<i>Analysis results</i> view from our tool. . . . .	86
2.21	Results for RQ1: Error-proneness of manual conflict detection . . . .	96
2.22	Results for RQ2: Perceived helpfulness of automated conflict detection	96
3.1	The highlighted part (dashed lines) denotes how Chapter 3 contributes to the overall workflow of the MoPrivFair methodology. . .	105
3.2	Example model: SecBPMN2 model representing a business process for flight plan negotiation. . . . .	111
3.3	UML deployment diagram of SecBPMN2 example . . . . .	113
3.4	UML class diagram of SecBPMN2 example. . . . .	115
3.5	The proposed integrated management framework. . . . .	118
3.6	Henshin rule for adding « <i>abacRequire</i> » UMLsec stereotype. . . . .	124
3.7	Henshin rule for adding UML nodes. . . . .	125
3.8	Henshin rule for adding « <i>encrypted</i> » to communication paths. . . . .	126
3.9	Henshin rule for adding « <i>integrity</i> » to dependencies. . . . .	127
3.10	Henshin rule for adding « <i>encrypted</i> » to communication paths correspond with SecBPMN2 message flows. . . . .	128
3.11	Henshin rule for adding « <i>secure links</i> » stereotype. . . . .	128
3.12	Henshin rule for importing core UML classes. . . . .	129
3.13	Henshin rule for transforming SecBPMN2 data object to UML class. . .	130
3.14	Henshin rule for transforming Accountability-annotated task. . . . .	132
3.15	Henshin rule for adding « <i>secure dependency</i> » stereotype. . . . .	133
3.16	Process with involved tasks and artifacts. . . . .	134
3.17	Example of the generated trace models (excerpt). . . . .	138
4.1	The highlighted part (dashed lines) denotes how Chapter 4 contributes to the overall workflow of the MoPrivFair methodology. . .	147
4.2	Example Model: Excerpt from the class and the state machine diagrams of the school software. . . . .	152
4.3	The semi-automated, model-based process for analyzing individual fairness . . . . .	154
4.4	Excerpt from the generated trace of events. . . . .	168



---

## List of Tables

2.1	Definitions of the considered data-minimization concepts in our work.	25
2.2	Definitions of the considered fairness concepts in our work. . . . .	27
2.3	Mapping data protection threats to BPMN elements . . . . .	47
2.4	Overview of conflict + potential conflict anti-patterns per pair of requirements. . . . .	68
2.5	Experience levels of participants in our experiment. . . . .	94
2.6	Execution time of conflict detection technique. . . . .	98
3.1	UMLsec adversary patterns. . . . .	113
3.2	SecBPMN2 elements to UML elements. . . . .	120
3.3	SecBPMN2 security annotations to UMLsec security policies. . . . .	122
3.4	Detailed Summary for the transformation results . . . . .	140
3.5	The overall transformation results and the execution time. . . . .	141
4.1	The UMLfair profile . . . . .	158
4.2	The verification results of the LTL claims . . . . .	167
4.3	Student data . . . . .	170
4.4	Correlations results . . . . .	171
4.5	An overview about the analyzed UML models . . . . .	173
4.6	The Detected Individual-Fairness Violations . . . . .	174



# Chapter 1

## Introduction

The seemingly never-ending collection of data by many of today's systems and organizations and the advances in the amount of storage and processing power have raised public and legal awareness on *security*, *privacy* [32] and *data-misuse* concerns [30]. While security-specific data protection goals are focusing on preventing unauthorized data access and data manipulation that could undermine trusted systems, people, however, have relatively low confidence that organizations and IT systems will use their data in a responsible way. For example, according to PricewaterhouseCoopers's (PwC) 2017 report<sup>1</sup>, only 25% of customers believe that companies handle sensitive personal data responsibly. Even fewer—only 15%—believe companies will use these data to improve their lives.

The risk that personal data might be misused is Europeans' top worry, according to a 2017 European Union public opinion survey<sup>2</sup>. Restrictions on data collection, in the first place, and data usage play a central role in the European Union's General Data Protection Regulation (GDPR, [2]). The GDPR aims, as specified in *Article 1(2)*, to protect the fundamental rights and freedoms of natural persons and their right to the protection of personal data.

Apart from security, two key data protection goals are *data-minimization* [53, 139] and *fairness* [7, 27], which aim at mitigating the risks of unauthorized revealing of data subjects' real identities and discrimination, respectively. The clearest

---

<sup>1</sup>The PwC report is available online at <https://www.pwc.com/us/en/advisory-services/publications/consumer-intelligence-series/protect-me/cis-protect-me-findings.pdf> (accessed: 06/12/2019).

<sup>2</sup>The survey is available online at <https://ec.europa.eu/commfrontoffice/publicopinion/index.cfm/Survey/getSurveyDetail/instruments/SPECIAL/surveyKy/2171> (accessed: 06/12/2019).

mention for these two risks occurs in *Recital 75* of the GDPR, which states that "*the risk to the rights and freedoms of natural persons, [...], may result from personal data processing which could lead to physical, material or non-material damage, in particular: where the processing may give rise to **discrimination**, [...], unauthorized **reversal of pseudonymization**, [...]*". Pseudonymization is a data-minimization mechanism where a pseudonym can be used as an identifier for a data subject other than one of the data subject's personal identifiable information. Based on *Recital 28* of the GDPR, the explicit introduction of "pseudonymization" in the GDPR is not intended preclude any other measures of data protection.

*Data-minimization* aims at minimizing "*the possibility to collect personal data about others*" and "*within the remaining possibilities, [to minimize] collecting personal data*" (Pfitzmann et al. in [103], p.6). *Article 5(c)* of the GDPR refers to data-minimization as one of the main principles of processing personal data. Addressing data-minimization is of vital importance to support user's privacy and to avoid legal concerns. According to *Article 25(1)*, organizations and IT systems need to implement appropriate technical and organizational measures, such as pseudonymization, which are designed to implement data-minimization, in an effective manner.

*Fairness* aims to ensure equal treatment between data subjects by preventing the misuse of data in decision-making processes to discriminate data subjects on the ground of *protected characteristics* such as gender and ethnicity [14]. Worth noting that fairness in this context is mentioned in the GDPR in the sense of non-discrimination. For instance, *Recital 71* states a requirement to "*implement technical and organizational measures appropriate to [...], and prevent, inter alia, discriminatory effects on natural persons on the basis of racial or ethnic origin, political opinion, [...]*". These characteristics are referred to in *Article 9* of the GDPR as "*special categories*". Moving beyond the *Recitals*, algorithmic discrimination is addressed by *Article 22*, which prevents decision making based on special categories of personal data.

In today's systems, security, data-minimization, and fairness are more vital and intertwined than ever before. Beyond many security-specific concepts such as confidentiality and integrity, five data-minimization concepts, namely *Pseudonymity*, *Anonymity*, *Unlinkability*, *Undetectability* and *Unobservability*, and two fairness concepts, namely *Individual-Fairness* and *Group-Fairness*, are considered fundamental to avoid privacy and fairness violations, respectively. Data-minimization concepts were first defined by Pfitzmann et al. [103] and later included in the ISO 15408 standard of common criteria for information technology security evaluation [60]. The two fairness concepts are formally explored by many research works in the algorithmic fairness field (e.g., [7, 45, 49]).

There exist a range of protection mechanisms and approaches [146, 157] that address specific security, data-minimization and fairness needs. However, data pro-

---

tection violations often do not come from loopholes in the applied protection technologies [54], but rather from three distinct sources: first, *conflicts between security, data-minimization and fairness protection requirements* [50, 52]. For example, in health-care, users have strong privacy concerns about how and for what purpose their health information is handled, which may interfere with an organization’s documentation responsibilities for ensuring complete accountability. Second, *a non-alignment between the organizational and technical data protection requirements* [11, 118]. Data protection requirements for the organizational and technical aspects of a system are currently dealt with separately, giving rise to substantial misconceptions and errors. Third, *hidden dependencies between the data in the system that might lead to influence unappropriated biases against individuals based on protected characteristics* [49, 112]. For instance, most recently, it is reported that algorithms used to set credit limits in Apple’s credit card might be inherently biased against women<sup>3</sup>.

For the effective assurance of data protection needs and for reducing difficulties of finding the sources of vulnerabilities in the later stages of system development, it is important to avoid the above three issues right from the start of the development process. In many application domains (including finance, health, automotive etc) there exist regulations which require detailed documentation of the software to support the needed certification, and this requirement can be fulfilled using models [141]. Using process and software models, the complexity of systems can be handled through abstraction, enabling their analysis and optimization [47]. Although the use of process and software models in practice varies between different domains [141], they have been considered as a key enabler for important tasks of high business value, such as security and privacy analysis [47]. In fact, there has been a significant amount of model-based analysis approaches that aims to incorporate data protection requirements, specifically, security and privacy requirements, into the design phase of the system development life cycle. Approaches in this direction can be classified into *business process model-based* approaches [82] and *software model-based* approaches [78].

Business process model-based approaches rely on business process modeling for organizational data protection requirements (e.g., [17, 89, 119, 126]). These approaches abstract from technical details to allow the specification of high-level data protection requirements by non-technical stakeholders, such as business analysts. In contrast, software model-based approaches allow the developers to design architectural and behavioral aspects of the system while considering low-level technical details to support the validation against pre-defined technical data protection policies (e.g., [5, 62, 81]). The software model is a cornerstone for further development stages, such as generating code for the implementation. Hence, software models can be seen as an approximation to the actual execution semantics of the

---

<sup>3</sup>Information about the discriminatory behavior of Apple’s algorithms is available online at <https://www.bbc.com/news/business-50365609> (accessed: 06/12/2019)

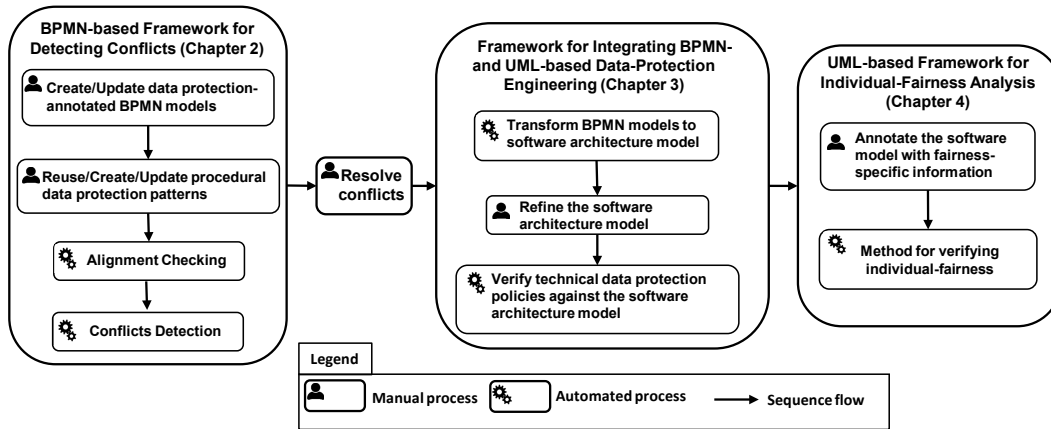
developed software. There is also much work on model-and-code synchronization (e.g., [19]). Therefore, data-protection analysts can benefit from the software models at the run time of a system [15]. In legacy systems, data-protection analysts can generate business process models from the log files of the system by using a process mining technique (e.g., [132]) while software models can be generated in a reverse engineering process from the source code of the system (e.g., [20]).

Despite the variety of model-based approaches that provide data protection analysis in the system design phase, there is no *model-based* approach that permits any of the following challenges: (i) detecting conflicts between data protection requirements; (ii) establishing traceability between high-level (i.e., organizational) data protection requirements and verifiable technical data protection policies; (iii) and analyzing individual fairness. In the following, we provide a brief overview of the state of the art with respect to the considered challenges above.

*Conflict Detection.* A few existing approaches are available to deal with different types of data protection requirements in the early stages of development (e.g., [16, 32, 65, 95]). These approaches focus on the identification of security and data-minimization requirements in the elicitation phase without detecting conflicts between them. Ganji et al. [50] and Alkubaisy [10] highlight the importance of detecting conflicts between security and privacy requirements, specifically for data-minimization requirements. Both papers discuss the components required for a potential approach, however, without providing a complete solution. Furthermore, to our knowledge, no approach supports fairness requirements in the early development stages. Hence, an approach that permits detecting conflicts between security, data-minimization and fairness requirements in the early stages of the system development is currently not available.

*Integration and Traceability.* The fact that different modeling languages are extended to cover data protection aspects such as security and privacy, attracted many researchers to study the traceability of these aspects via the integration between these languages through the use of automated transformation support. However, earlier automated transformation approaches used Unified Modeling Language (UML, [3]) as sole modeling language throughout the entire process [55], thereby leaving the role of business analysts unaddressed, or focused on representing data protection-related requirements "*at the business analyst views*" ([120], p.2), leaving technical data protection concerns and the verification of data protection aspects to future work [120]. Therefore, support for integrated management of organizational and technical data protection requirements is generally missing.

*Fairness Analysis.* Existing fairness analysis approaches do not analyze the system *ex-ante* but *ex-post*, namely during the testing phase of the system development life cycle (e.g., [29, 49]) or at the run-time of the system (e.g., [7, 8]). Considering the



**Figure 1.1:** The MoPrivFair (**Model-based Privacy & Fairness**) methodology.

fairness of a software system after implementing it raises substantial difficulties in identifying and explaining the discriminatory behavior of the software. Despite the availability of many model-based approaches [68], we have not found an existing approach described in the literature that would use software models for fairness analysis. The need for fairness analysis based on software models has been motivated by Brun et al. [18]. However, the work in [18] is not supported by an approach, that realizes the idea of analyzing fairness based on software models.

In this thesis, we propose a model-based methodology called *MoPrivFair* (**Model-based Privacy & Fairness**) that addresses the three challenges above. Figure 1.1 provides an overview of the MoPrivFair methodology. As shown in Figure 1.1, our proposed MoPrivFair methodology comprises three sub-frameworks:

- First, a *BPMN-based framework for detecting conflicts* between data protection requirements. The goal of this framework is to analyze business process models annotated with security, data-minimization and fairness requirements to detect conflict among them.
- Second, a *framework for integrating BPMN- and UML-based data-protection engineering*. This framework aims at specifying data protection-aware software architecture models based on business process models annotated with data protection requirements.
- Third, a *UML-based framework for fairness analysis*. The goal of this framework is to analyze UML-based software designs in order to detect discrimination between individuals on the basis of their protected characteristics.

In the MoPrivFair methodology, conflicts between data protection requirements have to be resolved before any further development. Conflicts resolution tech-

niques are varied and might require negotiation with different users of the system and trade-off analysis [42, 57]. Discussion about conflict resolution is, therefore, out of the scope of this thesis. The focus of Figure 1.1 is to illustrate our main contribution and the overall work-flow of our work in this thesis. The MoPrivFair methodology is explained in more detail in Section 1.2.

## 1.1 Challenges and Research Directions

We identify and address three challenges in this thesis:

- **Conflict detection:** The variety of requirements arising from security, data-minimization, and fairness considerations gives rise to various types of conflicts. Importantly, conflicts between data protection-related requirements if propagated to the final system can lead to severe effects, including user dissatisfaction, privacy infringement, and legal sanctions. Detecting conflicts between security, data-minimization, and fairness requirements is a challenging task, as such conflicts are context-specific and their detection requires a thorough understanding of the underlying business processes. Therefore, conflicts should be detected and reported as early as during the design of the business process models of the targeted system.
- **Integration and traceability:** In the system development process, data protection requirements for the organizational and technical aspects of a system are currently dealt with separately, giving rise to substantial misconceptions and errors. Specifically, existing business process model- and software model-based data protection-oriented approaches support engineering data protection-aware systems in distinct development phases and from distinct stakeholders' perspectives. Therefore, vulnerabilities may arise from misunderstandings between these stakeholders. The raised vulnerabilities may be notoriously hard to detect due to the lack of a traceability mechanism for data protection requirements across the different modeling phases of the system.
- **Fairness analysis:** A system may have undesirable hidden data-flows that might indirectly leak personal protected characteristics such as *gender*, and as a result, influence biases in an automated decision-making process against individuals based on the leaked protected characteristic. Considering the fairness of a system after implementing it raises substantial difficulties in identifying and explaining the discriminatory behavior of the system. To reduce difficulties in the later stages of the system development process, it is important to deal with fairness from the early stages of system development.

The three main research directions of this thesis span over these challenges. In the following sections, we introduce these research directions and formulate the research questions of this thesis.



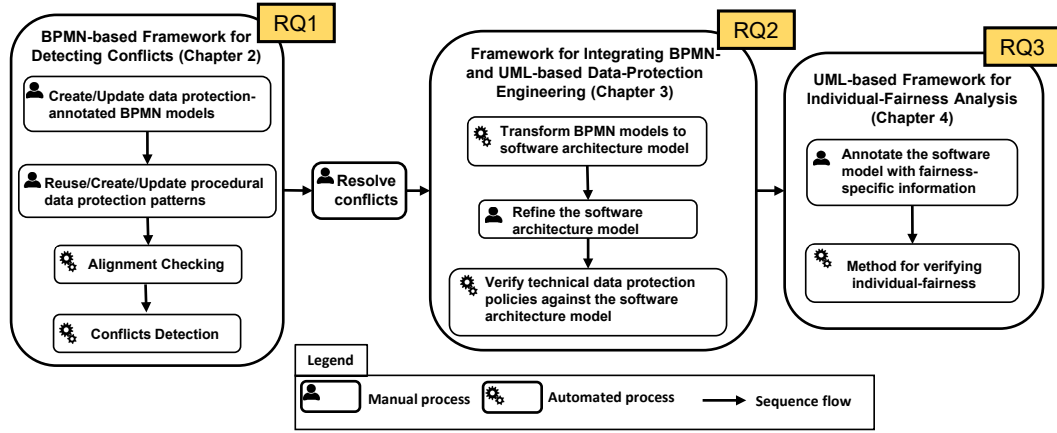
### 1.1.1 Conflict detection

Requirements are prone to conflicts [48, 148]. Data protection requirements such as security, data-minimization and fairness requirements are no exception [50, 52, 54, 95]. The need to account for multiple types of data protection requirements at once has been known to result in conflicting requirements. According to Mouratidis et al. [95], "*The successful analysis of security and privacy individually does not necessarily lead to the successful cooperation*". The same can be said about the analysis of fairness and other data protection requirements individually. For instance, according to recent research from Google AI [52], one of the main challenges in the field of algorithmic fairness is conflicts with data-minimization requirements.

Kim et al. [69] define requirements conflicts as "*The interactions and dependencies between requirements that can lead to negative or undesired operation of the system*". According to Easterbrook [38], two main sources of such conflicts are *clashes between system users' needs* and *misinterpretation of requirements*. Clashes between system users' needs represent trade-offs between users' and organization' needs. For example, data subjects may require that an activity with a service evaluation purpose should be executed in a full-anonymous way. This requirement may interfere with an organizations' needs. In some scenarios, it is important to store information about the executors of service-evaluation activities for accountability issues. Misinterpretation of the elicited requirements can happen intentionally or unintentionally while implementing the requirements in business processes. Misunderstanding the specifications of the elicited requirements or intentionally deviating from them due to business needs will result in a non-alignment between the business processes and the requirements [129]. These deviations may result in unexpected conflicts between the requirements that may lead to unwanted vulnerabilities.

Detecting conflicts between data protection requirements is a challenging task. Since such conflicts are context-specific, their detection requires a thorough understanding of the underlying business processes [50]. Specifically, conflicts not only result from trade-offs between requirements related to the same asset in the system (e.g., anonymous vs. accountable execution of a task), but also from those related to different assets. For example, a task may be required to be executed anonymously, while writing data to a secure data storage where the identity of the writer must be known for accountability reasons.

Importantly, undetected conflicts will have serious effects on the remainder of the system development process. A few existing approaches are available to deal with different types of data protection requirements in the early stages of development [16, 32, 65, 95]. These approaches focus on the identification of security and data-minimization requirements in the elicitation phase without detecting conflicts be-



**Figure 1.2:** Mapping between the research questions and the frameworks of the MoPrivFair methodology.

tween them. Furthermore, to our knowledge, no approach supports fairness requirements in the early development stages. Hence, an approach that permits detecting conflicts between security, data-minimization and fairness requirements in the early stages of the system development is currently not available. Considering support for conflict detection between these requirements during the design of a system business process models, we investigate the following research question:

**RQ1. How to detect conflicts between security, data-minimization, and fairness requirements by analyzing business process models?**

Our objective is to help business process designers in detecting conflicts between security, data-minimization and fairness requirements during the design of the business process models of the system in question. The answer to RQ1 requires: (i) an appropriate graphical modeling language for designing business processes annotated with security, data-minimization and fairness requirements; (ii) an alignment checking technique to avoid conflicts that may result from possible non-alignments between the users' data protection requirements and their specifications in business process models. (iii) a conflict detection technique that permits to report and visualize conflicts between specified security, data-minimization and fairness requirements in business process models.

Figure 1.2 shows which parts of the proposed MoPrivFair methodology are supported by which research question. As shown in left part of Figure 1.2, the MoPrivFair methodology addresses the research question RQ1 by a *semi-automated* BPMN-based framework for detecting conflicts between security, data-minimization and fairness requirements. This contribution is explained in more detail in Section 1.2.

### 1.1.2 Integrating business process and software modeling

There has been a significant amount of model-based approaches that aim to support the management of organizational and technical data protection requirements in the design phase of the system development process. *Business process-based* approaches are abstract from technical details to support business analysts in specifying organizational data protection requirements as part of business processes for the target system (e.g., SecBPMN2 [129]). Different from business process model-based approaches, *software model-based* approaches support system developers in specifying technical data protection requirements and data protection assumptions in architectural models. The resulting models can be validated against pre-defined technical data protection-related policies (e.g., UMLsec [62]).

Since these existing approaches address data protection requirements in distinct development phases and from the perspectives of different stakeholders, they deal with data protection requirements separately; therefore, an alignment of data protection requirements across the business process- and software architecture-models is not guaranteed [11, 118]. For building systems that preserve data protection requirements, it is important to manage data protection requirements consistently, so that the introduction of vulnerabilities during the development process is avoided [152]. A main source of vulnerabilities are misunderstandings between expert stakeholders, as triggered by their implicit knowledge about terminology. According to Yu et al. [152], "*a single inaccurate requirement traceability link assumed by developers may already be exploited by malicious attackers*". Ensuring traceability of data protection requirements between representations of the system in different abstraction levels is therefore important.

Model transformation [134] is a promising direction to address this problem. Generally, models are used to describe the system in question from different viewpoints and at different levels of abstraction. The use of different kinds of models during the system design phase leads to the challenge of keeping the models consistent with each other. At this point, model transformation techniques play a central role. Thus we need a model transformation from business process models to software models which ensures traceability of data protection requirements.

Earlier automated transformation approaches used Unified Modeling Language (UML, [3]) as sole modeling language throughout the entire process [55], thereby leaving the role of business analysts unaddressed, or focused on representing data protection-related requirements "*at the business analyst views*" ([120], p.2), leaving technical data protection concerns and the verification of data protection aspects to future work [120]. Therefore, support for integrated management of organizational and technical data protection requirements is generally missing. Concerning this, we investigate the following research question:

**RQ2. How to support the integrated management of modeling business process and software architecture models, such that the traceability of data protection requirements is guaranteed?**

Our objective is to support the management of data protection requirements from the views of the involved expert stakeholders, in particular, business analysts and systems engineers, consistently in an integrated manner to: First, avoid the introduction of vulnerabilities during the development process due to the conceptual gap between the involved stakeholders. Second, ensure traceability for data protection requirements across business process- and software architecture-models.

RQ2 is about how to support the traceability of the data protection requirements in the system design phase. The answer to this research question requires: (i) mapping the business process model elements to software architecture model elements; (ii) mapping high-level data protection concepts in the business process level to verifiable technical data-protection policies in the software architecture; (iii) a *model transformation* supporting the automatic translation of business process models annotated with data protection requirements to software architectural models while establishing traceability for the data protection requirements.

Integrating and tracing data protection requirements from business process models to software architecture is a complicated task. Business processes are mainly about behavior and tasks which specify how a system achieves its goals, while software architecture refers to the structural aspects of the system, which specify how the system should be built to achieve its goals. Therefore, it is not possible to automatically specify all the architectural details based on a given business process. Therefore, as shown in the middle part of Figure 1.2, our proposed MoPrivFair methodology addresses the research question RQ2 by a *semi-automated* BPMN-based framework for integrating BPMN- and UML-based data-protection engineering. This contribution is explained in more detail in Section 1.2.

### 1.1.3 Fairness analysis

Automated decision-making software became responsible for sensitive decisions with far-reaching societal impact in many areas of life. However, the risk that a falsely developed decision-making software may lead to unlawful discrimination against persons, illegally exploiting their *protected characteristics* has raised public and legal awareness of software *fairness* [140, 153]. For example, Article 22, Paragraph 4 of the European General Data Protection Regulation (GDPR, [2]) forbids decisions based on special personal characteristics as defined in Article 9 of the same regulation, such as ethnicity, religion, and gender.

Existing fairness analysis approaches do not analyze the system *ex-ante* but *ex-post*, namely during the testing phase of the system development life cycle (e.g., [29, 49]) or at the run-time of the system (e.g., [7, 8]). Considering the fairness of a software system after implementing it raises substantial difficulties in identifying and explaining its discriminatory behavior. To avoid complicated explanations and expensive fixes, fairness awareness has to be pro-actively embedded in the design phase of the system development, similar to other data protection requirements such as security and privacy [18]. Despite the significant amount of work that uses software models to support reasoning about critical issues such as security [68], so far there is no approach that uses software models to support fairness analysis.

A distinguished type of fairness is called *individual fairness*. A decision-making software preserves the individual fairness if it produces the same decision for every two individuals whose data, that are given as input to the decision-making software, are identical except for the protected characteristics [49]. Considering support for individual fairness at the system design phase, a *model-based* approach that permits detecting discrimination during the design of system models is lacking. Thereby, we investigate the following research question:

**RQ3. How to detect undesired discriminatory behavior, that violates individual fairness, by analyzing software design models?**

Our objective is to reduce the effort needed for uncovering discrimination in the later stages of the development. Answering RQ3 requires: (i) an extension to a software modeling language to permit annotating software models with fairness information; (ii) a method for verifying individual fairness of annotated models. As shown in the right part of Figure 1.2, our MoPrivFair methodology addresses the research question RQ3 by a *semi-automated* UML-based framework for individual fairness analysis. This contribution is explained in more detail in Section 1.2.

## 1.2 Research Contribution

In the previous section, we have outlined the research questions that are related to: First, detecting conflicts between data protection requirements. Second, integrating business process and software architecture modeling while supporting traceability for data protection requirements. Third, fairness analysis at the system design level. In this thesis, we address the proposed research questions by a model-based methodology called MoPrivFair (**Model-based Privacy & Fairness**).

As shown in Figure 1.2, the proposed MoPrivFair methodology addresses our research questions with a three-fold contribution:

- a BPMN-based data-protection engineering framework for detecting conflicts between data protection requirements (Chapter 2).
- a framework for integrating BPMN- and UML-based data-protection engineering while supporting traceability (Chapter 3).
- a UML-based framework for analyzing individual fairness during the system design level (Chapter 4).

In the MoPrivFair methodology, detected conflicts between data protection requirements have to be resolved before any further development. Conflicts resolution techniques are varied and may require human intervention [42, 57]. However, as mentioned earlier in this chapter, discussion about conflict resolution is out of the scope of this thesis. In the following, we briefly describe our main contribution:

**First, a BPMN-based framework for detecting conflicts.** To address the problem of detecting conflicts between security, data-minimization and fairness requirements in RQ1, we propose (as shown in the left part of Figure 1.2) a *semi-automated* BPMN-based data-protection engineering framework that supports:

- (i) the design of business processes considering security, data-minimization and fairness requirements;
- (ii) the encoding of such requirements as reusable, *domain-specific* patterns;
- (iii) the checking of alignment between the encoded requirements and annotated BPMN models based on these patterns; and
- (iv) the detection of conflicts between the specified requirements in the business process models based on a catalog of *domain-independent* anti-patterns.

The security annotations in our BPMN extension were reused from an existing security-oriented BPMN extension called SecBPMN2 [129]. To express the domain-specific patterns and domain-independent anti-patterns, we extended a graphical query language for BPMN 2.0 models called SecBPMN2-Q [129]. While the security annotations were reused from the SecBPMN2, our proposed framework is novel because it is the first to directly support modeling data-minimization and fairness requirements in BPMN models. It is also the first to support automatic conflict detection between specified security, data-minimization and fairness requirements in BPMN models. We report on the feasibility and the usability of our conflict detection technique based on a case study featuring a healthcare management system, and an experimental user study during the design of the business process models.

**Second, a framework for integrating BPMN- and UML-based data-protection engineering.** To address RQ2 that is concerned with integrating BPMN- and UML-based data-protection engineering while supporting traceability for data protection requirements, we propose (as shown in the middle part of Figure 1.2) a *semi-automated* framework that suggests to iteratively:

- (i) transform business process models enriched with organizational data protection requirements using the SecBPMN2 to a preliminary software architectural model enriched with data protection policies using UMLsec [62];
- (ii) refine the generated UMLsec architecture model manually with additional design decisions; and
- (iii) verify the resulting UMLsec architecture model against their contained data protected policies by using an automated tool called CARiSMA [4].

The novelty of our framework is that we automatically establish traceability between high-level data protection requirements and verifiable technical data protection policies. We report on the applicability of our framework based on a case study featuring an air traffic management system.

**Third, a UML-based framework for analyzing individual fairness.** To address RQ3 that is concerned with reasoning on individual fairness by analyzing the software models, we present (as shown in the right part of Figure 1.2) a *semi-automated* framework that supports the analysis of UML-based software designs with regard to individual fairness. The analysis in our framework is established by generating temporal logic claims, whose verification against the targeted software model enables reporting on the individual fairness of the software. Our framework for individual fairness analysis includes the following contributions:

- (i) a UML profile *UMLfair* for annotating UML software models with fairness-specific information; and
- (ii) a method for verifying individual fairness of annotated UML models.

Given a UML model annotated with fairness-specific information, our method for reasoning about individual fairness includes: First, generating temporal logic claims from the UML model. Second, reporting on individual fairness based on the verification results of the generated claims. Our framework for individual fairness analysis is novel in the sense that it is the first that permits fairness analysis based on software models at system design time. We applied our framework to three case studies featuring a school management system, a delivery management system and a loan management system.

### 1.3 Research Methodology

The research methodology guiding this thesis is explained in the taxonomy of software engineering proposed by Shaw [135]. Shaw distinguishes five approaches to address a software engineering problem, namely *qualitative or descriptive model, techniques, systems, empirical predictive model, and analytic model*.

Each of the three main contributions put forward by this thesis is supported by a *technique* supported by an *analytic model*:

**"Technique:** Invent new ways to do some tasks, including procedures and implementation techniques.

**Analytic model:** Develop structural (quantitative or symbolic) models that permit formal analysis." ([135], p.660)

The proposed MoPrivFair methodology provides novel automated techniques to: (i) conduct conflict detection between data protection requirements on the business process modeling level, (ii) integrate BPMN- and UML-based data protection engineering while establishing traceability for data protection requirements, and (iii) analyze software models with regard to individual fairness.

Our techniques based on standard graph-oriented modeling languages, namely, the Business Process Modeling Notation (BPMN) [1] and the Unified Modeling Language (UML) [3]. The semantics of these languages are formally specified by algebraic graph transformations [35, 74]. Thereby, our techniques are amenable to a formal analysis for evaluating their correctness.

Furthermore, in [135], Shaw distinguishes five kinds of validation techniques to show that a research result satisfies the requirements posed by the motivating problem: *persuasion, analysis, implementation, evaluation, and experience*.

In this thesis, we apply all of these validation techniques. Throughout the thesis, *persuasion* is used to motivate our design choices and rationales. In our contributions, we use practical case studies to *analyze* and *evaluate* the applicability of our techniques. In first contribution, a user experiment is used to study the usefulness of our conflict detection technique. For all of our contributions, we provide and discuss implementations. All of our contributions are validated using narrative demonstrations exemplifying potential experiences of applying the techniques.



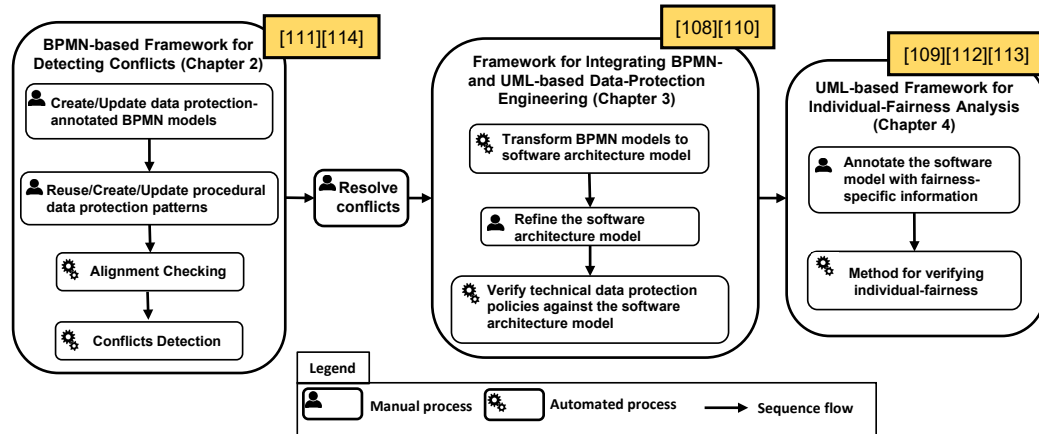
## 1.4 Thesis Outline

In the next chapters, we elaborate on the sub-frameworks of our proposed Mo-PrivFair methodology, their tool support, and their evaluation. Specifically, the remaining chapters are structured as follows.

- In Chapter 2, we propose a semi-automated BPMN-based data protection engineering framework for detecting conflicts between security, data-minimization and fairness requirements. We first introduce the key data protection-related concepts that are used in this chapter. We also present the SecBPMN2 and the SecBPMN2-Q on which our contributions are built. We then introduce an extension of BPMN that permits the specification of these requirements in business process models, based on existing security annotations from the SecBPMN2 [129] and new data-minimization and fairness annotations. Afterward, we demonstrate our conflict detection framework by example, discuss tool support, and validate the applicability and usability of our conflict detection techniques based on a case study featuring a healthcare management system, and an experimental user study, respectively.
- In Chapter 3, we propose a semi-automated framework for supporting integration management between BPMN- and UML-based data protection engineering while establishing traceability. We first present the research work and the technologies on which our contributions are built. In this chapter, data protection requirements are captured via the language extensions SecBPMN2 [129] and UMLsec [62]. We then provide a model transformation to bridge the conceptual gap between SecBPMN2 and UMLsec. We demonstrate our integration-management framework by example, discuss tool support, and validate show how our framework can be practically applied based on a case study featuring an air traffic management system.
- In Chapter 4, we propose a model-based framework that permits an analysis for individual fairness. The framework is based on UML system design models. We first introduce a UML profile *UMLfair* that extends the UMLsec profile [62] to permit annotating the system design models with fairness-specific information. Afterward, we demonstrate our framework for individual fairness by example, discuss tool support, and validate the applicability of our work based on three case studies featuring a school management system, a delivery management system and a loan management system.
- In Chapter 5, we summarize and conclude this thesis. We also give an outline on possible future research directions.

## 1.5 List of Publications

This thesis shares material with seven research papers written by the author of this thesis. These research papers are listed below. The co-authors have explicitly confirmed the individual contributions of the author of this thesis to these papers<sup>4</sup>. Figure 1.3 shows a mapping between these research papers and the frameworks of the proposed MoPrivFair methodology.



**Figure 1.3:** Mapping between the research papers that are written by the author of this thesis and the frameworks of the MoPrivFair methodology.

- [114] *Qusai Ramadan, Daniel Strüber, Mattia Salnitri, Jan Jürjens, Volker Riediger, Steffen Staab. A Semi-Automated BPMN-based Framework for Detecting Conflicts between Security, Data-Minimization and Fairness Requirements. To appear* in a special journal issue of SoSyM (Software and Systems Modeling). This paper is an extension to our ECMFA'18 paper, which was invited for submission to a special journal issue in SoSyM for extended versions of best papers at ECMFA'18.
- [113] *Qusai Ramadan, Marco Konersmann, Amir Shayan Ahmadian, Jan Jürjens, Steffen Staab. Analyzing Individual Fairness based on System Design Models. Submitted, 2019.*
- [112] *Qusai Ramadan, Amir Shayan Ahmadian, Jan Jürjens, Steffen Staab, Daniel Strüber. Explaining Algorithmic Decisions with respect to Fairness. In: SE/SWM 2019: Multikonferenz Software Engineering und Management, Special Track on Explainable Software. pp. 161-162.*

<sup>4</sup>The signed confirmations are submitted to the PhD committee.

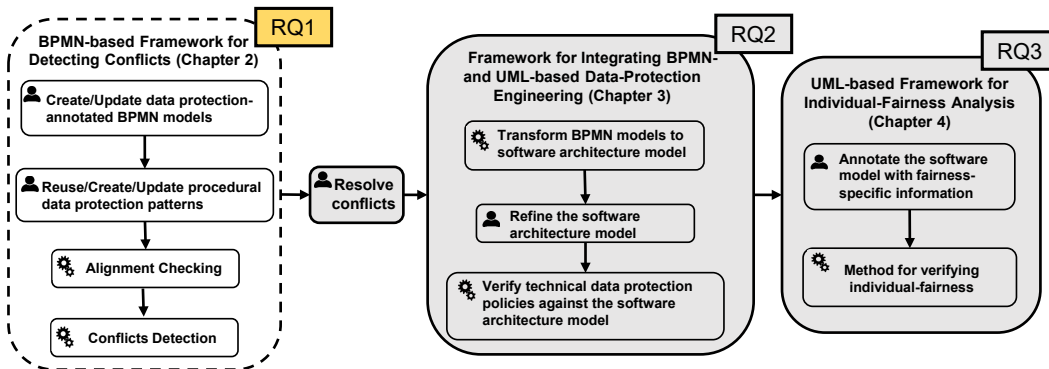
- 
- [111] *Qusai Ramadan*, Daniel Strüber, Mattia Salnitri, Volker Riediger, Jan Jürjens. **Detecting Conflicts Between Data-Minimization and Security Requirements in Business Process Models**. In: ECMFA 2018: European Conference on Modeling Foundations and Applications. Springer. pp. 179-198.
- [109] *Qusai Ramadan*, Amir Shayan Ahmadian, Daniel Strüber, Jan Jürjens, Steffen Staab. **Model-based discrimination analysis: a position paper**. In: FairWare@ICSE 2018: IEEE/ACM International Workshop on Software Fairness. IEEE/ACM. pp. 22-28.
- [110] *Qusai Ramadan*, Mattia Salnitri, Daniel Strüber, Jan Jürjens, Paolo Giorgini. **Integrating BPMN- and UML-based Security Engineering via Model Transformation**. In: SE 2018: Fachtagung des GI-Fachbereichs Softwaretechnik. Gesellschaft für Informatik. pp. 63-64.
- [108] *Qusai Ramadan*, Mattia Salnitri, Daniel Strüber, Jan Jürjens, Paolo Giorgini. **From Secure Business Process Modeling to Design-Level Security Verification**. In: MODELS 2017: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. IEEE. pp. 123-133.



## Chapter 2

# BPMN-based Framework for Detecting Conflicts between Data Protection Requirements

This chapter presents a sub-framework of the proposed MoPrivFair (*Model-based Privacy & Fairness*) methodology in this thesis. An overview of the MoPrivFair methodology is provided in Section 1.2 in the first chapter of this thesis.



**Figure 2.1:** The highlighted part (dashed lines) denotes how Chapter 2 contributes to the overall workflow of the MoPrivFair methodology.

Requirements are inherently prone to conflicts [48, 148]. Data protection requirements such security, data-minimization and fairness requirements are no exception. In this chapter, we address the problem of detecting conflicts between security, data-minimization and fairness requirements early during the business process modeling to avoid difficulties of detecting conflicts in later stages of system

development<sup>1</sup>. According to Easterbrook [38], "*Failure to recognize conflict between the perspectives of the participants will cause confusion throughout the lifecycle. Participants' understanding of the specification will differ, leading to further misunderstandings during design and implementation*".

Specifically, in this chapter, we propose a BPMN-based data protection engineering framework that supports: (i) the design of business processes considering security, data-minimization and fairness requirements, (ii) the encoding of such requirements as reusable, domain-specific patterns, (iii) the checking of alignment between the encoded requirements and annotated collaboration BPMN 2.0 models based on these patterns, and (iv) the detection of conflicts between the specified requirements in the BPMN models based on a catalog of domain-independent anti-patterns. The security requirements were reused from SecBPMN2, a security-oriented BPMN 2.0 extension, while the fairness and data-minimization parts are an original contribution. In order to formulate patterns and anti-patterns, we extended a graphical query language called SecBPMN2-Q. We report on the feasibility and the usability of our approach based on a case study featuring a healthcare management system, and an experimental user study.

## 2.1 Introduction

The seemingly never-ending collection of customers' data by many of today's systems and organizations and the advances in the amount of storage and processing power have raised public awareness on *privacy* [32] and *data-misuse* concerns [30]. Therefore, protecting the privacy of users and preventing data-misuse concerns become a key activity in companies and governmental organizations.

Apart from security, key data protection concepts are *data minimization* [53, 139] and *fairness* [7, 27]. Data-minimization aims at minimizing "*the possibility to collect personal data about others*" and "*within the remaining possibilities, [to minimize] collecting personal data*" (Pfitzmann et al. in [103], p.6). Fairness aims to ensure equal treatment between data subjects by preventing the misuse of data in decision-making processes to discriminate data subjects on the ground of personal *protected characteristics* as defined by laws or organizational policies [14]. For example, *Article 22(4)* of the European General Data Protection Regulation (GDPR, [2]) prohibits decision making based on *protected characteristics* as defined in *Article 9* of the same regulation, such as ethnicity, religion, and gender.

---

<sup>1</sup>This chapter shares material with the paper accepted for publication in the journal SoSyM "A Semi-Automated BPMN-based Framework for Detecting Conflicts between Security, Data-Minimization and Fairness Requirements" [114] and the ECMFA'18 paper "Detecting Conflicts between Data-Minimization and Security Requirements in Business Process Models"[111].

As mentioned in Chapter 1, beyond traditional security concepts such as confidentiality and integrity, five data-minimization concepts, namely *Pseudonymity*, *Anonymity*, *Unlinkability*, *Undetectability* and *Unobservability*, and two fairness concepts, namely *Individual-Fairness* and *Group-Fairness*, are considered fundamental data protection concepts to avoid privacy and fairness threats, respectively. Data-minimization concepts were first defined by Pfitzmann et al. [103] and later included in the ISO 15408 standard of common criteria for information technology security evaluation [60]. The two fairness concepts are formally explored by many research works in the algorithmic fairness field [7, 45, 49].

While privacy-enhancing technologies [146] and algorithmic fairness techniques [157] respectively address specific data-minimization and fairness needs, security, privacy and fairness violations often do not come from loopholes in the applied protection technologies [54], but from conflicts between data protection interests at the business level of the target system [10, 50, 52–54, 92]. For example, according to recent research from Google AI [52], one of the main challenges in the field of the algorithmic fairness is conflicts between data-minimization and fairness requirements. Importantly, undetected conflicts between such requirements can lead to severe effects, including privacy infringement and legal sanctions.

Detecting conflicts between security, data-minimization, and fairness requirements is a challenging task, as such conflicts are context-specific and their detection requires a thorough understanding of the underlying business processes. The variety of requirements arising from security, data-minimization, and fairness considerations gives rise to various types of conflicts. According to Easterbrook [38], two main sources of such conflicts are:

- *Clashes between the system users' needs.* Data subjects may require that any activity with a decision-making purpose should not be able to distinguish whether their protected characteristics exist or not in the data store from where the activity retrieves data (*Undetectability*). This requirement may interfere with an organization's needs. In some scenarios, different treatments between data subjects might be authorized. For example, it might be legal that car insurers charge a premium to *male* drivers to account for gender differences in accident rates [14].
- *Misinterpretation of requirements.* Misunderstanding the specifications of the requirements or intentionally deviating from them due to business needs will result in a non-alignment between the business processes and the requirements [129]. These deviations may result in unexpected conflicts between the requirements that may lead to unwanted vulnerabilities.

For example, for two persons who only differ in their gender and are otherwise identical, the activity may be required to produce the same output (*Individual Fair-*

ness). To fulfill this requirement, a business analyst may specify that the value of some gender attribute should not be accessed by the decision-making activity (*Undetectability*). However, this solution does not prevent *indirect discrimination* [30], which may arise from the availability of other data that are highly correlated with gender, such as a person's first name. In fact, to prevent discrimination, it might be necessary to access protected characteristics to identify correlated data or re-balance the output such that no group is discriminated against. In this scenario, as a consequence, the undetectability and the individual fairness requirements are conflicting with each other because we can not preserve both requirements.

Conflicts resulting from non-alignments between elicited data protection requirements and their specifications in the business processes, if not avoided early, can make the process of locating their root causes in later development stages difficult. To avoid these conflicts, the alignment between the requirements and the specifications in business processes should be ensured.

### 2.1.1 Problem statement and research questions

A few existing approaches are available to deal with different types of data protection requirements in the early stages of development. These approaches focus on the identification of security and data-minimization requirements in the elicitation phase without detecting conflicts between them [16, 32, 65, 95]. The output of these approaches is usually a set of *textual* requirements. Relying on textually specified data protection requirements to *manually* discover conflicts between them is a difficult and error-prone task for two main reasons.

First, conflicts between the data protection requirements depend on the context of how the technical and organizational components of the target system interact with each other. Specifically, conflicts not only result from trade-offs between requirements related to the same asset in the system (e.g., anonymous vs. accountable execution of a task), but also from those related to different assets. For example, a task may be required to be executed anonymously while writing data to a secure data storage where the identity of the writer must be known for accountability reasons. The detection of such conflicts requires an understanding of the underlying business processes and their included interactions between security and data-minimization requirements, which is a difficult task if the requirements are provided in a textual format and distributed through multiple documents.

Second, a single data protection concept may have multiple meanings based on *what* (which of the system assets) and from *whom* (i.e., adversary type) to protect. These variations make it hard to decide whether two specific requirements are con-



flicting. For example, providing fully anonymous execution of a specific task hinders the ability of the system to keep the task's executor accountable, leading to a conflict. In contrast, providing *partial* anonymity by means of using pseudonyms is not conflicting with accountability. Such details, if provided in a textual format, will make it difficult to keep track of what should be protected, from whom it should be protected, and how it should be protected.

Furthermore, to our knowledge, there is no approach supports fairness requirements in the early development stages. Detecting conflicts between fairness and other data protection requirements in the early stages of the system development is currently not possible. A textual syntax with precise semantics amenable to automatic analysis may be a solution to address the above challenges. However, we believe that expressing conflicts between data-protection requirements as graphical patterns during the design of the business process models of a system is a powerful way to communicate conflicts with the system's stakeholders. A graphical solution helps to manage the complexity of the models of large-scale real-world systems. Motivated by this, in this chapter, we investigate the following research question:

**RQ1.** *How to detect conflicts between security, data-minimization and fairness requirements by analyzing business process models?*

Our objective is to support business analysts with an approach that permits detecting conflicts between security, data-minimization and fairness requirements early as during the business process modeling time in order to reduce the effort needed for detecting conflicts in the later stages of system development.

### 2.1.2 Contribution

To address the research question RQ1, we present an extension of the Business Process Modeling Notation (BPMN 2.0, [1]) in two main directions: First, in order to address the challenge of possible non-alignments between the users' needs and their specifications in business process models, we propose an alignment checking technique based on reusable domain-specific patterns. Second, in order to address the challenge of detecting conflicts between security, data-minimization, and fairness requirements, we propose a conflicts detecting technique based on a catalog of domain-independent anti-patterns. Specifically, our contributions are:

1. a *semi-automated BPMN-based data-protection engineering framework* for supporting: (i) the enrichment of collaboration BPMN models with security, data-minimization, and fairness requirements, (ii) the encoding of such requirements as reusable, domain-specific patterns, (iii) the checking of alignment

between the encoded requirements and annotated BPMN models based on these patterns, and (iv) conflict detection between the requirements in annotated BPMN models based on a catalog of *domain-independent anti-patterns*,

2. a catalog of *domain-independent anti-patterns* to support the automatic conflict detection of among security, fairness, and data-minimization requirements, where each anti-pattern represents a conflict or potential conflict,
3. a *case study* featuring a healthcare management system, showing how our process can be used to uncover conflicts between security, data-minimization and fairness requirements, and
4. a *user evaluation*, in which we studied the usefulness of our conflict detection technique in an experiment with 30 participants.

The security annotations in our BPMN 2.0 extension were reused from an existing security-oriented BPMN extension called SecBPMN2 [129]. To express the domain-specific patterns and domain-independent anti-patterns, we extended a graphical query language for BPMN 2.0 models called SecBPMN2-Q [129]. While the security annotations were reused from the SecBPMN2, our proposed framework is the first to directly support modeling data-minimization and fairness requirements in BPMN models. It is also the first to support automatic conflict detection between specified security, data-minimization and fairness requirements in BPMN models.

This chapter is organized as follows. Section. 2.2 provides the necessary background. Section 2.3 provides a running example that will be used throughout the chapter to explain our contribution. Section 2.5 describes our proposed framework for detecting conflicts between security, data-minimization and fairness requirements. Section 2.4 introduces our BPMN extension. Section 2.6 presents our alignment verification approach. Section 2.7 presents the considered types of conflicts and our approach to detect them. Section 2.8 presents the tool support for our approach. Section 2.9 and 2.10 are devoted to the validation based on a case study and a user evaluation. Section 2.11 discusses the limitations of our approach and future work. Section 2.12 and 2.13 survey related work and conclude, respectively.

## 2.2 Background

In this section, we introduce the fundamental data-minimization and fairness concepts used in our work. We also provide an overview of discrimination and data-minimization from the legal aspects. In addition, we introduce a BPMN-oriented security engineering approach whose security concepts we reused.

**Table 2.1:** Definitions of the considered data-minimization concepts in our work.

Data-minimization concept	Definition as provided by Pfitzmann et al. [103]
<i>Anonymity</i>	is the inability of an adversary (i.e., outsider or insider) to sufficiently identify a subject within a set of subjects, called the anonymity set.
<i>Pseudonymity</i>	is a special case of anonymity where a pseudonym is used as an identifier for a data subject other than one of the data subject's personal identifiable information.
<i>Unlinkability</i>	is the inability of an adversary to sufficiently distinguish whether two Items Of Interests (IOIs, e.g., subjects, messages, actions, ...) within a system are related. Although it is not explicitly mentioned in [103], the definition of unlinkability implies that the two or more IOIs are of comparable types, otherwise it is infeasible to make the comparison ([32], p.8).
<i>Undetectability</i>	is the inability of an adversary to sufficiently distinguish whether an IOI exists or not. By the definition [103], undetectability of an IOI can only hold against outsider adversary (i.e., neither being the system nor one of the participants in processing the IOI).
<i>Unobservability</i>	is the undetectability of an IOI against all subjects uninvolved in it (i.e., outsider adversary) and the anonymity of the subject(s) involved in the IOI against other subject(s) involved in that IOI (i.e., insider adversaries).

### 2.2.1 Data-minimization concepts

Pfitzmann et al. [103] define five data minimization concepts that can be refined into privacy requirements for the target system [16, 32, 65, 95]. In Table 2.1, the five data-minimization concepts that are considered in our work are listed, each with its definition as provided in [103].

### 2.2.2 Overview of data-minimization: legal aspects and standards

In this section, we provide a brief overview of data-minimization from the current legal and standard frameworks. In what follows we focus on: (1) the concept of data-minimization under the European General Data Protection Regulation (GDPR); and (2) The data-minimization concepts of the Common Criteria for Information Technology Security Evaluation (referred to as Common Criteria).

**The European General Data Protection Regulation (GDPR).** The general goal of the GDPR, as specified in *Article 1(2)*, is to protect the fundamental rights and freedoms of natural persons and their right to the protection of personal data [2]. The GDPR consists of *articles* and *recitals*. The articles represent legal requirements organizations must follow to demonstrate compliance while the recitals provide additional information to supplement the articles.

Article 5 of the GDPR defines six principles that a data controller should demonstrate compliance with them when processing personal data. Among the defined principles, Article 5(c) states that personal data shall be "*adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed (data-minimization)*". Often to stay in compliance with this principle, system developers design the functionalities of the system in a way that allows processing personal data for the purposes they are collected only. However, this does not necessarily ensure that the collection of personal data is limited to what is necessary to achieve the system objectives.

Article 25 of the GDPR states that a data controller should adopt organizational and technical measures to meet data protection by design and data protection by default. The article suggests *pseudonymizing* personal data as one of the measures to achieve data-minimization by default. The GDPR does not mention other data-minimization concepts such as anonymity, unlinkability, and unobservability. However, Recital 28 of the GDPR states that "*the explicit introduction of 'pseudonymisation' in this Regulation is not intended to preclude any other measures of data protection*". This means that the GDPR introduces the pseudonymity as an example of one possible technical measure for achieving data-minimization.

**Common Criteria.** The Common Criteria for Information Technology Security Evaluation is an international standard (ISO/IEC 15408) for computer security certification. The Common Criteria provide support in building a secure system by offering classes of functional security and privacy components that a developer can select from. Specifically, the classes of the security and privacy components provides a standard framework that establishes the general concepts and principles of IT security evaluation. Together with the security refinement guidelines, the classes support a data protection expert in eliciting data protection requirements. The privacy function component, which is defined in the second part<sup>2</sup> of the Common Criteria, is restricted to the following data-minimization concepts: *Anonymity, Pseudonymity, unlinkability, and Unobservability*.

In comparison with the taxonomy of Pfitzmann et al. [103] whose definitions for the data-minimization concepts are reused in our work, we find that the Common Criteria are restricted in several directions: First, the *Undetectability* is not considered by the Common Criteria as one of the privacy requirements. Second, the outsider perspective in the Common Criteria is not considered. More specifically, all the considered data-minimization concepts by the Common Criteria are defined to

---

<sup>2</sup>The Common Criteria consists of three parts. The first part introduces the general concepts used throughout the Common Criteria. The second part presents the security and privacy functional components. The third part provides measures for evaluating the security assurance. The second part of the Common Criteria is available online at [https://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R5\\_marked\\_changes.pdf](https://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R5_marked_changes.pdf) (accessed: 16/12/2019).

protect a data subject from the software in question or from a set of users for that software. However, both the software and its users reflect the insider adversary perspective only. For example, the *anonymity* in the Common Criteria requires that "other users or subjects are unable to determine the identity of a user bound to a subject or operation." ([60], p.119). However, one may be interested in protecting the users' identities of a system from an outsider adversary who is not part of the system. Third, some of the data-minimization concepts in the Common Criteria are applied to a specific item of interest. For example, the *unlikability* in the Common Criteria requires that "users and/or subjects are unable to determine whether the same user caused certain specific operations"([60], p.123). Differently, the definition of the unlikability by Pfitzmann et al. [103] is not restricted to the users' operations but it considers other possible item of interests such as data objects and messages.

### 2.2.3 Fairness concepts

Although there is no single definition of what fairness means [149], two main types of fairness are distinguished namely, individual fairness and group fairness. These concepts are provided in Table 2.2 as defined by Galhotra et al. [49].

**Table 2.2:** Definitions of the considered fairness concepts in our work.

Fairness concept	Definition as provided by Galhotra et al. [49]
<i>Individual fairness</i>	An activity with a decision-making purpose preserves the individual fairness if it produces the same decision for every two individuals whose data, that are given as input to the decision-making activity, are identical except for the protected characteristics.
group fairness	An activity with a decision-making purpose preserves the group fairness if it produces equally distributed outputs for each protected group.

Consider, for example, a decision-making activity in a bank to decide if loan applicants should be given loans. We say that the activity preserves individual fairness with respect to *gender* if the activity produces the same result (i.e., loan vs not loan) for every two identical loan applicants who differ only in their gender. We say the activity preserves group fairness with respect to gender if the outcome fractions of males and females who will get a loan are equal.

It is worth mentioning that protected characteristics are not limited to those listed as protected by the laws and regulations such as the GDPR [2] and the UK Equality Act 2010. Depending on the organizations' policies and the context other data may be considered as protected. For example, the *type of technology* that people use to access the web is not considered as protected characteristic in *Article 9* of the GDPR [2], but it can act as protected in the policies of a specific organization in the case of, for example, advertising decisions.

However, avoiding using protected characteristics in a decision-making activity does not necessarily ensure fairness. Other data may act as *proxies* for protected characteristics due to correlations between them, and as a consequence, may lead to *indirect discrimination* [29]. For example, in 2016, it was found that a decision-making software by Amazon.com, Inc., excluded minority neighborhoods with African American community in the United States from being able to participate in a delivery-free service<sup>3</sup>, although the software did not explicitly use *ethnicity* for making the decisions.

The decision on whether a piece of data is correlated with a protected characteristic or not depends on a correlation metric and a threshold value that can be specified by domain experts. On the other hand, discrimination on the ground of protected characteristics might be allowed if there is a legitimate purpose that can justify it. For example, it might be legal to discriminate on *age* for life insurance decisions. Data whose effect on the outputs of a decision-making activity can be justified are called *explanatory data* [145].

#### 2.2.4 Overview of discrimination from the legal aspects

In this section, we provide a brief overview of discrimination from the current legal frameworks. Two important sources of law when considering fairness are equality laws and data protection laws. The goal of any equality law of a country is to provide guidance and notice to individuals, organizations, corporations, and agencies regarding the parameters of illegal discrimination, including the characteristics that are legally-protected, such as gender and age. Second, the aim of any data protection law of a country or a region is to regulate the processing of personal data such as identifying when it is allowed to process a specific data item of an individual.

In what follows we focus on: (1) the concept of non-discrimination under the European General Data Protection Regulation (GDPR); (2) the German General Act on Equal Treatment 2006; (2) the UK Equality Act 2010.

**The General Data Protection Regulation (GDPR).** As mentioned earlier in Section 2.2.2, the GDPR consists of *articles* and *recitals*. The articles represent legal requirements organizations must follow to demonstrate compliance while the recitals provide additional information to supplement the articles.

The risk of discrimination is explicitly mentioned in *Recital 75* of the GDPR, which states that "*the risk to the rights and freedoms of natural persons, [...], may result from personal data processing which could lead to physical, material or non-material damage,*

---

<sup>3</sup>Detailed description for the discriminatory behavior of Amazon's software is available at <https://www.bloomberg.com/graphics/2016-amazon-same-day/> (accessed: 05/12/2019).

in particular: where the processing may give rise to *discrimination*, [.....]". This recital demonstrates that preventing discrimination is one of the fundamental rights and freedoms of natural persons.

Fairness in the context of preventing discrimination is mentioned indirectly in the GDPR in the sense of non-discrimination. For instance, *Recital 71* states a requirement to "implement technical and organizational measures appropriate to [...], and prevent, inter alia, discriminatory effects on natural persons on the basis of racial or ethnic origin, political opinion, [...]". This recital adds more information to *Article 22* of the GDPR, which prevents decision making based on special categories of personal data. *Article 9* of the GDPR refers to the following list of personal data as special categories: racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and genetic data, biometric data, data concerning health or data concerning a natural person's sex life or sexual orientation. *Article 9(2)* of the GDPR adds exceptions for the prohibition of processing special categories of data, including reasons of public interest in the area of public health.

The GDPR does not provide a precise definition of discrimination. However, the term "discriminatory effects" in *Recital 71* shows that the GDPR takes into consideration the so-called indirect discrimination. However, the GDPR does not provide or suggest mechanisms for preventing indirect discrimination. It also does not define fairness measures or requirements such as individual fairness and group fairness.

**The German General Act on Equal Treatment (General Act).** This Act is a German federal law that aims to prevent or to stop discrimination on the ground of protected characteristics<sup>4</sup>. The General Act defines the following list of personal data as protected characteristics that should not be subject to discrimination: race or ethnic origin, gender, religion or belief, disability, age or sexual orientation. The General Act does not apply in all social and legal areas. Rather, it only prohibits discrimination in certain situations if it is based on the ground of specific protected characteristics. Examples of situations where the General Act applies include: employment, health services, and education.

The General Act provides a distinction between five forms of discrimination. These five forms are defined with respect to *Section 1* of the General Act, which defines the list of the protected characteristics:

- (i) *Direct discrimination*: less favourable treatment of one person than another in a comparable situation on any of the grounds referred to under *Section 1* of the General Act.

---

<sup>4</sup>The German General Act on Equal Treatment is available online at [http://www.gesetze-im-internet.de/englisch\\_agg/](http://www.gesetze-im-internet.de/englisch_agg/) (access: 16/12/2019).

- (ii) *Indirect discrimination*: discrimination through an apparently neutral provision, criterion or practice would put persons at a disadvantage compared with other persons on any of the grounds referred to under *Section 1* of the General Act, unless that provision, criterion or practice is objectively justified by a legitimate aim.
- (iii) *Harassment*: discrimination when unwanted conduct in connection with any of the grounds referred to under *Section 1* of the General Act takes place with the purpose or effect of violating the dignity of the person concerned and of creating an intimidating, degrading, humiliating or offensive environment.
- (iv) *Sexual harassment*: is concerned with any sexually determined behavior through which a person feels uncomfortable and hurt in his/her dignity.
- (v) *Instruction to discriminate*: an instruction to discriminate against a person on any of the grounds referred to under *Section 1* of the General Act shall be deemed as discrimination. This situation happens when someone instructs someone else, such as an employee, to discriminate against another person.

The General Act does not specify what the term “comparable” in the definition of direct discrimination exactly means. For example, one can interpret the term comparable as two persons who are identical in their data and only differ in their protected characteristics. If this is the case, then this definition of direct discrimination reflects the violation of the so-called individual fairness, which states that two persons who are identical in their data and only differ in their protected characteristics or their proxies should receive the same treatment.

In the General Act, there might be some situations where indirect discrimination is allowed if it can be justified. However, this case does not apply to direct discrimination as the definition of direct discrimination in the General Act does not provide any exception for directly processing protected characteristics.

**The UK Equality Act 2010.** The goal of this Act is to provide a single framework for numerous prior discrimination Acts in the United Kingdom such as the Equal Pay Act 1970, the Sex Discrimination Act 1975, and the Race Relations Act 1976.

The UK Equality Act<sup>5</sup> 2010 provides nine categories of protected characteristics: *Age, disability, gender reassignment, marriage and civil partnership, pregnancy and maternity, race, religion or belief, sex, sexual orientation*. In comparison to the special categories of data that are provided by the European General Data Protection (GDPR), the list of the protected characteristics in the UK Equality Act 2010 seems to be

---

<sup>5</sup>The UK Equality Act 2010 is available online at [http://www.legislation.gov.uk/ukpga/2010/15/pdfs/ukpga\\_20100015\\_en.pdf](http://www.legislation.gov.uk/ukpga/2010/15/pdfs/ukpga_20100015_en.pdf) (access: 16/12/2019).



restricted. For example, according to the GDPR, making decisions on the basis of genetic information is not allowed. However, the genetic information is not defined as a protected characteristic in the UK Equality Act 2010.

Similar to the GDPR and the German General Act, the UK Equality Act 2010 does not differentiate between specific types of fairness requirements such as individual fairness and group fairness. However, different from the GDPR, the UK Equality Act 2010 states explicitly that discrimination can happen directly or indirectly. These two forms of discrimination are defined as follows:

- *Direct discrimination* is defined in *Section 13(1)* of the UK Equality Act 2010 as: if with respect to a specific protected characteristic, A (i.e., a person) treats B (i.e., another person) less favourably than A treats or would treat others.
- *Indirect discrimination* is defined in *Section 19(1)* of the UK Equality Act 2010 as: A (i.e., a person) applies to B (i.e., another person) a provision, criterion or practice which is discriminatory in relation to a relevant protected characteristic of B's.

Similar to the GDPR the UK Equality Act 2010 adds exceptions for processing protected characteristics. For example, *Sections 158-159* of the Equality Act 2010 permit positive actions under certain circumstances. Positive action is a range of measures under the Equality Act 2010 which can be lawfully taken to encourage and train people from under-represented groups. For example, if an academic organization has a low rate of applications from women for an academic career in certain subjects, then positive actions may be undertaken by the organization to encourage women and improve this situation.

### 2.2.5 BPMN-based data-protection engineering

Modeling data protection requirements during the design phase of the business processes models is a promising research direction in the field of data protection engineering [82]. The key idea is to extend graphical business process modeling languages such as BPMN [1] to support the modeling and analysis of data protection requirements.

In the state of the art, there is a lack of research work on supporting data-minimization and fairness requirements in business process models. For data-minimization requirements, we are only aware of the work proposed in [120], which considers one data-minimization requirement, namely anonymity. However, fairness and further fundamental data-minimization requirements such as

unlinkability were not addressed yet. Readers interested in a comprehensive overview of the concepts that have been considered by previous BPMN data protection-oriented extensions are referred to Maines et al.'s survey in [82].











To capture conflicts between security, data-minimization, and fairness requirements, a unified framework for modeling these types of requirements is needed. Compared to other BPMN data protection-oriented extensions, we found that SecBPMN2 [129] offer the following advantages:

- First, in contrast to the research work in [13, 17, 75, 89, 120, 124, 151] which support only a restricted set of security aspects, SecBPMN2 enriches BPMN 2.0 modeling language with 10 security concepts. Reusing the SecBPMN2 security concepts allows us to study interactions between a comprehensive set of security, data-minimization, and fairness requirements, enabling a powerful approach to conflict detection. In Section 2.2.6, we list the 10 security concepts of SecBPMN2 together with their annotations and definitions.
- Second, while other works [96, 150] use textual stereotypes to enrich business process models with security requirements (e.g., «confidentiality»), SecBPMN2 represents security requirements using graphical annotations [129]. Compared to textual annotations, graphical ones can reduce the cognitive complexity for understanding the resulting business process models [91], and as a result, contribute to usability.
- Third, SecBPMN2 provides a security query language for specifying queries that can be matched against a given SecBPMN2 model, called SecBPMN2-Q [129]. We reuse and extend this query language in our approach for two dependent purposes: First, formulating the security, data-minimization, and fairness requirements of the system in questions as patterns that can be used later for reasoning about the alignment between these requirements and the corresponding SecBPMN2 model of system-to-be. Second, specifying domain-independent conflicts between security, data-minimization, and fairness requirements as anti-patterns that can be used later for uncovering conflicts in SecBPMN2 models. Details description on the query engine of SecBPMN2-Q is provided in Section 2.2.7.

### 2.2.6 SecBPMN2 security concepts

The security concepts of SecBPMN2 are designed with respect to security concepts of the Reference Model on Information Assurance and Security (RMIAS) [25], which was assembled through an analysis of security aspects proposed by known reference models such as the Confidentiality-Integrity-Availability (CIA) triad.

In what follows, we list the 10 security concepts that are supported by SecBPMN2, each with its corresponding annotation and definition as provided in [129]. Worth noting that the *binding of duties*, *separation of duties* and *non-delegation* SecBPMN2 security concepts are not considered as part of the RMIAS. These concepts were added to the SecBPMN2 by the authors of [129] in order to support a comprehensive set of security concepts.

-  *Accountability* specifies that the system should hold the executors of the activities responsible for their actions.
-  *Authenticity* imposes that the identity of a given activity's executor must be verified, or that it should be possible to prove a given data object as genuine, respectively.
-  *Auditability* indicates that it should be possible to track of all actions performed by an executor or accessor of an activity, data object, or message flow.
-  *Non-delegation* specifies that an activity shall be executed only by assigned users.
-  *Non-repudiation* imposes that an executor or accessor of an activity, data object, or message flow should not be able to deny his/her actions.
-  *Binding of duties* requires that the same person should be responsible for the completion of two related tasks.
-  *Separation of duties* requires that different persons should be responsible for the completion of two related tasks.
-  *Confidentiality* indicates that only authorized users are allowed to read data from a given activity, message flow, or data object.
-  *Integrity* indicates that only authorized users are allowed to modify data from a given activity, message flow, or data object.
-  *Availability* indicates that it should be possible to ensure that an activity, data object, or message flow is available and operational when they are required by authorized users.

In what follows, the *data protection* concept, otherwise it is mentioned, is used to refer to all kind of security, data-minimization, and fairness concepts that are considered in this chapter.

### 2.2.7 SecBPMN2-Q

SecBPMN2-Q is a security query language for specifying queries that can be matched against a given SecBPMN2 model [129]. For query evaluation, SecBPMN2-Q uses an artificial intelligence system based on disjunctive logic programming, DLV [80]. In particular, it is based on planner functionality called DLV- $\mathcal{K}$  [40]. The planner uses a knowledge base and defines possible plans to be executed in order to achieve given objectives. In case of SecBPMN2, the knowledge base consists of the business process(es) analyzed, while the objectives are the (anti-)patterns to verify. SecBPMN2 instructs DLV- $\mathcal{K}$  to search possible executions of the process(es) that satisfy the patterns; if it finds at least one, it means that the (anti-)pattern is matched. This solution allows to overcome performance issues due to the possible very complex design of business processes.

## 2.3 Running Example

In this section, we provide the running example business process model that will be used throughout this chapter. Figure 2.2 represents a business process in the context of healthcare management. A patient uses a telemedicine device to receive an over-distance healthcare service and evaluates the service through an online evaluation portal. A patient who wishes to donate in one of his vital organs can fill a donation form and send it through an online donation portal.

Executors of a business process are represented by *pools*<sup>6</sup> and *lanes* such as “Telemedicine Device” and “System Portal”, respectively. Communication between pools is represented by *message flows*; the content of such communications is represented using *message*. For instance, “Telemedicine Device” sends the message “measures” to “System Portal”. Atomic actions are represented with *tasks*, for example, “measure vital signs”. A task is positioned inside a pool with the meaning that the actor represented by the pool will execute the task. For example, “measure vital signs” task will be executed by “Telemedicine Device”.

*Data Objects* represent data used in a business process model. For example “EHR” represents an electronic healthcare record, i.e., data about patients. Data objects are connected to tasks using a *data association*: a directional relation used to model the flow of data between a task and a data object. If a *data association* starts from the data objects and targets the task, then the task reads the data object. If the *data association* targets the data objects then the task writes the data objects. If the data objects is connected to the same task with two *data associations* with different orientations,

<sup>6</sup>In this chapter, we use *italic* for concepts and “sans serif font” for examples.

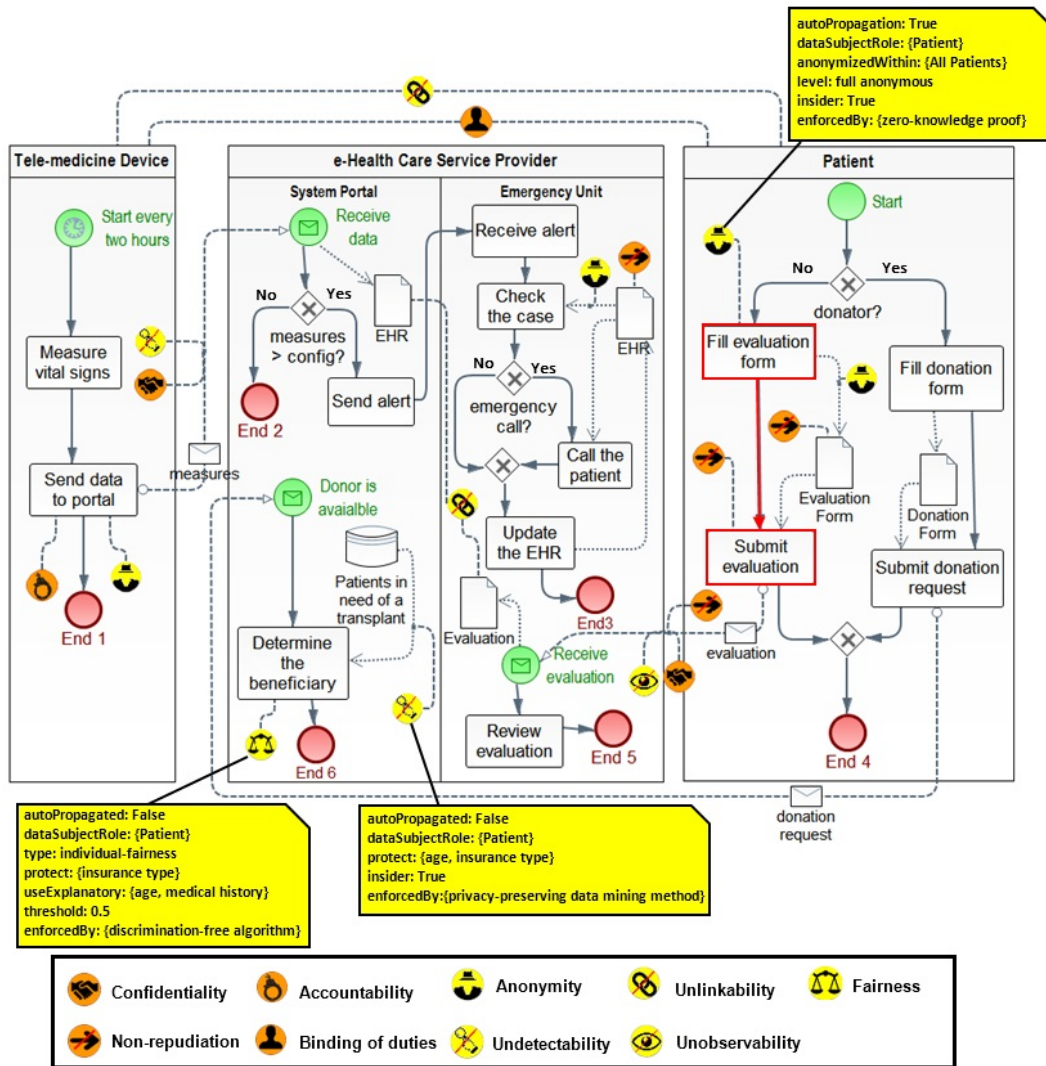


Figure 2.2: Running example: Specifying security, data-minimization, and fairness requirements in a healthcare business process.

then the task modifies the data object. For instance, “Check the case” task reads the “EHR” data object, while “Update the EHR” writes the same data object.

Events represent external actions/states that interact with a business process. Events in SecBPMN2 are represented with circles. There are two types of events (relevant for the purpose of our running example): (i) *start events*, which represent the initial point of a business process; (ii) *end events*, which represent the terminal point of a business process. For example, the start event of the process executed in the pool “Tele-medicine Device” specifies that the process will start every two hours,

while the two start events in the swimlane “System portal” specifies that the processes will start when a message is received.

*Gateways* specify deviations of the execution sequence, they split the processes in two or more branches and allow an executor of the process to execute one branch based on a condition specified. For example, the gateway “measures>=config?” in swimlane “System portal”, specifies that if “measures” is higher than the parameter “config”, then the right branch is executed, otherwise, the left branch is executed.

Security concepts are represented with orange annotations. As explained in Section 2.2.6, SecBPMN2 supports 10 security annotations. However, in our running example, we use only 4 of them, described as follows:

- *Confidentiality* (🔒) is associated to message flows, meaning that the content of the message is to be preserved and not to be accessed by unauthorized users, respectively. Depending on the security mechanism used to enforce such security annotations, the communication channel that is used to exchange the messages, represented in the business process, will encrypt messages or use technologies such as Virtual Private Networks (VPN).
- *Accountability* (👁️) can be associated with tasks and expresses the need for monitoring the execution of the tasks. In Figure 2.2 is associated to “Send data to portal” meaning that the task’s executor must be monitored. Implementing such monitors will require a monitoring component that intercepts the calls to the tasks or store the logs of the task’s execution, for later inspections.
- *Bind of Duty* (👤) requires the same actor to be responsible for the completion of a set of related activities. This annotation must be linked to two pools or two lanes and can be enforced using an access control security mechanism, which forces a set of activities to be executed by the same user.
- *Non-repudiation* (➡️) indicates that the execution (or non-execution) of a BPMN element must be provable.

Some of the security annotations of SecBPMN2 can be defined in one or more variants, depending on which element the annotation is connected. For example, as shown in Figure 2.3, *Non-repudiation* (➡️) has three variants.

Our new fairness and data-minimization concepts, discussed in this chapter, are represented with yellow annotations in Figure 2.2, expressing that these concepts are more directly related to data-minimization and fairness than security. The data-minimization and fairness annotations are an original contribution of our work and we will describe them through the next sections of this chapter.

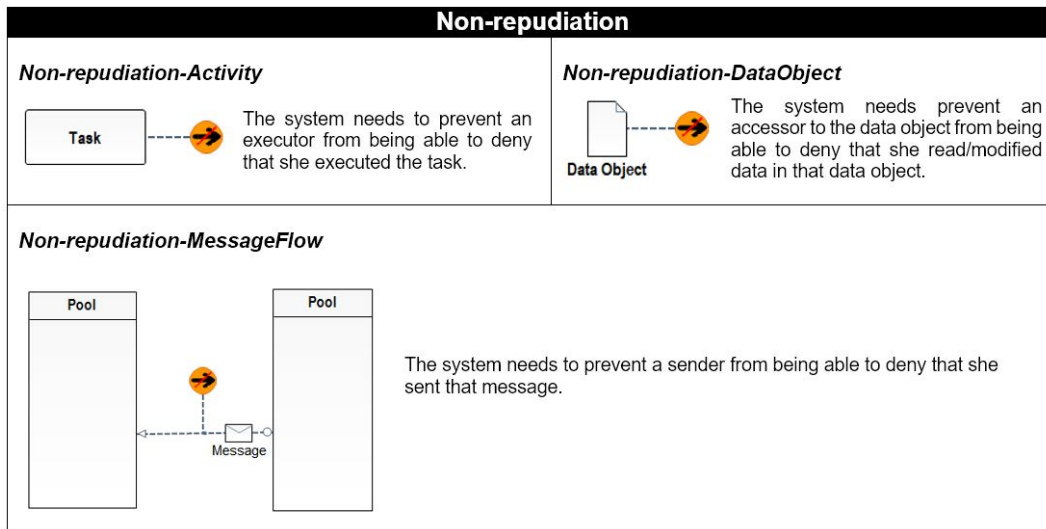


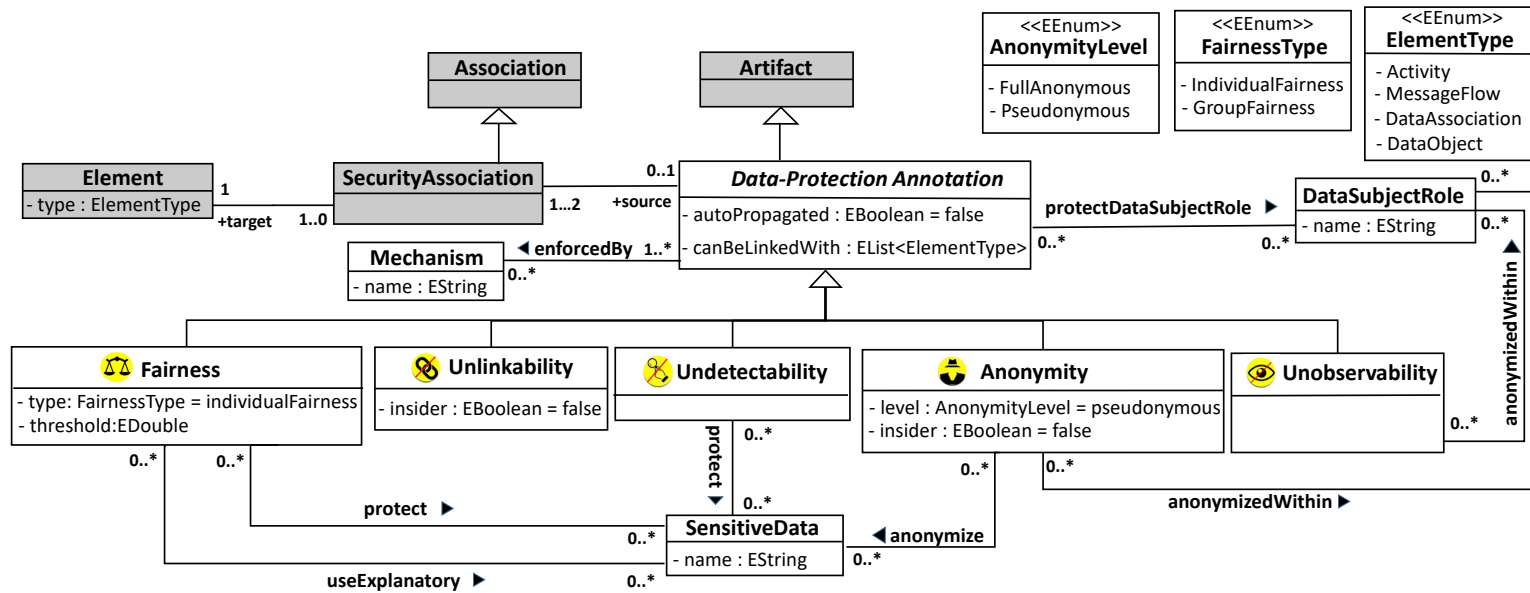
Figure 2.3: All variants of SecBPMN's *non-repudiation* annotation.

## 2.4 Modeling Security-, Data-Minimization- and Fairness-Requirements

In this section, we propose a BPMN extension for specifying security, data-minimization, and fairness requirements. Our support for data-minimization and fairness requirements for the design of business processes is an original contribution, while the security-specific elements are reused from SecBPMN2, an existing security-oriented BPMN 2.0 extension.

### 2.4.1 Data-minimization and fairness annotations

The meta-model of our BPMN extension with data-minimization and fairness concepts is shown in Figure 2.4. Gray parts in the meta-model are part of SecBPMN2 elements while the white parts are new elements. As shown in Figure 2.4, in order to allow users to enrich business process models with data-minimization and fairness requirements, we extended the *artifact* class from BPMN with five concepts, namely *anonymity* (👤), *undetectability* (✂️), *unlinkability* (🔗), and *unobservability* (👁️), and *fairness* (⚖️). The first four concepts are data-minimization-specific.



\*\* Attributes show their default values.

Figure 2.4: Meta-model of our BPMN extension.



Since an additional concept described by Pfitzman et al., *pseudonymity*, is a special case of anonymity, we use one annotation for both concepts. Similarly, since both *individual-* and *group-fairness* concepts are special types of fairness, as described in Section 2.2.3, we use the same annotation for both of them. However, to allow capturing the variations between these concepts, a *type* attribute in the fairness annotation allows specifying the fairness type (i.e., group- or individual fairness), while a *level* attribute in the anonymity annotation allows specifying the required level of anonymity (i.e., fully anonymous vs pseudonymous). Using one annotation to represent related concepts is recommended to reduce graphical complexity [82].

Also for the purpose of reducing graphical complexity, we introduced a specific annotation for *unobservability*, although it, by definition, can be achieved by preserving both anonymity against insider adversaries and undetectability against outsider adversaries. We designed the graphical syntax of our annotations by following Moody's guidelines for increasing the usability of modeling languages [91]. The data-minimization and fairness annotations share two common visual aspects with the security annotations of the SecBPMN2: they all have a solid texture, and a circular shape; they differ in their fill color, using yellow instead of orange. We believe that having different colors for security on one hand and fairness and data-minimization, on the other hand, contributes to usability, as one can easily distinguish between fairness and data-minimization annotations from security ones. Given the graphical syntax of our annotations is described, in what follows we describe the meta-model of our BPMN extension.

A special type of association called *SecurityAssociation* is used to link the proposed annotations with elements in the BPMN model. Each of the proposed annotations is constrained to be linked with one or a list of BPMN elements of the following types: *activity*, *message flow*, *data association* or *data object*. That is to avoid overlapping semantics of different annotations. For example, two messages cannot be linked to each other as related (i.e., unlinkability) if they are sent anonymously (i.e., anonymity). Therefore, having both unlinkability and anonymity annotations for message flows would be redundant. The linkage constraints are enforced by the SecBPMN2 editor. Details on the linkage constraints and the rationale beyond them are provided in 2.4.2.

As shown in Figure 2.4, our proposed data-protection annotations have a reference to the *Mechanism* class called *enforcedBy*. This reference can be used as an attribute to allow business analysts to specify the mechanism(s) needed to enforce a data-minimization and fairness requirement in later phases of development. Other details for specifying data-minimization and fairness requirements are captured using other attributes. For example, in the case of a fairness annotation, specific references can be used to describe in particular *protected characteristics* and *explanatory data*, which are described in Section 2.2.3.

To reduce specification overhead, the proposed annotations have an attribute *autoPropagated* which supports the propagation of the requirement to other elements in the model. Four cases of propagation are possible, depending on the type of the element the annotation is linked with:

1. for an activity, the requirement is propagated to all following tasks in the same lane,
2. for a message flow, the requirement is propagated to all message flows that goes from the source pool of the considered message flow to its target pool,
3. for a data input association, the requirement is propagated to all data input associations that read data from that data object in the same lane, and
4. for a data output association, the requirement is propagated to all data output associations that write data to that data object in the same lane.

In the rest of this section, the proposed data-minimization and fairness annotations are defined. Each of them is defined in terms of one or more variants since the semantic of annotations change based on which element the annotation is connected. The definitions are summarized in Figure 2.5. Next, we describe the definitions and the attributes that can be used to shape the exact semantics of the annotations.

**Anonymity**, as shown in Figure 2.5, comes in four variants, based on the BPMN 2.0 element it is linked to:

- (i) *Anonymity-Activity* specifies that the executor of the task should be anonymous within a set of executors for the task with respect to a given adversary perspective.
- (ii) *Anonymity-MessageFlow* specifies that the sender of the message should be anonymous within a set of senders for the message with respect to a given adversary perspective.
- (iii) *Anonymity-DataInputAssociation* specifies that the task should only read an anonymized variant of a predefined list of sensitive data when retrieved from the data object.
- (iv) *Anonymity-DataOutputAssociation* specifies that the task should not write a predefined list of sensitive data to the data object.

Based on our meta-model in Figure 2.4, the following attributes can be used to shape the exact semantic of the anonymity annotation: the reference attribute *pro-*

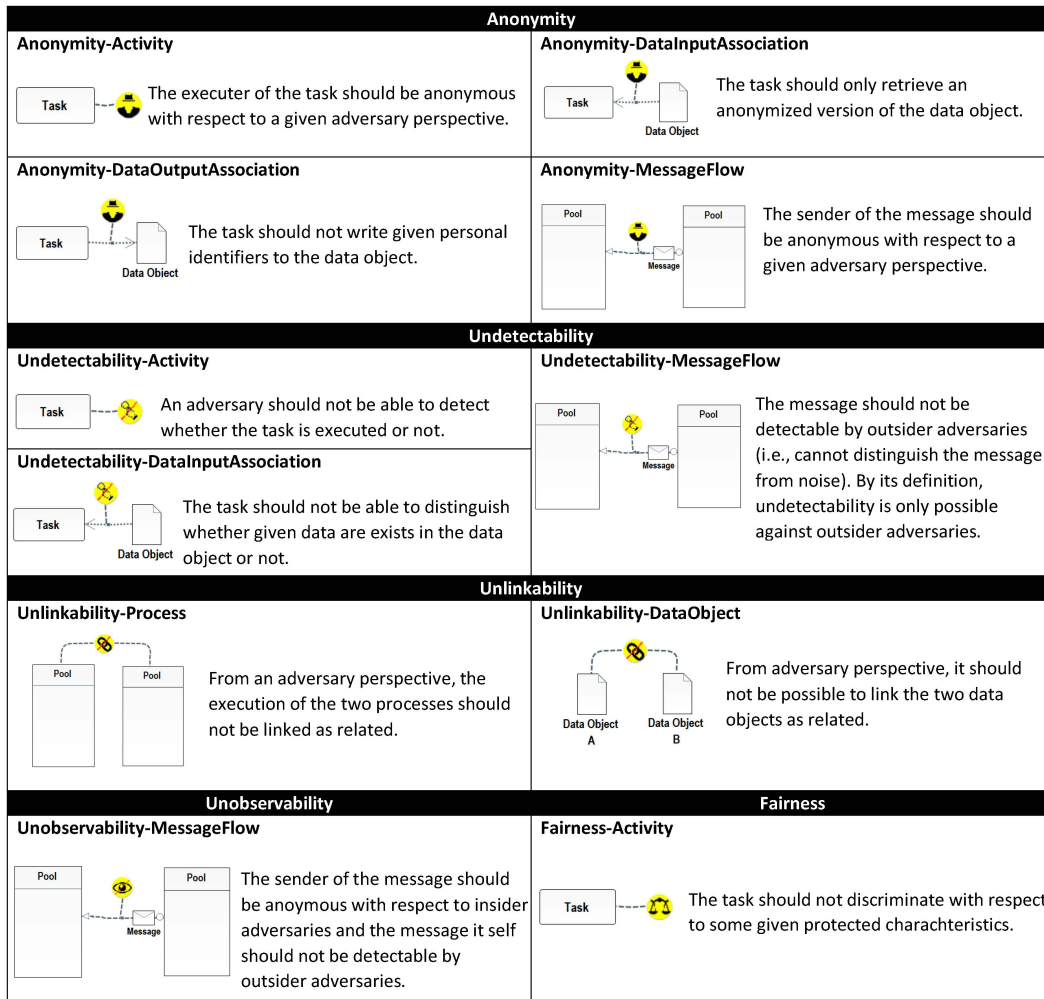


Figure 2.5: All variants of our data-minimization and fairness annotations.

*tectDataSubjectRole* specifies the roles who can execute an activity or send a message in case anonymity linked to an activity or a message flow, respectively. While in case anonymity linked to a data association, it specifies the roles of the data subjects whom the data are about. The reference attribute *anonymizedWithin* can be used to specify the anonymity set that an involved data subject should be anonymous within. In case anonymity is linked to a data association, the reference attribute *anonymize* can be used to specify the list of sensitive data that should be anonymized when writing/reading them to/from a data object.

The attribute *level* specifies the required anonymity level (i.e., fully anonymous or pseudonymous). In some scenarios, the system requires the executor of an activity

should be accountable, and thus, pseudonyms should be used to de-identify the executor of the activity. The attribute *insider* specifies against who to protect. The considered adversary type is either just outsider (*false*) or both outsider and insider (*true*). We define the outsider adversary as any entity being part of the surrounding of the system considered. The insider is any entity being part of the system considered, including the system itself.

The example model Figure 2.2 shows four anonymity annotations associated to different BPMN elements. Consider, for example, the one associated with the “Fill evaluation form” activity, which is at the right hand of the model. This annotation specifies that a patient shall be able to execute the “Fill evaluation form” task anonymously within the set of “all patients” without being identifiable by either outsider or insider adversaries. The annotation suggests using the “zero-knowledge proof” [90] mechanism to enforce anonymous authentication. Since the requirement is propagated, the same requirement applies to the “Submit evaluation” task.

**Unlinkability**, as shown in Figure 2.5, comes in two variants, based on the BPMN 2.0 element it is linked to:

- (i) *Unlinkability-Process* can be linked with two pools/lanes to specify that an adversary of the given type shall not be able to identify two executed processes as related. In other words, if linked to two pools, this annotation imposes that a subject may make use of multiple services without allowing others to link these uses together as related [103].
- (ii) *Unlinkability-DataObject* can be linked with two data objects to specify that, from the given adversary perspective, it should not be possible to identify the two data objects as related. More specifically, if linked to two data objects, this annotation specifies that the two data objects should do not share data that may allow others to profile a data subject by linking information from different sources about the data subject as related to each other.

Based on our meta-model in Figure 2.4, the reference attribute *protectDataSubjectRole* can be used to specify the data subjects who should be protected from the linkability threats in terms of using their roles. Since unlinkability can only be applied to two specific processes or data objects, it cannot be propagated to other elements. The attacker type is specified using the *insider* attribute, in the same way as in the *anonymity* case. The example model in Figure 2.2 includes two unlinkability annotations. Consider, the unlinkability annotation associated with the two data objects namely, “EHR” and “Evaluation”. This annotation specifies that both outsider and insider adversaries must not be able to link an “EHR” and an “Evaluation”

data objects as related. The annotation suggests using the “Role-based Access Control Mechanism (RBAC)” [46] and the “k-anonymity” [143] mechanisms to enforce this requirement. We did not show these specifications in the model to keep it readable.

**Undetectability**, as shown in Figure 2.5, has three variants, based on the BPMN 2.0 element it is linked to:

- (i) *Undetectability-Activity* specifies that an adversary should not be able to detect whether an activity is executed or not.
- (ii) *Undetectability-DataInputAssociation* specifies that the task should not be able to distinguish whether a predefined list of sensitive data exists in a data object or not.
- (iii) *Undetectability-MessageFlow* specifies that an adversary cannot sufficiently distinguish true messages from false ones (e.g., random noise).

Based on our meta-model in Figure 2.4, the following attributes can be used to shape the exact semantic of the undetectability annotation: The reference attribute *protectDataSubjectRole* can be used to specify the roles of the involved data subjects whom their sensitive data should be protected from the detectability threats. In case an undetectability annotation is linked to a data input association, an additional reference attribute to the *SensitiveData* class called *protect* is applied. This attribute allows us to specify the predefined list of sensitive data and that a conclusion about their existence in the data object in question should not be possible.

By definition ([103], p.16), undetectability is only possible against outsider adversaries. This is true only when the target of undetectability is to protect a sensitive message or the execution of a sensitive activity from the detectability threat, as the work in [103] defines the undetectability at the network level. However, in an information system, one may be interested in preventing insiders from being able to draw a conclusion about whether, at least, some sensitive data are available in a specific data object or not. Thereby, in our work, the considered adversary type if an undetectability annotation is linked to an activity or message flow is the outsider adversary. While if an undetectability annotation is linked to a data input association, then insider is the considered adversary.

The example model Figure 2.2 shows two undetectability annotations. Consider, for example, the undetectability annotation linked with the data-input association between the “Determine the beneficiary” task and the “patients need of a transplant” data store in the “System Portal” pool. The annotation specifies that the “Determine the beneficiary” task, as insider activity in the business process, should not be able

to draw a conclusion about whether the “age” and the “insurance type” of the patient are stored as part of their data in the “patients need of a transplant” data store. The annotation suggests to use a “privacy-preserving data-mining method” (e.g., [88]) as a mechanism to enforce this requirement.

**Unobservability**, as shown in Figure 2.5, can only be applied to message flows, leading to one variant called *Unobservability*: the sender of the message should be anonymous with respect to insider adversaries and the message itself should not be detectable by outsider adversaries.

Based on the meta-model in Figure 2.4, the following reference attributes can be used to specify the exact semantic of the unobservability annotation: the attribute *protectDataSubjectRole* specifies all roles that can act as the sender for the message. The attribute *anonymizedWithin* can be used to specify the anonymity set that the involved data subjects should be anonymous within.

The example model includes an unobservability annotation linked with the message flow between the “Submit evaluation” task and the “Receive evaluation” event. This annotation specifies that an outsider adversaries should not be able detect true messages being sent over the message flow from false ones, and the patient who sent messages over the message flow must be anonymous to the insider adversary within all patients. The annotation suggests to use the “DC-networks” [24] and the “Dummy traffic” [115] as mechanisms to enforce this unobservability requirement.

**Fairness**, as shown in Figure 2.5, can only be applied to activities, leading to precisely one variant called *Fairness*: it specifies that the activity should not discriminate between the data subjects based on a specific list of their sensitive data, called protected characteristics.

Based on the meta-model in Figure 2.4, the exact semantic of the fairness annotation can be shaped by the *protectDataSubjectRole*, *Protect* and *useExplanatory* reference attributes and the *type* attribute, as it follow. The reference attribute *protectDataSubjectRole* specifies the set of the data subjects that may be affected by the output of the fairness-annotated activity in terms of their roles in the system.

Depending on the business needs, in one context, discrimination between data subjects based on their *SensitiveData* (i.e., in this case, sensitive data represent protected characteristics) may not be allowed (e.g., age- and gender-based discrimination for hiring decisions), while in another context discrimination between data subjects based on the same *SensitiveData* or a subset of them might be allowed (e.g., age-based discrimination for life insurance). Such data can be specified in the fairness-annotated activity by using the *protect* and *useExplanatory* attributes. The former permits specifying the list of protected characteristics that should not influence the

output of the fairness-annotated activity. The latter specifies *explanatory data*, i.e., sensitive data whose usage in the fairness-annotated activity can be justified in the given context to counter discrimination.

The attribute *type* specifies the required fairness type (i.e., individual- or group-fairness). In some scenarios, the output of the fairness-annotated activity should be equally distributed between the fractions of the affected data subjects with respect to their protected characteristics (group fairness). In other scenarios, the fairness-annotated activity must produce the same output for every two data subjects who differ only in their protected characteristics (individual fairness).

The attribute *threshold* is used for proxy discrimination. More specifically, in addition to what is defined as protected characteristics and explanatory data, a task may process other data which are not protected by law, but can act as proxies for protected characteristics if there is high correlations between them. For example, although the *address* data item is not defined as a protected characteristic by the recent regulations, in some countries such as the United States, the address can act as a proxy for the *ethnicity* due to a high correlation between them.

Deciding when two data items are correlated so strongly that the one acts as a proxy for the other requires a correlation metric and a numeric threshold, which have to be determined by domain experts. Varying metrics have been used in previous work to measure correlations [145]. For example, the term “Pr(insurance type = Private | job = Engineer)” represents the probability that a person is privately-insured given that the person is working as an engineer. In contrast, the term “Entropy(insurance type = Private | job = Engineer)” measures the *uncertainty* that a person is privately-insured given that the person is working as an engineer. It is noteworthy that variant correlation metrics may lead to different results, and a domain expert has to choose the most appropriate metric carefully.

The example model in Figure 2.2, specifically at the bottom of the pool “System Portal”, shows one fairness annotation associated with the “Determine the beneficiary” task. The specifications of this annotation are shown in a box linked with the fairness annotation. This annotation specifies that the “Determine the beneficiary” task should consider an equal priority for every two patients who need an organ transplant and differ only in their “insurance type” (i.e., public or private insurance) unless there is something that prevents such a difference in their “ages” and/or their “medical histories”. For instance, a privately-insured kid with a critical medical history may receive a higher priority than a publicly-insured aged patient. Although this decision discriminates against the second patient twice, as an aged and a publicly-insured patient, this may be considered legal if it can be justified. The annotation suggests using a “discrimination-free algorithm” (e.g., [22]) to enforce fairness and “0.5” as the threshold for when to consider a data item as a proxy for the “insurance type”.

### 2.4.2 The underlying background of the linkage constraints

In our work, as described in Section 2.4.1, an annotation can be linked with a BPMN element if that element may act as a subject to the corresponding data-protection threat of that annotation. In this section, we give details on the rationale beyond the linkage constraints of our proposed data-minimization and fairness annotations.

The underlying background for our linkage constraints benefits from the research work of Deng et al.[32], that aims to elicit data protection requirements based on the data flow diagram of a targeted system. However, differently from our work, the authors in [32] mapped every data-protection threat to all elements of the data flow diagram, while in our work we argue that some data-protection threats are the consequence of other threats in the system.

Specifically, we mapped the threats to specific elements in a restricted list of BPMN model elements to avoid overlapping meaning of different data-minimization annotations. Generally, Table 2.3 shows that the BPMN *activities*, *data associations*, *data objects*, and *message flows* can be subject to one or more privacy threats.

In our work, as shown in Table 2.3, we consider the following threats:

- identifying data subjects who their real identity should not be uncovered for privacy reasons (*Identifiability*).
- linking different activities/ data from different sources as related (*Linkability*).
- leaking sensitive information about private communications or stored data even if they are encrypted (*Detectability*).
- identifying data subject by insiders and detecting privacy information about them by outsiders if happen together they can lead to the so-called *Observability* threat.
- discriminating data subjects on the ground of *protected characteristics* (e.g., gender) (*Discrimination*).

The aforementioned threats can be mitigated by achieving the data protection goals behind our proposed annotations, respectively, as follows: *Anonymity*, *Unlinkability*, *Undetectability*, *Unobservability*, and *Fairness*. Table 2.3 displays the mapping between the basic elements of BPMN and the set of privacy threats *identifiability*, *linkability*, *detectability*, *observability*, *discrimination*.



Table 2.3: Mapping data protection threats to BPMN elements

Category	Element	Data Protection Threats				
		Identifiability	Linkability	Detectability	Observability	Discrimination
Flow Objects	Activity	●	○	●	○	●
	Event	—	—	—	—	—
	Gateway	—	—	—	—	—
Data Objects	Data Object	○	●	○	○	—
	Data Store	○	●	○	○	—
Connect Objects	Sequence Flow	—	—	—	—	—
	Message Flow	●	○	●	●	—
	Data Association	●	○	●	○	—
Swimlanes	Lane	○	○	○	○	—
	Pool	○	●	○	○	—

The symbol (●) indicates a potential data-protection threat at the corresponding BPMN element. The symbol (○) indicates that the BPMN element can be subject to the corresponding data-protection threats. The symbol (—) indicates that the BPMN element cannot be subject to the corresponding data-protection threats.

Each intersection in Table 2.3 marked with the symbol (●) indicates a potential data-protection threat at the corresponding BPMN element. This is because these elements can be used to either: (i) store sensitive data (e.g., data object), (ii) represent sensitive activity/process (e.g., activity) or (iii) transmit sensitive data (e.g., *message flow*). Consider for instance sensitive messages being transmitted over a message flow. These messages can be subject to a number of data-protection threats such as *identifiability* where adversaries can trace back a message to its actual sender, *detectability* where adversaries are able to distinguish true messages from false ones, and *observability* where both the messages are detectable and the sender of these messages is identifiable by adversaries. Such threats can be mitigated by achieving *anonymity*, *undetectability*, and *unobservability*, respectively.

Entries with the symbol (—) in Table 2.3, indicate that the BPMN element cannot be subject to the corresponding threats. Table 2.3 shows that none of the data-protection threats can be mapped to *events*, *gateways*, and *sequence flows*. The rationale of this is that these BPMN elements do not store, process or transmit personal data. Specifically, as defined in the BPMN 2.0 [1] specification, these elements can be used to either show/control the order of activities in a BPMN model such as *sequence flows* and *gateways* or to represent changes in the system such as *events*. Therefore, we did not consider them as subject to data-protection threats.

The symbol (○) in Table 2.3 indicates that the BPMN element can be subject to the corresponding data-protection threats. However, because these threats are the consequences of other threats, we did not map them to the corresponding BPMN elements. The main reason for that is to avoid having multiple annotations whose semantics overlap, as suggested by Moody's guidelines for increasing the usability of modeling languages [91]. For example, a *data object* that stores personal identifiable information can be subject to the identifiability threat. However, this threat is the result of performing either an activity that retrieves a non-anonymized variant of that data object or linking the data object, in case it is an anonymized data object related to another non-anonymized data object. Since we already mapped the identifiability threat to the *data associations* (to show how the data can be stored to or retrieved from- a data object) and the linkability threat to the *data objects*, there is no need to map the identifiability threat to the data object.

## 2.5 Framework for Detecting Conflicts

In this section, we propose a semi-automated framework for conflict detection between security, data-minimization and fairness requirements. We describe the involved roles (i.e, stakeholders), the main activities of our framework, and the inputs and the outputs of each activity.

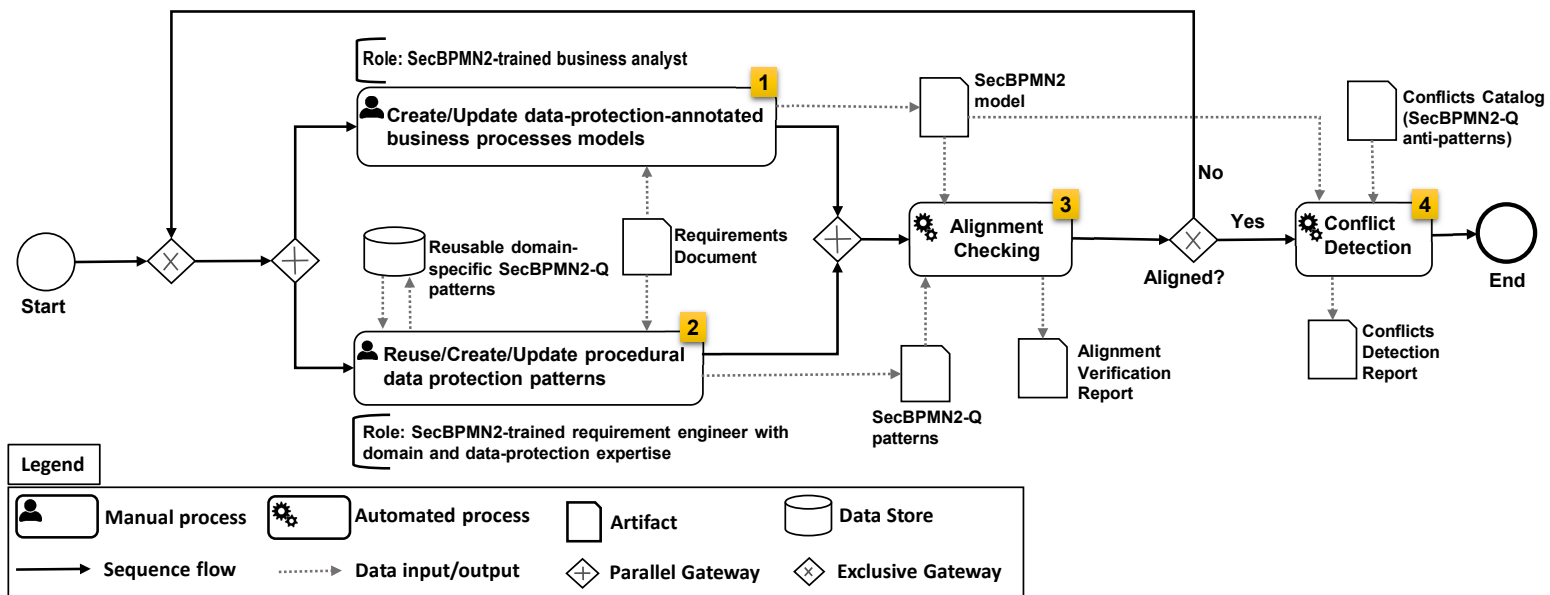


Figure 2.6: The proposed BPMN-based framework for conflict detection.

Detecting conflicts between data protection requirements is a challenging task. Since such conflicts are context-specific, their detection requires a thorough understanding of the underlying business processes [111]. As mentioned in Section 2.1, sometimes the source of conflicts is a non-alignment between the requirements and their implementations in the business processes.

The main source of non-alignment issues is when the business analysts who are responsible for modeling the business processes, misunderstand the requirements specifications or deliberately deviate from them due to business needs [129]. These deviations may result in unexpected conflicts between the requirements that may lead to unwanted vulnerabilities. This is not a new conclusion but a fundamental expectation. For example, Kim et al. [69] define requirements conflicts as "*The interactions and dependencies between requirements that can lead to negative or undesired operation of the system*". To overcome these challenges, we propose in this section a semi-automated process for conflict detection, as shown in Figure 2.6.

Our conflict detection framework permits:

- To ensure that the specified data protection requirements in a business process model are aligned with the elicited requirements. As byproducts, first, conflicts that may happen due to the non-alignment reason can be avoided, and second, the efforts that are needed for specifying the source of conflicts in later stages for resolving conflicts will be reduced, as the possibility that a detected conflict might result from a non-alignment is excluded.
- To detect and report conflicts between data protection requirements that are hard to foresee based on textually-specified requirements (due to the earlier described challenges in Section 2.1.1).

**Roles and assumptions about their skills.** The process of the proposed framework in this section can be executed by a team of *business analysts* and *requirements engineers*. The business analysts are responsible for designing a BPMN model enriched with elicited data protection requirements. The requirements engineers are responsible for modeling elicited requirements as procedural business process patterns that can be automatically matched to the enriched BPMN model with data protection requirements.

The mapping between the elicited requirements and the procedural patterns is not a one-to-one mapping, as it might be preferable to model two or more *supplementing* requirements in one procedural pattern. Two or more requirements are considered as supplementing requirements if they support each other towards achieving shared goals. For example, confidentiality and integrity are two supplementing

security-specific requirements that aim to prevent unauthorized access and modification for system resources. Moreover, since business projects within the same domain can share business practices [43], some data protection patterns can be defined in a *domain-specific* way to allow their reuse in future business projects within the same domain. Thereby, requirements engineers do not need to create patterns for all the elicited requirements every time a new business project starts.

However, decisions about when two or more requirements can be considered as supplementing and when a procedural pattern can be specified in a domain-specific way are critical and require domain knowledge and solid background knowledge about the considered data protection requirements. To this end, our assumptions on the skills of the involved *business analysts* are light-weight, as they do not have to be data protection experts. The involved business analysts have to be SecBPMN2-trained. Still, to ensure the correct use of our approach, some additional background about the used data protection concepts in our approach is appropriate. While in addition to the training, we assume more skills on the involved *requirements engineers*, as they have to be domain experts with a remarkable set of skills in data protection.

With these assumptions, our aim is to address the common situation in which data protection experts are not always available in all system development phases [58]. In this situation, we aim to support non-experts stakeholders with tools to report on non-aligned and conflicting data protection requirements. However, even in presence of data protection experts in every phase of our process—which is clearly the preferable situation—, our process can still be helpful, as the involved expert and non-expert stakeholders benefit from the early conflict detection as during the business processes modeling phase.

**Inputs.** As shown in Figure 2.6, three inputs are required: First, a *requirements document* containing data protection requirements. During the requirements elicitation phase of the system development, the business analysts produce this document while interacting with the users. Second, a set of reusable *domain-specific patterns*, each of them specifying a particular data protection requirement. The patterns are designed as graphical queries using our extension of SecBPMN2-Q. Having a repository of these patterns allows to reuse them over similar projects in the same domain (such as healthcare in our upcoming running example) while ensuring that new patterns can be added on demand. Third, a *domain-independent* conflicts catalog. This catalog is one of our main contributions and it resulted from our analysis of all possible situations where conflicts or potential conflicts between a security, a data-minimization, and a fairness requirement may happen. For each identified situation, we specified an anti-pattern using the SecBPMN2-Q. Details about our catalog of conflicts are described in Section 2.7.

**Outputs.** The outputs of the process are: (i) a *SecBPMN2 model* enriched with data protection requirements; (ii) a set of *SecBPMN2-Q patterns* which can be used to check the alignment of these requirements with the enriched *SecBPMN2 model*; (iii) a textual *alignment verification report* that describes the mismatched patterns in the *SecBPMN2 model*; and (iv) a textual *conflict detection report* that describes the detected conflicts in the *SecBPMN2 model*. On-demand, both the non-aligned and the conflicting requirements can be highlighted in the *SecBPMN2 model*.

The process of our proposed conflict detection framework consists of four phases. The numbers in Figure 2.6 represent the phases, described below.

**Phase 1.** In this phase, business analysts manually model the business processes of the target system, one process at a time, with respect to the data protection needs. The business analysts derive business processes from the provided *requirements document* to create a BPMN model and they use our extension of the *SecBPMN2* language to enrich the BPMN model with data protection requirements, again based on the requirements document. The output model is stored as a *SecBPMN2 model*. Details description on the *SecBPMN2* extension is provided in Section 2.4.

**Phase 2.** In this phase, requirements engineers: First, identify the set of domain-specific patterns to be reused for alignment checking. This step can be performed by mapping data protection requirements extracted from the *requirements document* to existing domain-specific patterns. Second, design procedural patterns for those requirements that do not have corresponding patterns. Newly identified domain-specific patterns may be added to the patterns' repository. Patterns can be formulated in our specialized query language that extends *SecBPMN2-Q* [129]. For managing the patterns, we assume a certain directory structure, based on naming conventions. The management is currently performed manually; we discuss automation opportunities as part of future work.

The output of this phase, as shown in Figure 2.6, is a set of *SecBPMN2-Q patterns* that combines all the new patterns from the second step with the retrieved ones from the first step, which, for alignment checking, will be automatically matched to the *SecBPMN2 model* obtained in *phase 1*. Note that, as depicted in Figure 2.6, *phase 1* and *phase 2* can be executed in parallel. Details on our extension to the *SecBPMN2-Q* are given in Section 2.6.

**Phase 3.** In real-world scenarios, the number of elicited requirements and the size of the business process models tend to be large, making it difficult for the involved business analysts to manually check whether the requirements are correctly specified in the business process models or not [130]. To support an automatic check, we extended the query engine of *SecBPMN2-Q* [129], which permits an alignment checking of security requirements and *SecBPMN2* models based on *SecBPMN2-*

Q patterns. Our extension supports the alignment checking of procedural data-minimization and fairness requirements as well.

As shown in Figure 2.6, the inputs of alignment checking are a *SecBPMN2 model* enriched with data protection requirements from *phase 1*; and the data protection requirements specified as *SecBPMN2-Q patterns* from *phase 2*. The output is an *alignment checking report* that describes the mismatched patterns. In case mismatched patterns are reported, the process in Figure 2.6 starts from the beginning. This will give the stakeholders two options for fixing the sources of mismatching:

- First, in *phase 1*, the requirements specifications in the SecBPMN2 model can be fixed to match their specifications in the mismatched patterns.
- Second, in *phase 2*, the specifications of the mismatched patterns can be relaxed to match their counterparts in the SecBPMN2 model.

The process of fixing the targeted SecBPMN2 model should be repeated until the alignment is ensured; after that *phase 4* can start. More details on alignment checking are given in Section 2.6.

**Phase 4.** Detecting conflicts between data protection requirements during the design of the business processes models can help in reducing the needed effort to fix the conflicts if they are discovered in later development phases. However, data-protection experts cost a lot of money and they are busy with developing new solutions for new data protection challenges so it is preferable to reduce them or optimize their time by automating the process of conflict detection. Moreover, the business process models frequently are composed of many model elements, a manual detection for conflicts between specified data protection requirements in the business process models is a challenging and error-prone task.

One possible scenario to avoid having conflicting requirements in the designed business process is to design a tool that prevents a business analyst from being able to enrich a business process models with conflicting requirements in *phase 1*. However, since the data protection requirements represent preferences for different users in the system, decisions about resolving conflicts during the run-time of designing a business process models cannot be easily taken. Conflicts should be reported and discussed with the stakeholders. Therefore, this phase supports automatic detection and reporting for conflicts between specified security, data-minimization, and fairness requirements in the input model.

As shown in Figure 2.6, the inputs of this phase are: First, a *SecBPMN2 model* that is aligned with the data protection requirements. Second, the domain-independent

*conflicts catalog* as SecBPMN2-Q anti-patterns. These anti-patterns can then be automatically matched to the SecBPMN2 model. The conflict detection benefits from our query engine from *phase 3*. The output of this phase is a *conflict detection report* that textually shows conflicts in the SecBPMN2 model as errors and potential conflicts as warnings to the user. Details about this phase are given in Section 2.7.

## 2.6 Alignment Checking

In this section, we propose an extension to the SecBPMN2-Q query language to allow specifying procedural security, data-minimization and fairness requirements as graphical queries. The queries can be automatically matched to security-, data-minimization-, and fairness-enriched SecBPMN2 models in order to check the alignment between the requirements and their specification in the models. This check helps to avoid unwanted consequences, such as conflicts between the specified requirements in the BPMN models. For example, consider the following two requirements from the healthcare management scenario:

- **Requirement 1:** A telemedicine-device should execute the “Send data to portal” task anonymously by using pseudonyms.
- **Requirement 2:** A telemedicine-device should be accountable when it performs the “Send data to portal” task.

If a business analyst unintentionally uses the anonymity level *fully anonymous* instead of *pseudonymous* for specifying **Requirement1** in Figure 2.2, a conflict with **Requirement2** will arise, since a telemedicine device with the ability to execute the “Send data to portal” task full anonymously cannot be accountable as required by **Requirement2**. Conflicts resulting from such non-alignments between the provided requirements and the enriched BPMN models, if not avoided early, can make the process of locating conflict root causes in later development stages difficult. To avoid these conflicts, the alignment between the requirements specifications and the enriched BPMN models should be ensured, a tedious and error-prone task for smaller models and an infeasible one with models with hundreds of elements.

To overcome this challenge, we present an automated alignment checking technique which takes as input: (i) an enriched *SecBPMN2 model* with security, data-minimization, and fairness annotations, and (ii) a set of security, data-minimization, and fairness requirements specified as *SecBPMN2-Q patterns*. The SecBPMN2-Q patterns can then be automatically matched to the annotated SecBPMN2 model to discover any non-alignment.



### 2.6.1 Modeling SecBPMN2-Q patterns

SecBPMN2-Q supports custom queries enriched with security requirements that can be matched to SecBPMN2 models [129] for alignment checking. We extended SecBPMN2-Q so that it supports our data-minimization and fairness annotations as well, allowing involved requirements engineers to formulate fairness, data-minimization and security requirements as patterns that can be matched to a given SecBPMN2 model.

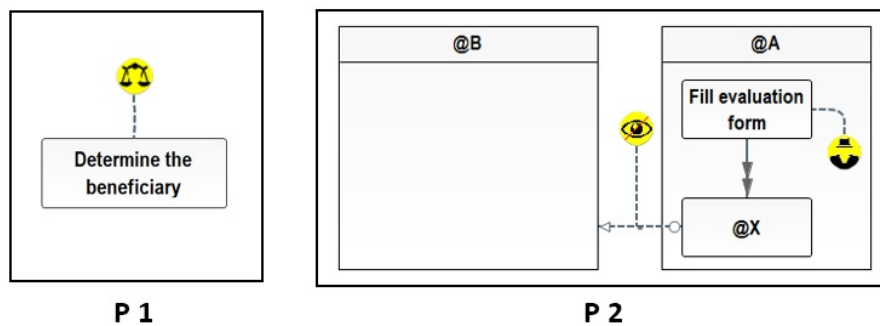


Figure 2.7: Requirements specified as SecBPMN2-Q patterns.

Based on the provided requirements, it might be preferable to: First, encode two or more requirements into one pattern to model situations where two or more different requirements support each other. Second, encode a requirement or a set of requirements as *domain-specific* SecBPMN2-Q patterns that can be reused in future projects within the same domain. Examples of these two ways of encoding requirements into SecBPMN2-Q patterns are captured in Figure 2.7.

Pattern P1 in Figure 2.7 specifies that the business process model should have a *fairness*-annotated task called “Determine the beneficiary”, regardless of its executor. This is because the “Determine the beneficiary” in pattern P1 is not part of a specific *pool* or *swim-lane*, which are usually used to illustrate that participants in the process [1]. Listing 2.1 shows how the properties of the fairness annotation in this pattern are specified. For alignment checking, our query engine considers annotations and their properties as part of the pattern to be matched to the targeted BPMN model.

Listing 2.1: Properties of *Fairness* in pattern P1

```

1 autoPropagated=False,
2 dataSubjectRole={Patient},
3 type=individual fairness,
4 protect={insurance type},
5 useExplanatory={age, medical history},
6 threshold=0.5,
7 enforcedBy={discrimination-free algorithm}.

```

A common practice is to reuse procedural processes from already existing business process models when designing a business process model for a new business project within the same domain. The main reason is the shared business needs between stakeholders of different projects or applications within the same domain [43]. For example, in Figure 2.2, the procedural process that starts by the “Fill donation form” task and ends by the “Determine the beneficiary” task can be reused in the business process models of future clinical projects. The reuse is not restricted only to the procedural description of business processes. If a business process procedure is associated with data protection requirements, the procedure description, and its associated data protection requirements can be considered together when the business process procedure is reused. This is because, within the same domain, the data protection requirements are mainly derived from the same laws and regulations. Therefore, if the procedural part that describe how the organ transplant is managed in our example model, in Figure 2.2, is reused in a new BPMN model, a requirement engineer can reuse pattern P1 to ensure that the fairness specifications on the “Determine the beneficiary” task are correctly specified as in pattern P1.

Considering pattern P2 in Figure 2.7, a label of the form “@” followed by a string acts as a placeholder for element names. This allows assigning any element of the same type when the pattern is matched to the input model. The *walk* relation (illustrated by an edge with double arrowhead), which is can be defined for pairs of activities, events or gateways, allows matching all pairs of elements in the input model for which there is a *control dependency* path between the source and the target element. Therefore, pattern P2 specifies that regardless who are the two participants in the considered process, the “Fill evaluation form” task has to be *anonymity*-annotated and the first message flow that allows transmitting messages between the two participants, after the “Fill evaluation form” task is reached, has to be *unobservability*-annotated. Listings 2.2 and 2.3 show how the properties of the anonymity and the unobservability annotations are specified in pattern P2, respectively.

**Listing 2.2:** Properties of *Anonymity* in pattern P2

```

1 autoPropagated=True,
2 dataSubjectRole={Patient},
3 anonymizedWithin={All Patients},
4 level=fully anonymous,
5 insider=True,
6 enforcedBy={zero-knowledge proof}.

```

**Listing 2.3:** Properties of *Unobservability* in pattern P2

```

1 autoPropagated=False,
2 dataSubjectRole={Patient},
3 anonymizedWithin={All Patients},
4 enforcedBy={Dummy traffic, DC-network}.

```

The reason for specifying the anonymity and the unobservability requirements together in pattern P2 is that a full anonymous accessing for the “Fill evaluation form” task by a participant against insiders (i.e., other participants in the process) can be violated, if there is a control-dependency path between the “Fill evaluation form” task and another task that sends messages to other participants without supporting unobservability or at least full anonymity against insiders during the messages transmission. More precisely, without unobservability the message recipient will be able to trace the messages back to its original sender and, as a result, indirectly violates her anonymity. Also, outsider adversaries will be able to detect that there is a communication between the participants in the process. Since this violation can arise regardless of who the involved participants in the process are and what task sends messages between them, the *Pool* BPMN elements in pattern P2 and the task sending messages between them are labeled with the “@” wildcard to specific a reusable domain-specific pattern.

### 2.6.2 Automated alignment checking

Once the data protection patterns to be checked in the SecBPMN2 model are identified, the business analyst can have the check performed automatically, by using our extension of the SecBPMN2 query engine.

For example, a match for both patterns P1 and P2 in Figure 2.7, together with the specifications of their properties, can be found in the SecBPMN2 model in Figure 2.2. More precisely, with respect to pattern P1, the SecBPMN2 model has a *fairness*-annotated “Determine the beneficiary” task and the specifications of the fairness annotation are similar to those for pattern P1. For pattern P2, the SecBPMN2 model has an *anonymity*-annotated “Fill evaluation form” task that has a control dependency with a task that sends messages over an *unobservability*-annotated message flow to another participant. The specifications of both annotations in the model are also matched to their specifications in pattern P2.

In the case of mismatched patterns, our query engine reports a set of textual messages that describe the mismatches. The involved stakeholders can then fix the SecBPMN2 model so that it matches the reported mismatched patterns, or relax the mismatched patterns so that they match the model. This process should be repeated until all the considered patterns are matched to the model.

By checking the alignment between the requirements and their specifications in the business process models, conflicts that may result from non-alignment can be avoided. Consequently, the results of the actual conflict detection (Section 2.7) are easier to interpret by users. However, performing the alignment check can have

broader positive implications. At its heart, it serves to detect inconsistencies between requirements and design artifacts and therefore may help to avoid costly fixes later in the development process arising from these inconsistencies. Details description of the alignment detection algorithm is provided in Section 2.8.1.

## 2.7 Conflict Detection

We propose an automated conflict detection technique that relies on encoded knowledge about conflicts and potential conflicts between pairs of requirements. Specifically, we propose a catalog of *conflict SecBPMN2-Q anti-patterns* which can be matched against business process models in order to detect *conflicts* and *potential conflicts*. Conflicts represent definitive trade-offs between the requirements; while the potential conflicts may result in conflicts under certain circumstances. Our tool shows conflicts as *errors* and potential conflicts as *warnings* to the user.

### 2.7.1 Automated conflict detection using anti-patterns

The reason for choosing SecBPMN2-Q for automating the conflict detection process is its expressiveness way for encoding conflict anti-patterns. Specifically, SecBPMN2-Q has a relation called *walk* which, as mentioned earlier in Section 2.6.1, allows to match all business processes in which there is a control dependency path between its source and the target elements. This property allowed us to capture conflicts that may arise due to a control dependency path between different requirements for different inter-dependent BPMN elements in the input business process model. In total, we designed 143 domain-independent anti-patterns, which we combined into one catalog. The anti-patterns in our catalog can be classified into four categories, as follows:

- (A) conflicts between data-minimization and security requirements. Out of 143 anti-patterns in our catalog, 47 anti-patterns belong to this category.
- (B) potential conflicts between data-minimization and security requirements. In total, our catalog contains 93 anti-patterns that belong to this category.
- (C) conflicts between data-minimization and fairness requirements. Our catalog contains 2 *constrained* anti-patterns that belong to this category.
- (D) potential conflicts between data-minimization and fairness requirements. Our catalog has 1 *constrained* anti-patterns that belongs to this category.

If an anti-pattern has a match in the business process model, a conflict or potential conflict is reported, respectively. A *constrained* anti-pattern has an additional constraint that must be satisfied so that a conflict or potential conflict arises.

Our patterns are not designed to take into account specific data-protection mechanisms for implementing the given requirements, as can be specified using the *enforcedBy* attribute. While this information may be helpful to identify whether two requirements are conflicting or not, addressing available data protection mechanisms is not feasible because of the following two reasons:

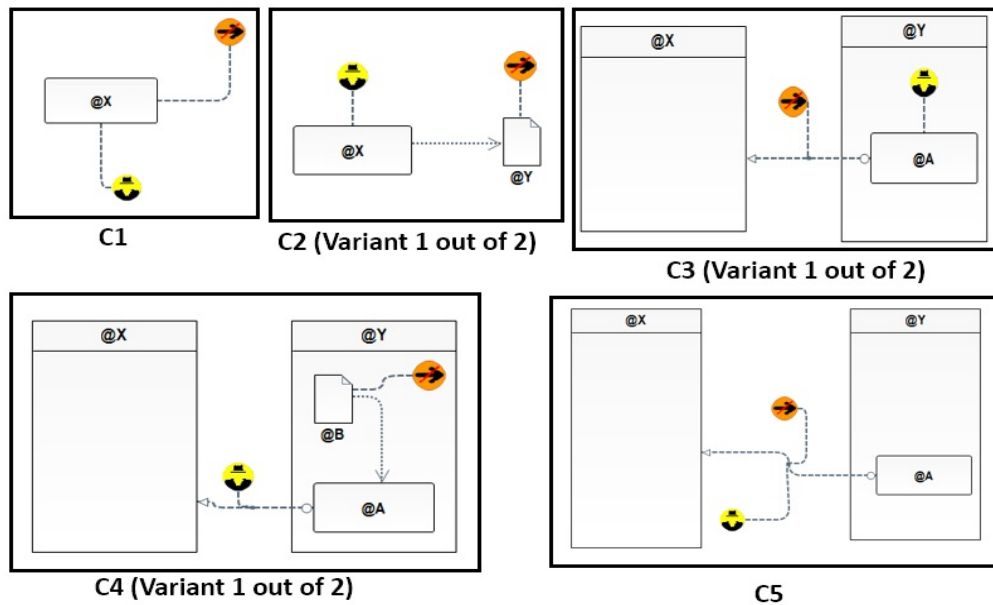
- First, the number of available data protection mechanisms grows continuously [146]. This makes it difficult to have a database that contains updated information about all available data protection mechanisms and their abilities to preserve data protection requirements.
- Second, the knowledge of security engineers on how to design data protection mechanisms and the knowledge of adversaries on how to break data protection mechanisms grows continuously as well [21]. A mechanism that today has the ability to preserve a specific data protection requirement might be broken tomorrow.

Consequently, our anti-pattern catalog would be outdated very soon if it addressed this knowledge. Therefore, our patterns generally do not use the *enforcedBy* attribute. To still raise awareness on possible conflicts, we model situations where the decision on whether two requirements are in conflict depends on the applied mechanisms, as potential conflicts.

In the following, we discuss how our approach can be used to detect conflicts and potential conflicts between security, data-minimization, and fairness requirements by using examples from the four categories of our catalog of anti-patterns. The selected examples do not cover all the (potential) conflicts that can happen between the data protection concepts in our catalog. An overview of our catalog can be found in Section 2.7.2, while a more detailed account is provided in Section 2.7.3. Details description of the conflict detection algorithm is provided in Section 2.8.2.

**A. Conflicts between data-minimization and security requirements.** Conflicts between data-minimization and security requirements occur in two flavors: First, requirements related to the same asset in the system may be conflicting. Second, requirements related to different but dependent assets may be conflicting.

For example on conflicting requirements related to the same asset, consider the *accountability* and *anonymity* annotations linked with the “Send data to portal” task



\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.8:** Conflicts C1–C5 between *non-repudiation* and *anonymity* as anti-patterns.

in the “Tele-medicine Device” pool at the left-hand side of Figure 2.2. For accountability, the system needs to track the executor of this task’s responsibility, while the anonymity annotation specifies that the executor should be fully anonymous against insider adversaries.

For example on conflicting requirements related to different but dependent assets, in Figure 2.2 at the right-hand side, consider the *anonymity* and *non-repudiation* annotations linked with the “Fill evaluation form” task and the “Evaluation form” data object, respectively. The former imposes that an executor to the “Fill evaluation form” task should be fully anonymous against insider adversaries; the latter indicates that an accessor to the “Evaluation form” data object should not be able to deny that she accessed the “Evaluation form”. Since the “Fill evaluation form” task writes data to the “Evaluation form” data object, a conflict is reported.

In Figure 2.8 we show a selection of anti-patterns defined using our SecBPMN2-Q extension. Together, the depicted anti-patterns represent all conflicts that can happen between *anonymity* and *non-repudiation*. As explained in Section 2.4.1, the anonymity level can be either full anonymous or pseudonymous. A pseudonym is an identifier for a data subject other than one of the data subject’s personal identifiable information. However, a pseudonym can be linked with the real identity of a data subject for accountability reasons. Hence, an anonymity require-

ment with pseudonyms is not conflicting with a non-repudiation requirement. Also a full anonymous against outsider adversaries is not conflicting with non-repudiation, as non-repudiation requirement aims at keeping data subjects accountable with respect to insiders. Therefore, the only situation where anonymity and non-repudiation requirements are in conflict is when the anonymity level is fully anonymous and against insiders. Therefore, as provided in Figure 2.8, all anonymity annotations in shown anti-patterns in Figure 2.8 are specified with the following attributes:  $\{level=full\ anonymous, insider=true\}$ .

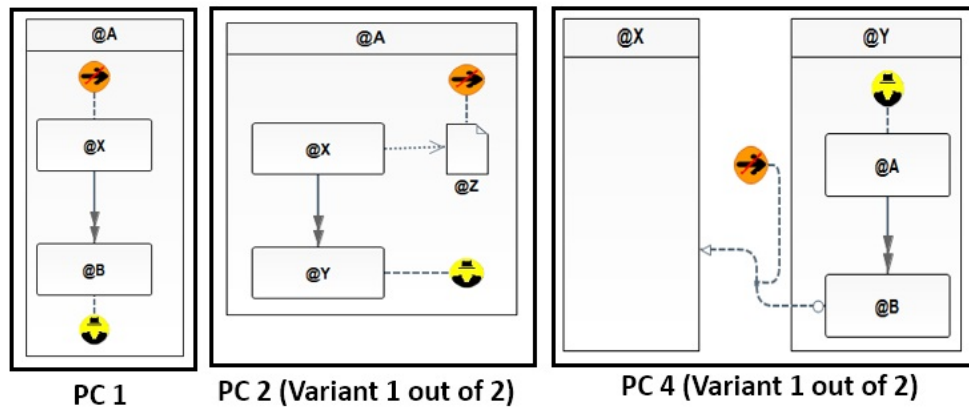
Consider, for example, conflicts C1 and C5 in Figure 2.8. These conflicts arise when *non-repudiation* and *anonymity* annotations are linked to the same task or message flow, respectively. C1 can be matched to one place in the example model in Figure 2.2, as follows: in the “Patient” pool at the right-hand side of the model, the *anonymity* annotation of the “Fill evaluation form” task is propagated to the “Submit evaluation” task, which is annotated with a *non-repudiation* annotation.

The anti-patterns C2, C3 and C4 in Figure 2.8 come in two variants since the data object call (read or write) can be inverted, and the assignment of requirements to elements can be swapped. The anti-pattern C2 can be matched to two places in the example model, as follows: First, in the “Patient” pool, the *anonymity*-annotated “Fill evaluation form” task is writing a data to the “Evaluation form” data object, which is annotated with a *non-repudiation* annotation. Second, in the “Patient” pool, the “Submit evaluation” task is annotated with *anonymity* (due to the propagation of the anonymity annotation that is linked with the “Fill evaluation form” task) and it is reading data from the *non-repudiation*-annotated “Evaluation form” data object.

The anti-pattern C3 can be matched to one place in the example model, as follows: the “Submit evaluation” task in the “Patient” pool is annotated with *anonymity* and it is sending messages over the *non-repudiation*-annotated message flow between the “Patient” and the “Emergency Unit” pools.

In contrast to the anti-patterns C1-C3 in Figure 2.8, C4 and C5 does not occur in the example model. Specifically: First, C4 does not have a match in the example model in Figure 2.2 because the model does not have a task that reads or writes data to/from a *non-repudiation*-annotated data object and in the same time sends messages over an *anonymity*-annotated message flow. Second, C5 does not occur in the example model, since the model does not have an *anonymity*- and *non-repudiation*-annotated message flow.

**B. Potential conflicts between data-minimization and security requirements.** Potential conflicts between security and data-minimization requirements as considered in our work result from control dependencies between activities with specified requirements. For example, Figure 2.2 includes a control dependency path between



\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.9:** Potential conflicts between *non-repudiation* and *anonymity* as anti-patterns.

the *anonymity*-annotated “Fill evaluation form” task and the *non-repudiation*-annotated “Submit evaluation” task. As explained in the next paragraph, such situations not necessarily give rise to a conflict.

Imagine, for example, a control dependency between two tasks where the first task allows a customer to anonymously use a service and the second task allows the service provider to prevent a customer from being able to deny his payment for receiving a service. In this situation, it may be sufficient for a service provider to prove that a customer performed the payment task without uncovering which service a customer is paying for, and as a consequence, preserve the customer anonymity. Potential conflicts should be reported and discussed.

In Figure 2.9 we show three anti-patterns specifying potential conflicts between *anonymity* and *non-repudiation*. In these patterns, we use a *walk* relation to match all pairs of elements in the input model for which there is a control dependency path between the source and the target element. Note, Figure 2.9 shows only 3 out of 8 overall potential conflicts for the considered pair of requirements. Two additional cases called PC3 and PC5 are formed in analogy to C3 and C5 in Figure 2.8; three additional variants arise from duality like in the discussion of the conflicts. Again, as provided in Figure 2.9, all anonymity annotations are specified with the following attributes:  $\{anonymity\ level=full\ anonymous, insider=true\}$ . We specified these attributes with these values for the same reason in Figure 2.8.

The potential conflict PC1 in Figure 2.9 illustrates the situation where a potential conflict can arise due to the existence of a control dependency path between an



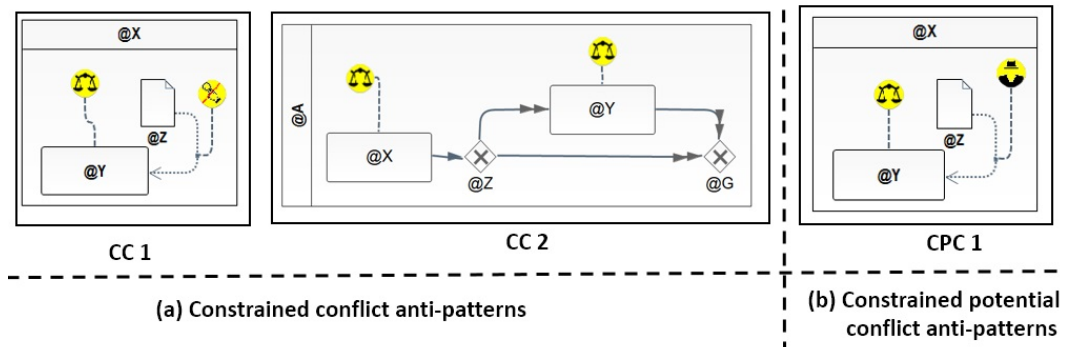
*anonymity*-annotated task and a *non-repudiation*-annotated task. PC2 represents the situation where a potential conflict can arise due to a control dependency path between a task that reads or writes data from/to a *non-repudiation*-annotated data object and an *anonymity*-annotated task. While PC4 specifies a path between an *anonymity*-annotated task and another task that sends messages over a *non-repudiation*-annotated message flow. All these situations may lead to conflict, depending on the actual circumstances in the system.

Using PC1, PC2 and PC4 of Figure 2.9 with the example model in Figure 2.2, four warnings will be reported, due to the following reasons:

- The example model has a control dependency path between the *anonymity*-annotated “Fill evaluation form” task and the *non-repudiation*-annotated “Submit evaluation” task, thus violating PC1.
- PC2 is violated twice:
  - (i) the example model has a control dependency path between the *anonymity*-annotated “Submit evaluation” task and the “Fill evaluation form” task, which writes data to the *non-repudiation*-annotated data object.
  - (ii) the example model has a control dependency path between the “Submit evaluation” task, which reads data from the *non-repudiation*-annotated data object, and the *anonymity*-annotated “Fill evaluation form” task.
- The example model has a control dependency path between the *anonymity*-annotated “Fill evaluation form” task and the “Submit evaluation” task which sends messages over a *non-repudiation*-annotated message flow, leading to a violation of PC4.

**C. Conflicts between data-minimization and fairness requirements.** From considering all possible combinations of requirements, we identified two critical situations that can lead to conflicts. We captured these situations as *constrained* conflict anti-patterns (shown in Figure 2.10.a). If the constrained-conflict anti-patterns CC1 and CC2 are matched in the input BPMN model conflicts are only reported if a corresponding constraint (described below) is satisfied. The constraints check how some relevant attributes of the annotations in the BPMN model are related.

The undetectability annotation in the constrained-conflict CC1 is specified with the attribute *{insider=true}*. The fairness annotations in both constrained conflicts CC1 and CC2 are not further specified. This means these anti-patterns can be matched to the input model irrespective of the attributes’ values of its fairness annotations.



\*\* The undetectability annotation in CC1 and the anonymity annotation in CPC1 are specified with the attribute *{insider=true}*

**Figure 2.10:** Conflicts and potential conflicts between *fairness*, *undetectability*, and *anonymity* as constrained anti-patterns.

The constrained conflict CC1 specifies the situation where a *fairness*-annotated task reads *undetectability*-constrained data from a data object. Generally, to preserve fairness, the *fairness*-annotated task in the model should have access to the data defined by the *protect* and the *useExplanatory* attributes of the fairness annotation. These two attributes, as explained in Section 2.4.1, define the protected characteristics that the output of the *fairness*-annotated activity should be fair with respect to, and the data that, from the perspective of the domain experts, is legitimate to use as input to the *fairness*-annotated task. If *group fairness* is required, the task needs to access what is defined as protected characteristics in the fairness annotation to ensure equal distribution for its outputs between the protected group fractions. In the case of *individual fairness*, the task needs to access what is defined as protected to find out which of its inputs may act as a proxy for protected characteristics [7]. This is important to avoid influencing the outputs of an *individual-fairness*-annotated task by protected characteristics.

With respect to the explanatory data, a *fairness*-annotated task needs to access what is defined as explanatory data by the fairness annotation to achieve business needs, regardless of the required type of fairness. Therefore, if CC1 is matched in a business process model and the matched undetectability annotation in the model requires undetectability for data specified as *protect* or *explanatory* data by the fairness annotations in the model, conflict should be reported. This is because it is impossible to have the same data undetectable and using it as input to fulfill the fairness requirement in the context described by the constrained conflict CC1.

The graphical pattern for CC1 alone is not sufficient to implement this intuition, since it does not check whether the undetectability and fairness annotations refer to

the same data, thus inducing a conflict. Therefore, in addition to matching the anti-pattern, our query engine evaluates an additional constraint on how the relevant attributes are specified in the model. Given a SecBPMN2 model  $m$  and a match of CC1 to a part of  $m$  that includes the fairness annotation  $fair$  and the undetectability annotation  $undetict$ , a conflict is reported if:

$$(fair.getProtected() \cup fair.getExplanatory()) \cap undetict.getProtected() \neq \emptyset$$

Here,  $fair.getExplanatory()$  and  $fair.getProtected()$  retrieve the set of data defined by  $fair$  as explanatory data and protected characteristics, respectively, while  $undetict.getProtected()$  retrieves the set of data specified as protected by  $undetict$ .

Matching the anti-pattern CC1 to the example model in Figure 2.2 yields one conflict to be reported: There is a match for CC1, as the “System Portal” pool has an *undetictability*-annotated data input association between the “Patient in need of a transplant” data store (represented as a data object) and the fairness-annotated “Determine the beneficiary” task. The constraint is fulfilled since there is an intersection between the data that should be undetectable when the task reads data from the data store and the data that are defined as *protected* and *explanatory* in the fairness annotation. More specifically, the “insurance type” and the “age” data which are, respectively, defined as part of the *protect* and the *useExplanatory* attributes of the fairness annotation, are also required to be undetectable, as specified by the *protect* attribute of the undetectability annotation.

The constrained conflict CC2 in Figure 2.10 describes the critical situation where a fairness-annotated task is *control-dependent* on another fairness-annotated task, in the sense that the one task is only reached under some condition (here expressed by gateway “@Z”) that depends on the output of the other task. Irrespective of the required fairness type of each fairness annotation, this situation may lead to a conflict if the explanatory data set that is defined by the fairness annotations of first task intersects with the protected characteristics set that is defined by the fairness annotation of the second task.

For example, let “identify insurance tariff” and “identify reimbursement factor” be two decision-making activities in a private health insurance company. The first activity aims to decide about the *insurance tariff* that a customer should have, while the second activity aims to decide about the *reimbursement factor* that the insurance company can offer for a customer. In this company, depending on the output of the “identify insurance tariff” decision-making activity, the customer’s *reimbursement factor* will be either the output of executing the “identify reimbursement factor” decision-making activity or a fixed value that can be retrieved from a predefined list of reimbursement factors. Irrespective the type of fairness (i.e., group- or individual-fairness) that each decision-making activity aims to provide, assume the following:

- First, the *gender* is defined as explanatory data for the “identify insurance tariff” activity, as it might be legally authorized to use the *gender* for identifying the insurance tariffs of customers.
- Second, the *gender* is defined as protected characteristic for the “identify reimbursement factor” activity, as it might be not allowed to use *gender* for deciding about the reimbursement factor.

Since “identify insurance tariff” discriminates on *gender* to identify the insurance tariff and since the decision of whether to execute “identify reimbursement factor” or not depends on the insurance tariff, neither individual-fairness nor group-fairness can be achieved. More specifically, based on the *insurance tariff* which can act as a proxy for *gender*: First, two customers who differ only in their *gender* will receive different reimbursement factors. This is because one customer will receive the reimbursement factor as an output for executing the “identify reimbursement factor” activity, while the second customer will receive it as a retrieved value from a predefined list of reimbursement factors, which does not guarantee equal reimbursement factors and, as a consequence, violating individual-fairness with respect to *gender*. Second, since *males* customers will receive insurance tariffs different than *females* customers, not all of them will receive their reimbursement factors as outputs for “identify the reimbursement factor”. Hence, the reimbursement factors will not be equally distributed between the fractions of both males and females, which is a violation of group-fairness with respect to *gender*.

The necessary agreement between the attributes is captured in the following constraint. Given a SecBPMN2 model  $m$  and a match of CC2 to a part of  $m$  that includes the fairness annotations  $fair_1$  and  $fair_2$  for the tasks to which “@X” and “@Y” are mapped, respectively, a conflict is reported if:

$$fair_1.getExplanatory() \cap fair_2.getProtected() \neq \emptyset$$

Here,  $fair_1.getExplanatory()$  and  $fair_2.getProtected()$  retrieve the set of data that are defined as explanatory data and protected characteristics, respectively.

From matching the constrained conflict CC2 to the example model in Figure 2.2, no conflicts are reported, since CC2 has no matches in the example model.

**D. Detecting potential conflicts between data-minimization and fairness requirements.** Our catalog of *conflict anti-patterns* contains one *constrained* anti-pattern that if matched in the input model and if its corresponding constraint is satisfied, detects a potential conflict. Figure 2.10 shows the constrained potential conflict CPC1. The anonymity annotation in CPC1 is specified with the following attribute:  $\{insider=true\}$ . For the fairness annotation, no attribute values are specified.

The anti-pattern CPC1 specifies the critical situation where a *fairness*-annotated task reads anonymized data from a data object. This situation, if matched in the input model, may lead to a potential conflict if the set of data specified by the *protect* and the *useExplanatory* attributes of the fairness annotation intersects with the data to be anonymized when the fairness-annotated task reads data from the data object.

A decision on whether the situation in CPC1 represents a conflict depends on the applied anonymization technique. In some situations, providing anonymized data to a *fairness*-annotated task may not prevent it from being able to [52]:

- First, detect proxies for protected characteristics in the case of individual-fairness is required.
- Second, produce equally distributed outputs fractions between protected groups in the case of group-fairness is required.

A potential conflict between a fairness and an anonymity requirement arises if the following constraint is fulfilled. Given a SecBPMN model  $m$  and a match of CPC1 to a part of  $m$  that includes the fairness annotation  $fair$  and the anonymity annotation  $anon$ , a conflict is reported if:

$$(fair.getProtected() \cup fair.getExplanatory()) \cap anon.getProtected() \neq \emptyset$$

Here,  $anon.getProtected()$  retrieves the set of data that are specified as protected by the matched anonymity annotation in  $m$ , while  $fair.getExplanatory()$  and  $fair.getProtected()$  retrieve the set of data that are defined by the matched fairness annotation in  $m$  as explanatory data and protected characteristics, respectively.

From matching the constrained potential conflict CPC1 to the example model in Figure 2.2, no conflicts are identified, as the anti-pattern has no matches in the example model.

### 2.7.2 Catalog of domain-independent conflicts: an overview

As provided in Section 2.2.6, SecBPMN2 supports 10 different security requirements. Similar to our proposed annotations, the same security annotations can be linked to different BPMN elements. We analyzed the situations where conflicts or potential conflicts between two data-protection annotations, i.e., security, data-minimization, or fairness annotations, may arise. For each identified situation, we specified an anti-pattern using our extension of SecBPMN2-Q.

**Table 2.4:** Overview of conflict + potential conflict anti-patterns per pair of requirements.

<b>Requirement pair</b>	<i>Accountability</i>	<i>Authenticity</i>	<i>Auditability</i>	<i>Non-repudiation</i>	<i>Non-delegation</i>	<i>Binding of Duties</i>	<i>Separation of Duties</i>	<i>Anonymity</i>	<i>Confidentiality</i>	<i>Integrity</i>	<i>Availability</i>	<i>Undetectability</i>	<i>Fairness</i>
<b>Anonymity</b>	2+2	6+6	8+8	8+8	2+2	1+0	1+0	4+5	0+8	0+11	0+11	0+0	0+1*
<b>Unlinkability</b>	0+0	0+0	0+0	0+0	0+0	0+1	0+0	0+0	0+0	0+0	0+0	0+0	0+0
<b>Unobservability</b>	1+1	3+3	4+4	4+4	1+1	0+0	0+0	2+2	0+4	0+6	0+6	0+0	0+0
<b>Fairness</b>	0+0	0+0	0+0	0+0	0+0	0+0	0+0	0+1*	0+0	0+0	0+0	1+0	1+0

To assure the correctness and completeness of our technique, each possible combination of requirements and their resulting conflicts were discussed by at least two experts in data protection. In absence of a more formal validation for the correctness and completeness of our technique, a manual check complementing our automated one may be desirable; still we argue that our automated check can support developers in effectively discovering conflicts in the system (see Section 2.10). Together our anti-patterns represent a basic catalog for finding conflicts that can be extended by advanced users, thus improving the confidence in the overall approach. Table 2.4 gives an overview of the resulting catalog of anti-patterns.

Specifically, Table 2.4 views all pairs of requirements for which we identified a (potential) conflict of the resulting catalog of anti-patterns. A more detailed account is provided in Section 2.7.3. Each cell in Table 2.4 shows the number of conflicts, plus the number of potential conflicts between the considered pair of requirements. For example, there are 8 conflicts and 8 potential conflicts for non-repudiation and anonymity. The origin of these numbers is explained in the previous descriptions of Figure 2.8 and Figure 2.9. The other numbers arise from the various possibilities of linking a data-minimization or a security annotation with other BPMN elements. In total our catalog contains 143 anti-patterns.

Considering conflicts and potential conflicts, *Accountability*, *Authenticity*, *Audibility*, and *Non-delegation* represent different requirements to keep insider users accountable for their actions. To preserve them, the identity of an action's executor must be verified. Therefore, all of these security concepts may have conflicts or potential conflicts with *Anonymity* (where required against insiders) and *Unobservability*, since part of its definition implies full anonymity against insiders.

While the *Separation* and *Binding of duties* can conflict with *Anonymity* if any of the activities to which they are applied also require to be executed anonymously. For instance, it will be hard, in case of *Binding of Duties*, to prove that two fully-anonymously executed activities are executed by the same person or not.

A potential conflict between the *Binding of duties* and *Unlinkability* is also possible. For example, as shown in Figure 2.5, *Unlinkability* can be linked with two pools to indicate that the two process executions can not be linked to each other as related. Therefore, it may conflict with *Binding of Duty*.

*Confidentiality*, *Integrity*, and *Availability* represent different requirements to allow authorized users to read, modify, or access a system asset, respectively. The satisfaction of these requirements relies on authorization, which, however, does not necessarily imply identification: The literature provides many techniques for performing authorization without uncovering the real identity of an action executor, for example, *zero-knowledge protocols* [90]. However, the system developers may

choose to implement these requirements by a mechanism that relies on identification, such as access control, which may lead to conflicts with data-minimization requirements. Hence, a decision about whether a conflict arises cannot be made at the abstraction level of process models. Therefore, as shown in Table 2.4, we classified the interactions between these security requirements and the data-minimization requirements as potential conflicts.

In some cases, *Confidentiality* and *Integrity* are considered as supplementing requirements to *Anonymity* [54]. For instance, anonymity against outsider adversaries implies that the outsider adversaries should not be able to trace a message back to its sender. However, if the sent message contains personal identifiable information and it is sent in clear (i.e., without encryption), an outsider attacker can easily link the messages to its sender. This kind of interaction can not be considered as conflicts or potential conflicts, and thus, they are omitted from Table 2.4.

Conflicts may not only occur between different categories of requirements (e.g., security requirements vs data-minimization requirements). Table 2.4 indicates that a particular *Anonymity* annotation might conflict with other *Anonymity* or *Unobservability* annotations. For example, requiring full anonymous accessing for an activity against insiders is in conflict with requiring the output of the same task to be sent anonymously using the *pseudonyms* of the accessors for the task.

Since the security concepts that are considered in our work aim either to monitor the *insiders'* activities such as (e.g., *Non-Repudiation*) or to prevent malicious access to assets that store or process data such as (e.g., *Confidentiality*), they can not have conflicts with *Fairness*. This is because *Fairness* is not about preventing access to data but it is about controlling the use for the data when access to them is provided. For this, during our analysis for all the possible situations where a security or a fairness annotation can interact with each other, we did not detect a situation that may lead to a conflict or a potential conflict between fairness and all other security concepts. In specific situations, a *Fairness* requirement can have conflicts with other fairness requirement or (potential) conflicts with data-minimization requirements. As shown in Table 2.4, *Fairness* can have conflicts with *Undetectability* or another *Fairness* requirement, and potential conflicts with *Anonymity*. The anti-patterns of these conflicts and potential conflicts are shown in Figure 2.10 and discussed in the previous section. The two intersections in Table 2.4 with a (\*) symbol represent the same potential conflict between fairness and anonymity.

Different from all other anti-patterns in our catalog, the three anti-patterns in Figure 2.10 are constrained anti-patterns, meaning that a match of any of them in the input model is insufficient to report a conflict. In addition, an associated constraint has to be checked in order to detect a conflict, as explained in detail in Section 2.7.1.



### 2.7.3 Full textual description of the proposed catalog of conflicts

This section provides a textual description of our catalog of conflicts (C) and potential conflicts (PC) between security, data-minimization and fairness requirements. It textually describes 143 anti-patterns. All the anti-patterns are specified by our extension to the SecBPMN2-Q and they are provided as part of our tool support online at <http://www.sts-tool.eu/downloads/secbpmn-dm>. Information of our tool support is provided in Section 2.8.

Figure 2.11 is a simplified version of our running example in Figure 2.2 and it is annotated with different security and data-minimization annotations, in comparison with Figure 2.2. In the following, we refer to Figure 2.11 to discuss conflicts and potential conflicts that occur due to matches between anti-patterns in our proposed catalog and the model in Figure 2.11.

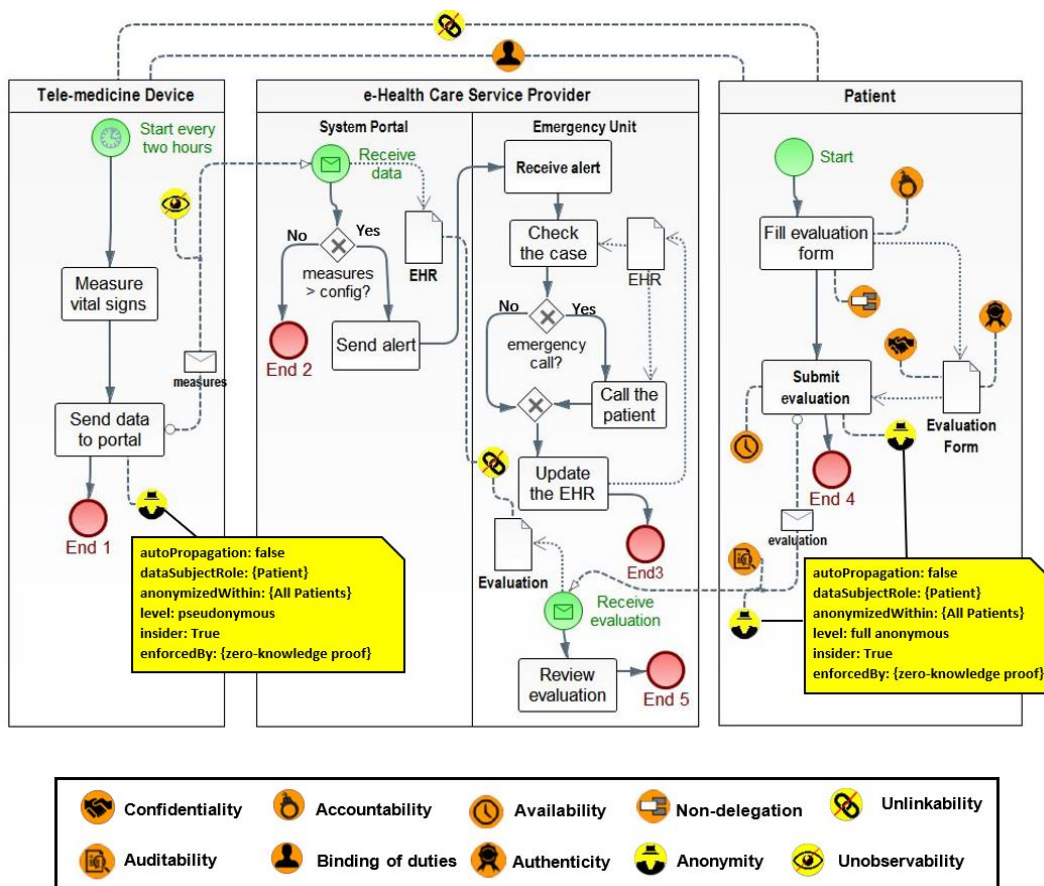


Figure 2.11: Example 2: A simplified version of our running example model with different security and data-minimization annotations.

**Non-repudiation.** *Non-repudiation* in SecBPMN2 can be linked with: First, an activity to indicate that it should be possible to prevent an executor from being able to deny that she executed the activity. Second, a data object to impose that it should be possible to prevent an accessor to the data object from being able to deny that she read/modified data in that object. Third, a message flow to specify that it should be possible to prevent a sender from being able to deny that she sent messages.

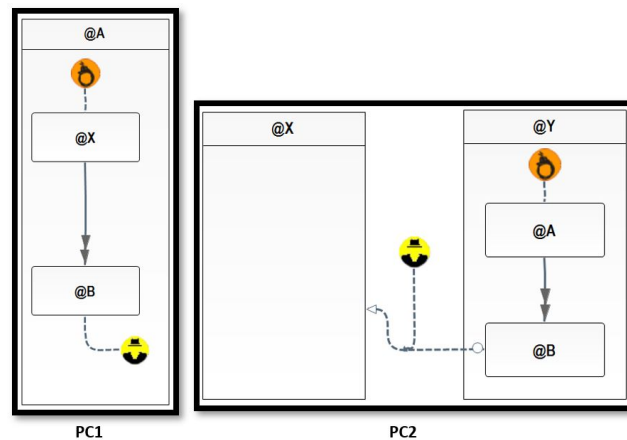
- **C1.1 Non-repudiation vs Anonymity.** A conflict between an anonymity and a non-repudiation requirement occurs if the business process model has: (1) an *Anonymity*- and a *Non-repudiation*-annotated task, (2) an *Anonymity*- and *Non-repudiation*-annotated message flow, (3) an *Anonymity*-annotated task reads data from a *Non-repudiation*-annotated data object (because of different data directions, two anti-patterns are defined), (4) an *Anonymity*-annotated task sends messages over a *Non-repudiation*-annotated message flow or vice versa (because of two possible representations, two anti-patterns are defined), or (5) a task that reads data from a *Non-repudiation*-annotated data object and sends messages over an *Anonymity*-annotated message flow (because of different data directions, two anti-patterns are defined).
- **C1.2 Non-repudiation vs Unobservability.** A conflict between an unobservability and a non-repudiation requirement occurs if the business process model has: (1) an *Unobservability*- and a *Non-repudiation*-annotated message flow, (2) a *Non-repudiation*-annotated task sends messages over an *Unobservability*-annotated message flow, or (3) a task that reads data from a *Non-repudiation*-annotated data object and sends messages over an *Unobservability*-annotated message flow (because of different data directions, two anti-patterns are defined).
- **PC1.1 Non-repudiation vs Anonymity.** A potential conflict between an anonymity and a non-repudiation requirement occurs if the business process model has: (1) a path between an *Anonymity*-annotated task and a *Non-repudiation*-annotated task, (2) a path between an *Anonymity*-annotated task and a task that writes data to a *Non-repudiation*-annotated data object (because of different data directions, two anti-patterns are defined), (3) a path between a task that sends messages over a *Non-repudiation*-annotated message flow and a task that sends messages over an *Anonymity*-annotated message flow, (4) a path between an *Anonymity*-annotated task and a task that sends messages over a *Non-repudiation*-annotated message flow or vice versa (because of the different possible representations, two anti-patterns are defined), or (5) a path between a task that reads data from a *Non-repudiation*-annotated data object and a task that sends messages over an *Anonymity*-annotated message flow (because of different data directions, two anti-patterns are defined).

- **PC1.2 Non-repudiation vs Unobservability.** A potential conflict between an unobservability and a non-repudiation requirement occurs if the business process model has: (1) a path between a task that sends messages over an a *Non-repudiation*-annotated message flow and a task that sends messages over an *Unobservability*-annotated message flow, (2) a path between a *Non-repudiation*-annotated task and a task that sends messages over *Unobservability*-annotated message flow, or (3) a path between a task that reads data from a *Non-repudiation*-annotated data object and a task that sends messages over an *Unobservability*-annotated message flow (because of different data directions, two anti-patterns are defined).

Examples on conflicts between non-repudiation and data-minimization annotations are already discussed in Section 2.7.1.

**Accountability.** *Accountability* in SecBPMN2 can be linked with activities and it specifies that the system should hold the executors of the activities responsible for their actions. Accountability may conflict with anonymity and unobservability.

- **C2.1 Accountability vs Anonymity.** A conflict between an anonymity and an accountability requirement happens if the business process model in question has any of the followings: (1) an *Anonymity*- and *Accountability*-annotated task, or (2) an *Accountability*-annotated task that sends messages over an *Anonymity*-annotated message flow.
- **C2.2 Accountability vs Unobservability.** A conflict between an unobservability and an accountability requirement occurs if the business process model has: (1) an *Accountability*-annotated task that sends messages over an *Anonymity*-annotated message flow.
- **PC2.1 Accountability vs Anonymity.** A potential conflict between an anonymity and an accountability requirement occurs if the business process model has: (1) a path between an *Anonymity*-annotated task and an *Accountability*-annotated task, or (2) a path between an *Accountability*-annotated task and a task that sends messages over an *Anonymity*-annotated message flow. PC1 and PC2 in Figure 2.12 are two SecBPMN2 anti-patterns that represent the first and the second situation, respectively. PC1 can be matched to one place in the model in Figure 2.11, as follows: in the “Patient” pool there is a path between the *accountability*-annotated “Fill evaluation form” task and the *anonymity*-annotated “Submit evaluation” task. PC2 can be matched to one place in Figure 2.11, as follows: in the “Patient” pool there is a path between the *accountability*-annotated “Fill evaluation form” task and the “Submit evaluation” task which sends messages over *anonymity*-annotated message flow.



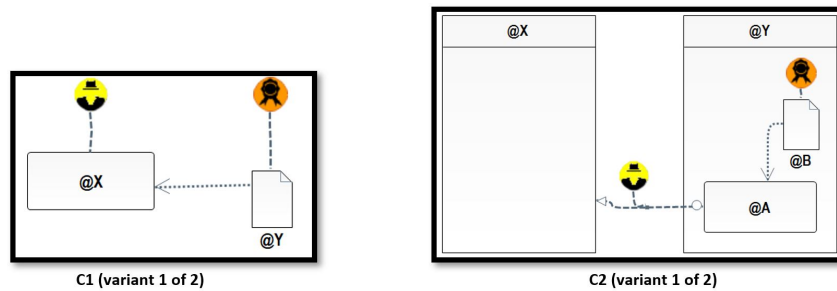
\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.12:** Potential conflicts PC1 and PC2 between *accountability* and *anonymity* as anti-patterns.

- **PC2.2 Accountability vs Unobservability.** A potential conflict between an unobservability and an accountability requirement occurs if the business process model has: (1) a path between an *Accountability*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow.

**Authenticity.** *Authenticity* in SecBPMN2 can be linked with: First, an activity to impose that the identity of the activity executor must be verified. Second, a data object to indicate that it should be possible to prove the data object is genuine.

- **C3.1 Authenticity vs Anonymity.** A conflict between an anonymity and an authenticity requirement occurs if the business process model has: (1) an *Anonymity*- and *Authenticity*-annotated task, (2) an *Anonymity*-annotated task that writes data to an *Authenticity*-annotated data object (because of different data directions, two anti-patterns are defined), (3) an *Authenticity*-annotated task that sends messages over an *Anonymity*-annotated message flow, or (4) a task that reads data from an *Authenticity*-annotated data object and sends messages over an *Anonymity*-annotated message flow (because of different data directions, two anti-patterns are defined). C1 and C2 in Figure 2.13 are two SecBPMN2 anti-patterns that represent the second and the fourth situation, respectively. C1 can be matched to one place in the model in Figure 2.11, as follows: in the “Patient” pool the *anonymity*-annotated “Submit evaluation” task reads data from the *authenticity*-annotated “Evaluation form” data object. C2 can be matched to one place in the model in Figure 2.11, as follows: in the “Patient” pool the “Submit evaluation” task is sending messaging

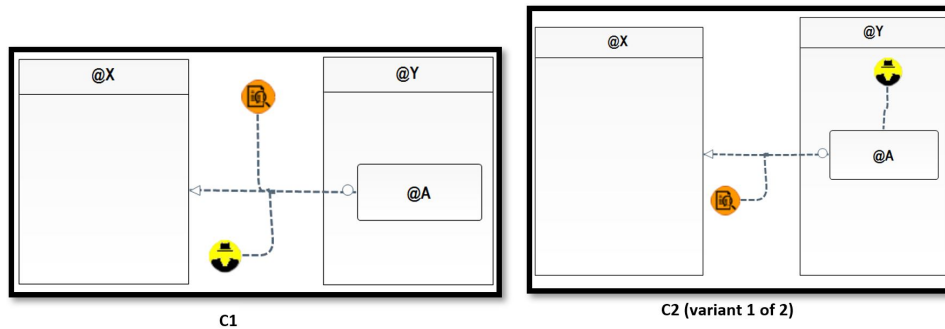


\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.13:** Conflicts C1 and C2 between *authenticity* and *anonymity* as anti-patterns.

over an *anonymity*-annotated message flow and it reads data from the *authenticity*-annotated “Evaluation form” data object.

- **C3.2 Authenticity vs Unobservability.** A conflict between an unobservability and an authenticity requirement occurs if the business process model in question has: (1) an *Authenticity*-annotated task that sends messages over an *Unobservability*-annotated message flow, or (2) a task that reads data from an *Authenticity*-annotated data object and sends messages over an *Unobservability*-annotated message flow (because of different data directions, two anti-patterns are defined).
- **PC3.1 Authenticity vs Anonymity.** A potential conflict between an anonymity and an authenticity requirement occurs if the business process model has: (1) a path between an *Anonymity*-annotated task and an *Authenticity*-annotated task, (2) a path between an *Anonymity*-annotated task and a task that writes data to an *Authenticity*-annotated data object (because of different data directions, two anti-patterns are defined), (3) a path between an *Authenticity*-annotated task and a task that sends messages over an *Anonymity*-annotated message flow, or (4) a path between a task that reads data from an *Authenticity*-annotated data object and a task that sends messages over an *Anonymity*-annotated message flow (because of different data directions, two anti-patterns are defined).
- **PC3.2 Authenticity vs Unobservability.** A potential conflict between an unobservability and an authenticity requirement occurs if the business process model has: (1) a path between an *Authenticity*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow, or (2) a path between a task that reads data from an *Authenticity*-annotated data object and a task that sends messages over an *Unobservability*-annotated message flow (because of different data directions, two anti-patterns are defined).



\*\* All the anonymity annotations are specified as follows: *{level=full anonymous, insider=true}*

**Figure 2.14:** Conflicts C1 and C2 between *auditability* and *anonymity* as anti-patterns.

**Audibility.** *Audibility* in SecBPMN2 can be linked with: First, an activity to indicate that it should be possible to keep track of all the actions performed by the executor of the activity. Second, a data object to impose that it should be possible to keep track of all the read and write actions concerning the data object. Third, a message flow to specify that it should be possible to keep track of all the actions executed to handle the communication (send/receive actions) within the message flow.

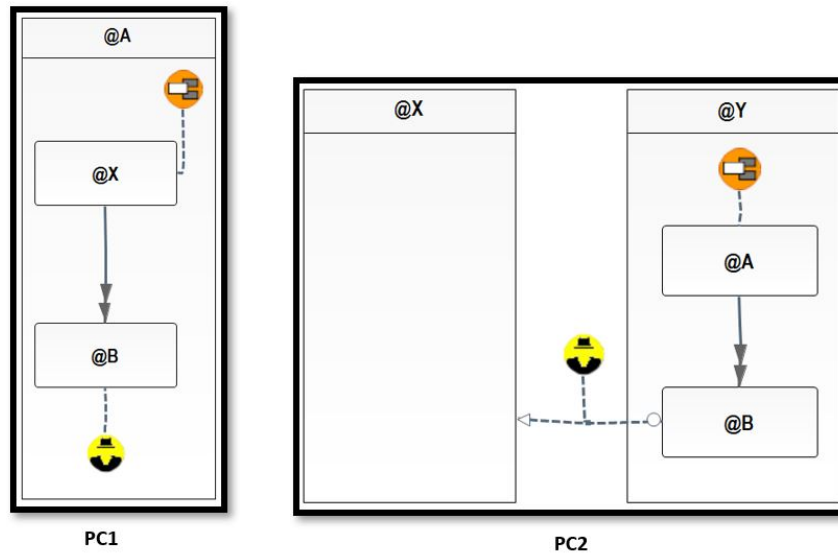
- **C4.1 Auditability vs Anonymity.** A conflict between an anonymity and an auditability requirement occurs if the business process model has: (1) an *Anonymity*- and *Auditability*-annotated task, (2) an *Anonymity*- and *Auditability*-annotated message flow, (3) an *Anonymity*-annotated task that reads data from an *Auditability*-annotated data object (because of different data directions, two anti-patterns are defined), (4) an *Anonymity*-annotated task that sends messages over an *Auditability*-annotated message flow or vice versa (because of two possible representations, two anti-patterns are defined), or (5) a task that reads data from an *Auditability*-annotated data object and sends messages over an *Anonymity*-annotated message flow (because of different data directions, two anti-patterns are defined). C1 and C2 in Figure 2.14 are two SecBPMN2 anti-patterns that represent the second and the fourth situation, respectively. C1 can be matched to one place in the model in Figure 2.11, as follows: the “Patient” pool is communicating with the “Emergency Unit” lane via an *auditability* and *anonymity*-annotated message flow. C2 can be matched to one place in the model in Figure 2.11, as follows: in the “Patient” pool the *anonymity*-annotated “Submit evaluation” task is sending messaging over an *auditability*-annotated message flow.
- **C4.2 Auditability vs Unobservability.** A conflict between an unobservability and an auditability requirement occurs if the business process model has: (1)

an *Unobservability*- and *Auditability*-annotated message flow, (2) an *Auditability*-annotated task sends messages over an *Unobservability*-annotated message flow, or (3) a task that reads data from an *Auditability*-annotated data object and sends messages over an *Unobservability*-annotated message flow (because of different data directions, two anti-patterns are defined).

- **PC4.1 Auditability vs Anonymity.** A potential conflict between an anonymity and an auditability requirement occurs if the business process model has: (1) a path between an *Anonymity*-annotated task and an *Auditability*-annotated task, (2) a path between an *Anonymity*-annotated task and a task that writes data to an *Auditability*-annotated data object (because of different data directions, two anti-patterns are defined), (3) a path between a task that sends messages over an *Auditability*-annotated message flow and a task sends messages over an *Anonymity*-annotated message flow, (5) a path between an *Anonymity*-annotated task and a task that sends messages over an *Auditability*-annotated message flow or vice versa (because of the different possible representations, two anti-patterns are defined), or (6) a path between a task that reads data from an *Auditability*-annotated data object and a task that sends messages over an *Anonymity*-annotated message flow (because of different data directions, two anti-patterns are defined).
- **PC4.2 Auditability vs Unobservability.** A potential conflict between an unobservability and an auditability requirement occurs if the business process model has: (1) a path between a task that sends messages over an *Auditability*-annotated message flow and a task that sends messages over an *Unobservability*-annotated message flow, (2) a path between an *Auditability*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow, or (3) a path between a task that reads data from an *Auditability*-annotated data object and a task that sends messages over an *Unobservability*-annotated message flow (because of different data directions, two anti-patterns are defined).

**Non-delegation.** *Non-delegation* in SecBPMN2 can be linked with an activity and it specifies that the activity shall be executed only by the users assigned. Non-delegation may conflict with anonymity and unobservability.

- **C5.1 Non-delegation vs Anonymity.** A conflict between these two requirements occurs if the business process model has: (1) an *Anonymity*- and *Non-delegation*-annotated task, or (2) a *Non-delegation*-annotated task that sends messages over an *Anonymity*-annotated message flow.
- **C5.2 Non-delegation vs Unobservability.** Since unobservability can only be linked with message flows, conflicts between unobservability and non-



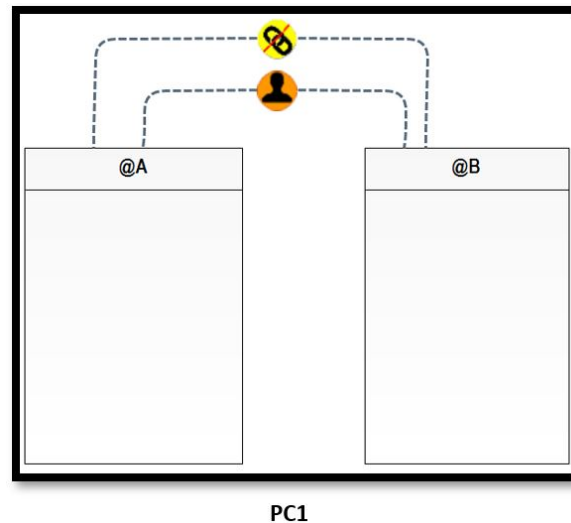
\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.15:** Potential conflicts PC1 and PC2 between *non-delegation* and *anonymity* as anti-patterns.

delegation requirements occur if the business process model has: (1) a *Non-delegation*-annotated task that sends messages over an *Anonymity*-annotated message flow.

- **PC5.1 Non-delegation vs Anonymity.** A potential conflict between an anonymity and a non-delegation requirement occurs if the business process model in question has: (1) a path between an *Anonymity*-annotated task and a *Non-delegation*-annotated task, or (2) a path between a *Non-delegation*-annotated task and a task that sends messages over an *Anonymity*-annotated message flow. PC1 and PC2 in Figure 2.15 are two SecBPMN2 anti-patterns that represent the first and the second situation, respectively. PC1 can be matched to one place in Figure 2.11, as follows: in the “Patient” pool there is a path between the *non-delegation*-annotated “Fill evaluation form” task and the *anonymity*-annotated “Submit evaluation” task. PC2 can be matched to one place in Figure 2.11, as follows: in the “Patient” pool there is a path between the *non-delegation*-annotated “Fill evaluation form” task and the “Submit evaluation” task, which sends messages over *anonymity*-annotated message flow.
- **PC5.2 Non-delegation vs Unobservability.** A potential conflict between an unobservability and a non-delegation requirement occurs if the business process model has: (1) a path between a *Non-delegation*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow.



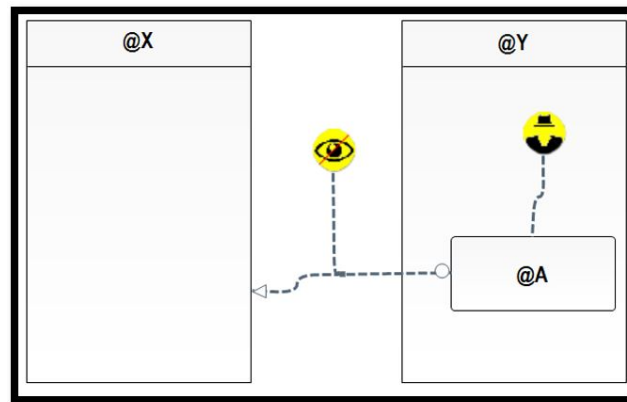


**Figure 2.16:** Potential conflict PC1 between *binding-of-duties* and *unlinkability* as anti-patterns.

**Binding of Duties.** *Binding of Duties* in SecBPMN2 can be linked with two pools and it specifies that the same person should be responsible for the completion of related activities. Binding of duties may conflict with anonymity and unlinkability.

- **PC6.1 Binding-of-Duties vs Anonymity.** A conflict between an anonymity and a binding-of-duties requirement occurs if the business process model has: (1) two pools that both are annotated with *Binding-of-Duties*, and each one of them includes an *Anonymity*-annotated task with the configuration {level=full anonymous, insider=true}.
- **PC6.1 Binding-Of-Duties vs Unlinkability.** A potential conflict between an unlinkability and a binding-of-duties requirement occurs if the business process model has: (1) a *Binding-Of-Duty*- and *Unlinkability*-annotated two pools, where the latter has the configuration {insider=true}. PC1 in Figure 2.16 is a SecBPMN2 anti-pattern that represents this situation. PC1 can be matched to one place in Figure 2.11, as follows: the “Patient” pool and the “Tele-medicine Device” pool are annotated with *binding-of-duties* and *unlinkability*.

**Anonymity.** *Anonymity* as specified in Section 2.4 can be specified differently based on the anonymity level (i.e., full anonymous vs. pseudonymous) and the type of adversaries considered (i.e., outsider+insider vs. only outsider adversaries). Thus anonymity requirements may conflict with other data minimization requirements.



C1 (variant 1 of 2)

\*\* The anonymity annotation is specified as follows:  $\{level=pseudonymous, insider=true\}$

**Figure 2.17:** Conflict C1 between *anonymity* and *unobservability* as anti-patterns.

- **C7.1 Anonymity vs Anonymity.** A conflict between two anonymity requirements occurs if the business process model has: (1) an *Anonymity*-annotated task that sends messages over an *Anonymity*-annotated message flow, the former with the configuration  $\{level=full\ anonymous\}$  and the latter with the configuration  $\{level=pseudonymous\}$ , or vice versa. (because of two possible representations, two anti-patterns are defined), or (2) an *Anonymity*-annotated task that sends messages over an *Anonymity*-annotated message flow, the former with the configuration  $\{insider=false\}$  and the latter with the configuration  $\{insider=true\}$ , or vice versa (because of two possible representations, two anti-patterns are defined).
- **C7.2 Anonymity vs Unobservability.** A conflict between an anonymity and an unobservability requirement occurs if the business process model has: (1) an *Anonymity*- and *Unobservability*-annotated message flow, the former with the configuration  $\{insider=false\}$ , or (2) an *Anonymity*-annotated task that sends messages over an *Unobservability*-annotated message flow, the former with the configuration  $\{level=pseudonymous\}$ . C1 in Figure 2.17 is a SecBPMN2 anti-pattern that represents the second situation. C1 can be matched to one place in Figure 2.11, as follows: in the “Tele-medicine Device” pool the “Send data to portal” task is annotated with anonymity and it requires that the  $\{level=pseudonymous\}$ . The “Send data to portal” task is also sending messages over *unobservability*-annotated message flow.
- **PC7.1 Anonymity vs Anonymity.** A potential conflict between two anonymity requirements occurs if the business process model has: (1) a path between two *Anonymity*-annotated tasks, the second with configuration  $\{level=pseudonymous\}$ , (2) a path between two tasks that send messages

over an *Anonymity*-annotated message flow, the first with the configuration  $\{level= pseudonymity\}$ , (3) a path between two tasks that send messages over an *Anonymity*-annotated message flow, the first with the configuration  $\{insider=false\}$ , (4) a path between an *Anonymity*-annotated task and a task that sends messages over an *Anonymity*-annotated message flow, the latter with the configuration  $\{insider=false\}$ , or (5) a path between an *Anonymity*-annotated task and a task that sends messages over an *Anonymity*-annotated message flow, the latter with the configuration  $\{level= pseudonymity\}$ .

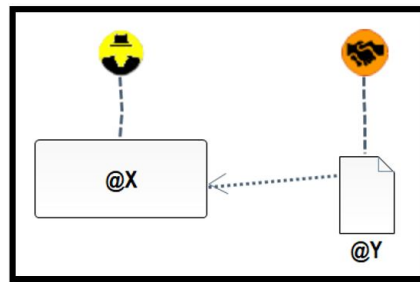
- **PC7.2 Anonymity vs Unobservability.** A potential conflict between these two requirements occurs if the business process model has: (1) a path between a task that sends messages over an *Anonymity* annotated message flow and a task that sends messages over an *Unobservability*-annotated message flow, the former with the configuration  $\{level = pseudonymity\}$ , or (2) a path between an *Anonymity*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow, the former with the configuration  $\{level=pseudonymity\}$ .

**Separation of Duties.** *Separation of Duties* in SecBPMN2 can be linked with two pools and it specifies that two or more distinct persons should be responsible for the completion of related activities. Separation of duties may conflict with anonymity.

- **C8.1 Separation-of-Duties vs Anonymity.** A conflict between an anonymity and a separation-of-duties requirement occurs if the business process model has: (1) two pools being annotated with *Separation-of-Duties*, and each one of them includes an *Anonymity*-annotated task with the configuration  $\{level=full anonymous, insider= true\}$ .

**Confidentiality.** *Confidentiality* in SecBPMN2 can be linked with: First, a data object to impose that only authorized users can read data from the data object. Second, a message flow to specify that only authorized users can receive and read the messages on that message flow.

- **PC9.1 Confidentiality vs Anonymity.** A potential conflict between an anonymity and a confidentiality requirement occurs if the business process model has: (1) an *Anonymity*-annotated task that reads data from a *Confidentiality*-annotated data object (because of the different possible representations, two anti-patterns are defined), (2) a path between an *Anonymity*-annotated task and a task that reads data from a *Confidentiality*-annotated data object (because of different data directions, two anti-patterns are defined), (3) a path between a task that reads data from a *Confidentiality*-annotated data object



PC1 (variant 1 of 2)

\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.18:** Potential conflict C1 between *confidentiality* and *anonymity* as anti-patterns.

and a task the sends messages over an *Anonymity*-annotated message flow (because of the different possible representations, two anti-patterns are defined), or (4) a task that reads data from a *Confidentiality*-annotated data object and sends messages over an *Anonymity*-annotated message flow (because of the different possible representations, two anti-patterns are defined). C1 in Figure 2.18 is a SecBPMN2 anti-pattern that represents the first situation. C1 can be matched to one place in Figure 2.11, as follows: in the “Patient” pool the *anonymity*-annotated “Submit evaluation” task is reading data from the confidentiality-annotated “Evaluation Form” data object.

- **PC9.2 Confidentiality vs Unobservability.** A potential conflict between an unobservability and a confidentiality requirement occurs if the business process model has: (1) a path between a task that reads data from a *Confidentiality*-annotated data object and a task that sends messages over an *Unobservability*-annotated message flow (because of the different possible representations, two anti-patterns are defined), or (2) a task that reads data from a *Confidentiality*-annotated data object and sends messages over an *Unobservability*-annotated message flow (because of the different possible representations, two anti-patterns are defined).

**Integrity.** *Integrity* in SecBPMN2 can be linked with: First, a data object to impose that only authorized users can read data from the data object. Second, a message flow to specify that only authorized users can receive and read the messages on that message flow. Third, a task to impose that only authorized users can do changes to the functionalities of the task.

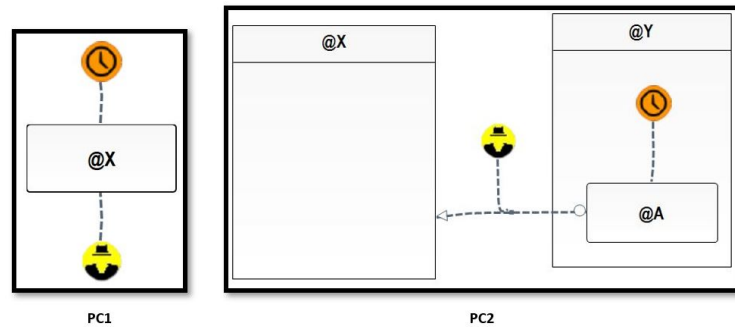
- **PC10.1 Integrity vs Anonymity.** A potential conflict between an anonymity and an integrity requirement occurs if the business process model has: (1)

an *Integrity*- and *Anonymity*-annotated task, (2) a path between an *Integrity*-annotated task and an *Anonymity*-annotated task, (3) an *Anonymity*-annotated task that reads data from an *Integrity*-annotated data object (because of the different possible representations, two anti-patterns are defined), (4) a path between an *Anonymity*-annotated task and a task that reads data from an *Integrity*-annotated data object (because of different data directions, two anti-patterns are defined), (5) a path between a task reads data from an *Integrity*-annotated data object and a task that sends messages over an *Anonymity*-annotated message flow (because of the different possible representations, two anti-patterns are defined), (6) a task that reads data from an *Integrity*-annotated data object and sends messages over an *Anonymity*-annotated message flow (because of the different possible representations, two anti-patterns are defined), or (7) an *Integrity*-annotated task that sends messages over an *Anonymity*-annotated message flow.

- **PC10.2 Integrity vs Unobservability.** A potential conflict between an unobservability and an integrity requirement occurs if the business process model has: (1) a path between an *Integrity*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow, (2) an *Integrity*-annotated task that sends messages over an *Unobservability*-annotated message flow, (3) a path between a task that reads data from an *Integrity*-annotated data object and a task the sends messages over an *Unobservability*-annotated message flow (because of the different possible representations, two anti-patterns are defined), or (4) a task that reads data from an *Integrity*-annotated data object and sends messages over an *Unobservability*-annotated message flow (because of the different possible representations, two anti-patterns are defined).

**Availability.** *Availability* in SecBPMN2 can be linked with: First, a data object to impose that the data object should be available when required by authorized users. Second, a message flow to specify that the transmission media that will be used to transfer messages should be available when requested by authorized users. Third, a task should be ready for execution by authorized users whenever the task is encountered in the control flow of the business process.

- **PC11.1 Availability vs Anonymity.** A potential conflict between an anonymity and an availability requirement occurs if the business process model has: (1) an *Availability*- and *Anonymity*-annotated task, (2) a path between an *Availability*-annotated task and an *Anonymity*-annotated task, (3) an *Anonymity*-annotated task that reads data from an *Availability*-annotated data object (because of the different possible representations, two anti-patterns are defined), (4) a path between an *Anonymity*-annotated task and a task that



\*\* All the anonymity annotations are specified as follows:  $\{level=full\ anonymous, insider=true\}$

**Figure 2.19:** Potential conflicts PC1 and PC2 between *availability* and *anonymity* as anti-patterns.

reads data from an *Availability*-annotated data object (because of different data directions, two anti-patterns are defined), (5) a path between a task reads data from an *Availability*-annotated data object and a task that sends messages over an *Anonymity*-annotated message flow (because of the different possible representations, two anti-patterns are defined), (6) a task that reads data from an *Availability*-annotated data object and sends messages over an *Anonymity*-annotated message flow (because of the different possible representations, two anti-patterns are defined), or (7) an *Availability*-annotated task that sends messages over an *Anonymity*-annotated message flow. PC1 and PC2 in Figure 2.19 are SecBPMN2 anti-patterns that represent the first and the seventh situations, respectively. PC1 can be matched to one place in Figure 2.11, as follows: in the “Patient” pool the “Submit evaluation” task is annotated with *availability* and *anonymity*. PC2 can be matched to one place in Figure 2.11, as follows: in the “Patient” pool the *availability*-annotated “Submit evaluation” task is sending messages over an *anonymity*-annotated message flow.

- **PC11.2 Availability vs Unobservability.** A potential conflict between an unobservability and an availability requirement occurs if the business process model has: (1) a path between an *Availability*-annotated task and a task that sends messages over an *Unobservability*-annotated message flow, (2) an *Availability*-annotated task that sends messages over an *Unobservability*-annotated message flow, (3) a path between a task that reads data from an *Availability*-annotated data object and a task the sends messages over an *Unobservability*-annotated message flow (because of the different possible representations, two anti-patterns are defined), or (4) a task that reads data from an *Availability*-annotated data object and sends messages over an *Unobservability*-annotated message flow (because of the different possible representations, two anti-patterns are defined).

**Fairness.** *Fairness* as specified in Section 2.4 can be linked with a task BPMN element. This annotation can be specified differently based on: First, the data that should not discriminate based on them. Second, the data that their effects on the output can be justifiable due to business needs. These two kinds of data can be specified using the *protect* and the *useExplanatory* attributes, respectively. Thus fairness requirements may conflict with other data-minimization requirements. In specific situations, two fairness requirements may also have a conflict with each other. However, since a conflict situation between a fairness and another requirement depends on how the *protect* and the *useExplanatory* attributes of the fairness annotation are specified. Since the specifications of these two attributes are domain-dependent, we define the conflicting situations as constrained conflicting situations such that if a match for the situation is found in an input model a corresponding constraint should be satisfied in order to report a conflict. Examples are already provided in Section 2.7.1.

- **C12.1 Fairness vs Undetectability.** In our work, a conflict between a fairness and an undetectability annotation can be reported:

if (the CC1 anti-pattern of Figure 2.10.a is matched in  $m$ ) and  
 $(fair.getProtected() \cup fair.getExplanatory()) \cap undetect.getProtected() \neq \emptyset$

Here,  $m$  is a SecBPMN2 model,  $fair.getExplanatory()$  and  $fair.getProtected()$  retrieve the set of data defined by  $fair$  as explanatory and protected data, respectively. The  $und.getProtected()$  retrieves the set of data specified as protected by  $und$ .

- **C12.2 Fairness vs Fairness.** In our work, a conflict between two fairness requirements can be reported:

if (the CC2 of Figure 2.10 is matched in  $m$ ) and  
 $(fair_1.getExplanatory() \cap fair_2.getProtected() \neq \emptyset)$

Here,  $fair_1.getExplanatory()$  and  $fair_2.getProtected()$  retrieve the set of data that are defined as explanatory data and protected characteristics, respectively.

- **PC12.1 Fairness vs Anonymity.** In our work, a potential conflict between a fairness annotation and an anonymity annotation can be reported:

if (the CPC1 anti-pattern of Figure 2.10.b is matched in  $m$ ) and  
 $(fair.getProtected() \cup fair.getExplanatory()) \cap anon.getProtected() \neq \emptyset$

Here,  $anon.getProtected()$  retrieves the set of data that are specified as protected by the matched anonymity annotation in  $m$ , while  $fair.getExplanatory()$  and  $fair.getProtected()$  retrieve the set of data that are defined by the matched fairness annotation in  $m$  as explanatory data and protected characteristics, respectively.

## 2.8 Tool Support

We developed a prototypical implementation of our work on top of STS-tool<sup>7</sup>, the supporting tool for SecBPMN2 [129]. STS-tool is a security requirement software tool that allows creating diagrams using the SecBPMN2 language, it supports automated analyses and enforcement of security constraints. An executable version of our implementation is available at <http://www.sts-tool.eu/downloads/secbpmn-dm>. Details on how to use our tool are provided in Appendix (A) of this thesis. Our implementation supports the contribution of our proposed BPMN-based framework for detection conflicts between data protection requirements:

- (i) modeling of security, data-minimization and fairness requirements in BPMN models, using a suitable model editor.
- (ii) modeling of security, data-minimization and fairness requirements as procedural patterns and anti-patterns.
- (iii) automatic alignment verification between security-, data-minimization- and fairness-procedural requirements specified as patterns and their specifications in the annotated BPMN models.
- (iv) automatic conflict detection between security, data-minimization and fairness requirements in annotated BPMN models with data-protection requirements, based on our catalog of domain-independent anti-patterns.

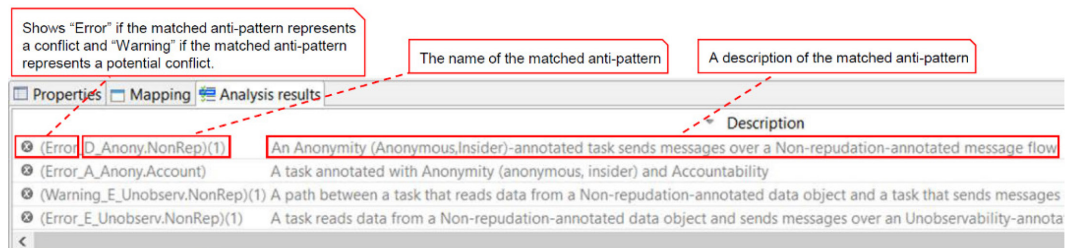


Figure 2.20: Analysis results view from our tool.

The examples shown in Figures. 2.2 and 2.7–2.10 come from screenshots of our implementation. Our alignment checking and conflict detection approaches require as an input a BPMN model with security, data-minimization and fairness annotations. The output of the alignment verification is a set of textual messages that describe the detected non-aligned requirements with their specifications in the model. The

<sup>7</sup>The STS-tool is available online at <http://www.sts-tool.eu> (accessed: 17/12/2019).



output of the conflict detection is a set of textual messages that describe the detected conflicts. Figure 2.20 shows a dedicated *view* from our tool, listing the messages. On demand, a non-aligned requirement or a conflict can be highlighted in the model. For example, the *red*-highlighted path in Figure 2.2 is the result of selecting the conflict message that describes the PC1 anti-pattern in Figure 2.9. Support for the constrained anti-patterns in Figure 2.10 is not yet implemented in the current version of our tool and it is part of future work.

In the following, we provide and describe the alignment checking and conflict detection algorithms<sup>8</sup>.

### 2.8.1 Algorithm for alignment checking

Algorithm 2.1 illustrates the alignment checking algorithm. The algorithm takes as input a SecBPMN2 model *model* annotated with the data protection annotations and a set of *patterns*. The patterns are specified by using the SecBPMN2-Q language. The algorithm returns an alignment report called *alignReport*. The report shows the result of the alignment verification as textual messages. On-demand, if a message shows that a SecBPMN2-Q is matched in the SecBPMN2 model, one can click on that message to highlight the matched pattern in the SecBPMN2 model. The following is a details description of the alignment checking algorithm.

First, as shown in line 1 of Algorithm 2.1, our algorithm is a function that takes as input a SecBPMN2 *model* and a set of SecBPMN2-Q *patterns*. The function starts by creating an empty report called *alignReport*. This report, as explained earlier, will be used to store the alignment verification results. Our algorithm iterates over all the SecBPMN2-Q patterns in the *patterns* set to search for matching in the SecBPMN2 *model*. Specifically, as shown in the lines 3-35 of Algorithm 2.1, for each SecBPMN2-Q pattern in *patterns*, the algorithm will perform the following:

- in line 4, an empty set called *bpmnPaths* will be created. This set will be used to store the BPMN paths in the SecBPMN2 *model* that match the specified BPMN parts in the SecBPMN2-Q *pattern*.
- in line 4, a function called *findBPMNPath* will be invoked. This function takes as input the SecBPMN2 *model* and the SecBPMN2-Q *pattern*. This function searches for BPMN paths in the SecBPMN2 *model* that match the speci-

---

<sup>8</sup>We do not share the implementation of the algorithms because it depends on several plugins of the STS tool, which is not an open-source tool. However, an executable version of the algorithms is available online at <http://www.sts-tool.eu/downloads/secbpmn-dm> (accessed: 17/12/2019)

**Algorithm 2.1:** Alignment checking

---

```

1 checkAlignment (model, patternsSet);
2 create alignmentReport;
3 foreach pattern ∈ patternsSet do
4   bpmnPaths ← ∅, bpmnPaths ← findBPMNPath(model, pattern);
5   if bpmnPaths ≠ emptyset then
6     foreach path ∈ bpmnPaths do
7       notMatchedSpecifications ← ∅;
8       annotPatternSet ← ∅, annotPathSet ← ∅;
9       annotPatternSet ← getAnnotations(pattern);
10      annotPathSet ← getAnnotations(path);
11      if satisfies(annotPatternSet, annotPathSet) then
12        foreach annotPattern ∈ annotPatternSet do
13          foreach annotPath ∈ annotPathSet do
14            if annotPattern.type == annotPath.type then
15              if checkTarget(annotPattern, annotPath) then
16                notMatchedSpecifications ←
17                  checkSpec(annotPattern, annotPath);
18              end
19            end
20          end
21        if notMatchedSpecifications == ∅ then
22          linkText = "a full match for the "+pattern.getName()+"
23            and its specification is found in the SecBPMN2 model";
24          alignReport.add(path, linkText);
25        end
26      else
27        linkText = "Error: a match for the BPMN parts of the
28          "+pattern.getName()+" is found but not for its annotations;
29        alignReport.add(path, linkText);
30      end
31    end
32  else
33    alignReport.add("Error, our algorithm can not find a match for the
34      "+pattern.getName()+" in the model");
35  end
36 return alignReport

```

---

fied BPMN parts in the SecBPMN2-Q *pattern* regardless of the specified data protection annotations in the SecBPMN2-Q *pattern*.

- if the *findBPMNPath* returns an empty set *bpmnPaths*, this means that there is no BPMN path in the SecBPMN2 *model* that matches the specified BPMN parts in the SecBPMN2-Q *pattern*. Therefore, as shown in line 31, an "Error" message will be added to the *alignReport*. The error message informs the user that the SecBPMN2-Q *pattern* in question does not have a match in the SecBPMN2 *model*.
- if the set *bpmnPaths* that is not empty, the *satisfies* function will be invoked. This *satisfies* function returns *true* if: (1) the data protection annotations of the matched BPMN path in the SecBPMN2 *model* are of the same type of those in the SecBPMN2-Q *pattern*, and (2) they are linked to the same BPMN elements. If the *satisfies* function returned *true*, then the algorithm iterates over the matched annotations to check if the specifications of their attributes are also matched. For this, as shown in line 16, a function called *checkSpec* will be invoked. The *checkSpec* return the attributes of the annotations in the SecBPMN2-Q *pattern* that are not correctly specified in the matched data protected annotations of the SecBPMN2 *model*.
- For each BPMN *path*, the algorithm checks if the *notMatchedSpecification* is empty; see line 21. If this is the case this means that: the specifications of the data protection annotations in the SecBPMN2-Q *pattern* match the specifications of the data protection annotations in the BPMN *path*. As a result, in line 23, a message will be added to the *alignReport* to inform the user that a full match for the SecBPMN2-Q *pattern* is found in the SecBPMN2 *model*. The matched BPMN *path* is also stored *alignReport*. This to allow the user to highlight the matched SecBPMN2-Q *pattern* in the SecBPMN2 *model*.

## 2.8.2 Algorithm for conflict detection

Algorithm 2.2 illustrates the conflict detection algorithm<sup>9</sup>. The algorithm takes as input a SecBPMN2 *model* annotated with the data protection annotations and a catalog of conflicts called *antiPatternsCatalog*. The conflicts in our catalog are specified as SecBPMN2-Q anti-patterns. The SecBPMN2-Q anti-patterns in the *antiPatternsCatalog* are of two types: *conflict* and *potential conflict*. The algorithm returns a report called *conflictReport*. This report shows the result of the conflict

<sup>9</sup>Algorithm 2.2 applies for all of our proposed anti-patterns except for the constrained anti-patterns that represent conflicts between fairness and data-minimization annotations (see Figure 2.10). In these cases, a match for the anti-pattern is not enough to report conflicts. In addition to the matching, other constraints should be satisfied in order to report conflict. The constraints are already explained through examples in Part (C) of Section 2.7.1

detection as textual messages. The message shows "Error" if a conflict anti-pattern matched in the SecBPMN2 *model* and "Warning" if the matched anti-pattern represents potential conflict. On-demand, the user can click on the message to highlight the matched (potential) conflict in the SecBPMN2 *model*.

The conflict detection algorithm is quite similar to Algorithm 2.1 of the alignment checking. However, Algorithm 2.1 searches for matches of SecBPMN2-Q *patterns* in a given SecBPMN2 model while Algorithm 2.2 searches for matches of SecBPMN2-Q *anti-patterns* in a given SecBPMN2 model. In the following, we provide a details description of the conflict detection algorithm.

First, as shown in line 1 of Algorithm 2.2, our conflict detection algorithm is a function that takes as input a SecBPMN2 *model* and a catalog of conflict called *antiPatternsCatalog*. As mentioned earlier, the conflicts in our catalog are specified as SecBPMN2-Q anti-patterns. As shown in line 2 of Algorithm 2.2, the *detectConflict* function starts by creating an empty report called *conflictReport*. This report will be used to store the detected conflicts and potential conflicts. Our algorithm iterates over all the SecBPMN2-Q anti-patterns in the *antiPatternsCatalog* to search for matching in the SecBPMN2 *model*, see lines 3-27 of Algorithm 2.2.

Specifically, as shown in the lines 3-27 of Algorithm 2.2, for each SecBPMN2-Q *antiPattern* in *antiPatternsCatalog*, the algorithm will perform the following:

- in line 4, an empty set called *bpmnPaths* will be created. This set will be used to store the BPMN paths in the SecBPMN2 *model* that match the specified BPMN parts in the SecBPMN2-Q *antiPattern*.
- in line 4, a function called *findBPMNPath* will be invoked. This function takes as input the SecBPMN2 *model* and the SecBPMN2-Q *antiPattern*. This function searches for BPMN paths in the SecBPMN2 *model* that match the specified BPMN parts in the SecBPMN2-Q *antiPattern* regardless of the specified data protection annotations in the SecBPMN2-Q *antiPattern*.
- if the set *bpmnPaths* that is returned by *findBPMNPath* is not empty, the lines from 7-11 will be executed. In line 9, the data protection annotations of the SecBPMN2-Q *antiPattern* will be retrieved and stored in the *annotAntiPatternSet*. In line 10, the data protection annotations of the BPMN *path* will be retrieved and stored in the *annotPathSet*. In line 20, the *satisfies* will be invoked. This *satisfies* function returns *true* if: (1) the data protection annotations of the matched BPMN path in the SecBPMN2 *model* are of the same type of those in the SecBPMN2-Q *antiPattern*, and (2) they are linked to the same BPMN elements.

**Algorithm 2.2:** Conflict Detection

---

```

1 detectConflict (model, antiPatternsCatalog);
2 create conflictReport;
3 foreach antiPattern ∈ antiPatternsCatalog do
4   bpmnPaths ← ∅, bpmnPaths ← findBPMNPath(m, antiPattern);
5   if bpmnPaths ≠ emptyset then
6     foreach path ∈ bpmnPaths do
7       matchedSpecifications ← false;
8       annotAntiPatternSet ← ∅, annotPathSet ← ∅;
9       annotAntiPatternSet ← getAnnotations(antiPattern);
10      annotPathSet ← getAnnotations(path);
11      if satisfies(annotPatternSet, annotPathSet) then
12        foreach annotAntiPattern ∈ annotAntiPatternSet do
13          foreach annotPath ∈ annotPathSet do
14            if annotAntiPattern.type == annotPath.type then
15              if checkTarget(annotAntiPattern, annotPath)
16                then
17                  matchedSpecifications ←
18                    verifySpecific(annotAntiPattern, annotPath);
19                end
20              end
21            end
22          end
23          if matchedSpecifications == true then
24            if antiPattern.type == conflict then
25              linkText = "Error:" + antiPattern.getName();
26              conflictReport.add(Path, linkText);
27            end
28            else
29              linkText = "Warning:" + antiPattern.getName();
30              conflictReport.add(Path, linkText);
31            end
32          end
33        end
34      end
35    end
36    if conflictReport is empty then
37      conflictReport.add("None of the anti-patterns in the conflicts catalog
38        has a match in the SecBPMN2 model");
39    end
40  return conflictReport

```

---

- If the *satisfies* function returned *true*, then the algorithm iterates over the matched annotations to check if the specifications of their attributes are also matched. For this, a function called *verifySpecific* will be invoked. This function does not check the specifications of all the attributes of the data protection annotations. The *verifySpecific* looks for the specified attributes in the data protection annotations of the SecBPMN2 *antiPattern*. The *verifySpecific* returns true if the specified attributes in the SecBPMN2 *antiPattern* are matched in the BPMN *path*. The returned values will be stored in a Boolean flag called *matchedSpecifics*.
- in line 21, the algorithm checks whether the *matchedSpecifics* is true. If this is the case this means that the specifications of the data protection annotations in the SecBPMN2-Q *antiPattern* are also matched in the data protection annotations of the BPMN *path*. As a result, in line 22, the algorithm checks whether the matched SecBPMN2-Q *antiPattern* represents conflict. If this is the case an "Error" message together with the matched BPMN *path* will be added to the *conflictReport*. Otherwise, a "Warning" message together with the matched BPMN *path* will be added to the *conflictReport*. On-demand, the user can click on the message to highlight the matched SecBPMN2-Q *antiPattern* in the SecBPMN2 *model*.
- in line 29, the algorithm checks if the *conflictReport* is empty. If this is the case, this means that none of the SecBPMN2-Q anti-patterns in our catalog is matched in the SecBPMN2 *model*.

## 2.9 Case Study

To study the feasibility of our approach, we applied it in a healthcare scenario. We extended a teleconsultations healthcare management case study from the Ospedale Pediatrico Bambino Gesù (OPBG), a pediatric Italian hospital. The case study was part of the *VisiOn* research project<sup>10</sup>. The main objective of *VisiOn* consisted of increasing the citizens' awareness of data protection. The final outcome of the project was a platform that can be used by public administrations and companies to design their systems, using security and privacy as first-class requirements. The teleconsultations case study described a situation where a patient health care record can be transferred from the OPBG system to specialists in another hospital for teleconsultations purposes. In this scenario, many security requirements are considered (e.g., confidentiality, accountability) but the privacy preferences were more related to data anonymization. In this chapter, we extended this scenario to cover situations where data minimization and fairness play an important role in protecting the

---

<sup>10</sup><http://www.visioneuproject.eu/> (accessed: 08/12/2019)

users' data. To this end, we modeled a process featuring an over distance healthcare service<sup>11</sup>, an excerpt being shown in Figure 2.2. Using our approach, as explained in Section 2.4, we were able to enrich the model with data-minimization and fairness requirements that represent data-protection preferences for patients.

For conflict detection, we annotated the model with security requirements that represent security needs from the system point of view. Assessing the accuracy of conflict detection based on this model required a ground truth. To this end, we manually analyzed the model and identified 9 conflicts and 21 potential conflicts, a subset being discussed in Section 2.7. Applied to the model, our conflict detection technique detected 8 out of the 9 manually detected conflicts. This is because one of the manually detected conflicts can be detected with a constrained anti-pattern which is not yet implemented. Our tool detects 21 potential conflicts as expected.

## 2.10 User Experiment

We further aimed to study the practical usefulness of our technique for conflict detection between security, data-minimization and fairness requirements. We focused on two main goals: first, to study if our initial assumption that the manual detection of conflicts is error-prone can be confirmed. Second, to study if users find the output of our technique helpful for supporting the conflict detection.

Based on these goals, we derived the following research questions:

- RQ1: How error-prone is conflict detection when performed manually?
- RQ2: How helpful do users find the output of the proposed automated conflict detection approach?

The research questions (i.e, RQ1 and RQ2) were studied in a user experiment with 30 new participants, a significantly larger sample than in our earlier preliminary study, which was based on 7 participants [108] and did not address RQ1, as it was based on a subject assessment only.

---

<sup>11</sup>The model with all security, data-minimization, and fairness requirements is publicly available at <https://github.com/QRamadan/MoPrivFair-ConflictsDetection/blob/master/README.md> (accessed: 31/12/2019).

### 2.10.1 Set-up of the experiment

Our overall sample consists of 30 participants, specifically, 18 master students, 1 bachelor student, 8 researchers, and 3 practitioners. Thus, we relied primarily on students as participants. We justify this choice with earlier research findings, according to which students perform nearly equally well when faced with a novel software development approach, and are therefore suitable as stand-ins for practitioners [125]. As a baseline, our technique assumes the user to be familiar with BPMN. Therefore, the students were recruited from a course in which BPMN was introduced and students completed relevant assignments two weeks before the experiment. BPMN-experienced researchers and practitioners were recruited based on the personal network of the author (convenience sampling).

**Table 2.5:** Experience levels of participants in our experiment.

	Mean	Median	St.dev
<b>BPMN</b>	2.55	2	1.27
<b>Security</b>	3.14	3	1.06
<b>Privacy</b>	2.97	3	0.99

The participants were asked to self-assess their expertise in the relevant background fields BPMN, security, and privacy on a 5-point likert scale. For each background field, Table 2.5 reports the mean and average, response values and the standard deviation. With expertise levels of 2.55/5 for BPMN, 3.14/5 for security, and 2.97/5 in privacy (means of response values), this distribution approximates the background of the intended user group, characterized by relevant knowledge, but absence of expert knowledge in business process modeling, security, and privacy.

We presented the participants with a questionnaire and an auxiliary "cheatsheet" with explanations and specifications of our notations<sup>12</sup>. The questionnaire consisted of four tasks in total. Tasks 1 and 2 were warm-up tasks for making the participants familiar with the security and data-minimization annotations from our approach. Tasks 3 and 4, elaborated below, focused on conflict detection. The tasks were based on the running example of this chapter as introduced in Section 2.4. We showed excerpts from the contained BPMN model and security, data-minimization, and fairness requirements. All the participants worked on the same tasks. All the tasks share the same BPMN model but in each task the number of security, data-minimization and fairness annotations in the BPMN model was different. Depending on the availability of a laptop, participants filled out the questionnaire on their computers or on a paper printout.

---

<sup>12</sup>For transparency we share all our experimental materials and raw data online at <https://github.com/QRamadan/MoPrivFair-ConflictsDetection/blob/master/README.md> (accessed: 31/12/2019).



Tasks 3 and 4 represent the experimental tasks. We used a variant of the within-subject experimental design [23], in which all participants act as their own control: Every participant was exposed to two treatments, conflict detection without tool support (RQ1) and with tool support (RQ2). In task 3, we studied RQ1 by having the participants manually identify conflicts and potential conflicts as detected by our technique. To this end, we showed them an annotated BPMN model with annotations of various types and asked them to identify conflicts and potential conflicts. As a first question, we asked them to identify conflicts manually and, in the positive case, write down the conflict. As a second question, we presented them with a list of types of potential conflicts and asked them in a multiple-choice question which type actually occurs.

Afterwards in Task 4, to study RQ2, we presented them with the output of our technique when applied to the examples. Specifically, we showed them screenshots of the user interface of our tool. Sources of conflicts are highlighted in the diagram, and a textual explanation of the identified conflict is given. We asked the participants to rate the helpfulness of this presentation on a 5-point Likert scale, 1 indicating "not helpful" and 5 indicating "very helpful". We also asked them for free-form feedback regarding the overall technique.

### 2.10.2 Results of the experiment

Regarding RQ1, Figure 2.21 shows the results of the task regarding manual conflict detection. Only a minority of 17% and 34% of all participants gave the correct answer in the conflict and potential-conflict cases, respectively. 72% and 28% of all participants, respectively, gave an incorrect answer in the conflict and potential-conflict cases: they stated that the model does not contain any conflicts, or selected the wrong conflicts types. A portion of 10% or 38% chose not to answer the question, indicating a varying difficulty level between both questions.

The weak performance of our participants in identifying conflicts manually confirms our initial assumption that this is an error-prone task.

Regarding RQ2, Figure 2.22 shows the results of the subjective assessment of the helpfulness of our conflict detection output. Overall, in both cases, we found a positive sentiment. Majorities of 54% and 68% gave a helpfulness score above the medium of the scale: A score of 5 was chosen by 39% in the conflict case and 25% in the potential-conflict case. A score of 4 was chosen by 29% in both cases. Scores below the medium were in a clear minority, 3.5% each for a score of 1 or 2 for conflicts and potential conflicts. The positive tendency is in line with the comments we received in the free-form feedback, such as: *"I liked the implementation of detecting*

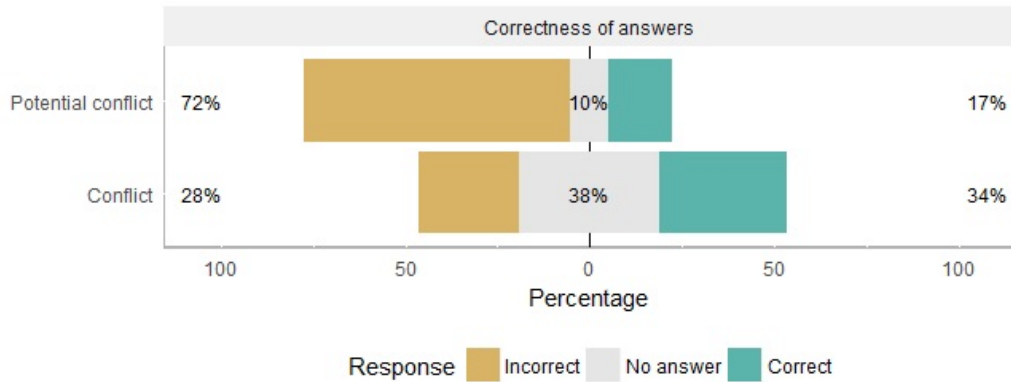


Figure 2.21: Results for RQ1: Error-proneness of manual conflict detection

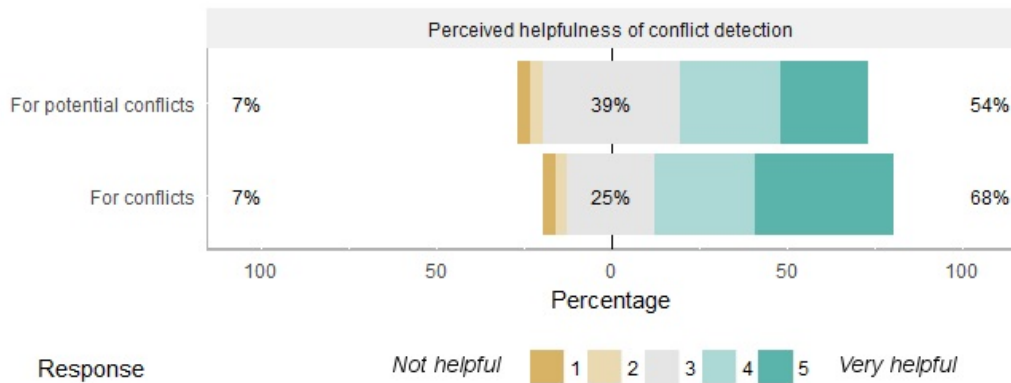


Figure 2.22: Results for RQ2: Perceived helpfulness of automated conflict detection

the conflicts in figure E” and “The notation is really helpful. In my opinion, it can facilitate the process of designing critical systems with sensitive information.”

We also analyzed the written free-form feedback for negative comments which may explain the comparatively large share of mid-ranged answers (39% for potential conflicts and 25% for conflicts, respectively). While we did not find any related comments on specific aspects of the output, one participant pointed out the need for careful instructions: “It is very hard to give meaningful feedback without a clear explanation of the proposed notation and the respective semantics.” We do not find that this comment is in line with the feedback from the majority of our participants, who did not comment negatively on the provided “cheatsheet” and its descriptions of the annotation and its semantics. Provided with the cheatsheet, a third of all participants were able to specify correct answers during manual conflict detection, indicating its general understandability. Still, this comment is significant, as it emphasizes the need for adequate training when applying our technique in real-life scenarios.

The two main observations from our study can be summarized as follows:

- First, our technique supports the detection of errors that are hard to identify manually (RQ1).
- Second, users generally perceive the output of our technique as helpful (RQ2).

### 2.10.3 Threats to validity

Our experimental design is subject to a number of threats to validity, as discussed below. The three main types of validity threats to our study are *external*, *internal*, and *construct* validity.

**A. External validity.** Our experimental design is subject to various threats to external validity: the obtained results may not generalize to other cases. There are two threats to external validity: First, a threat concerns the expressiveness of our language for catering to a wide variety of scenarios with their distinct requirements: Our experimental material was based on one particular case study. Second, we relied to a large extent on students as participants in the study. We provided a rationale for this decision in Section 2.10.1. Despite these limitations, our results can be generalized to case studies from other domains due to the following reasons: First, based on the results of previous study [125], students perform nearly equally well when faced with a novel software development approach, and are therefore suitable as stand-ins for practitioners. Second, the considered security, data-minimization, and fairness requirements in our healthcare case study are applicable to other system from different domains. Consider, for example, a conference management system such as the easy-chair. In such systems, various security and data-minimization requirements can be considered.

**B. Internal validity.** A threat to internal validity are experimenter expectations. Subjective assessments, such as those regarding the usefulness, may be affected by apparent expectations. Specifically, in RQ2, participants might have assessed our technique positively without having understood its output.

We point out that the participants were presented with detailed insights as provided by our technique, including a highlighting of the offending model elements and textual error messages such as: *"Error: The process requires anonymous execution for a task that writes data to a secure storage, where the executor should not be able to deny that she modified data."* We argue that this kind of feedback is key for establishing understandability. In line with this rationale and related positive textual feedback, the majority of participants assessed the helpfulness of our technique positively. The lower average score for the case of potential conflicts may come from this case being harder to understand.

**Table 2.6:** Execution time of conflict detection technique.

SecBPMN Model	Number of BPMN elements	Number of Annotations	Time in Seconds
Model 1	21	1	24 sec
Model 2	54	7	45 sec
Model 3	92	17	82 sec

**C. Construct validity.** The validity of our experiments may be disputed due to construct validity. When studying the practical applicability of our approach, we only considered the output of the conflict detection technique, rather than the full process, including the assumption that a user manually annotates the involved BPMN model. Even though annotating the models is an involved task, the resulting annotations arguably provide several benefits for communication and getting an intuitive overview of the contained requirements. This task can be made easier by providing appropriate tool support, as shown in earlier work of SecBPMN2 [130].

## 2.11 Limitations and Future Work

A limitation is that our proposed notation is currently not equipped with a formal semantics. We provided structured textual descriptions of all considered data protection requirements. The textual descriptions incorporate feedback from several rounds of revisions, and from the participants of an earlier experiment [111]. However, generally speaking, we cannot exclude the existence of cases that require additional formal investigation. While doing so is outside the scope of this work, we consider it a possible direction for follow-up work.

Regarding the performance of the conflict detection technique, the performance bottleneck is the use of SecBPMN-Q’s query engine. Previous evaluation results for the engine show that execution time grows linearly with the number of activities and exponentially with the number of processes [131]. To still offer first insights into the scalability of our technique, we performed a preliminary assessment based on our running example<sup>13</sup>, showing the results in Table 2.6.

Models 1 and 2 in Table 2.6 are smaller versions of the complete model (a.k.a. model 3), which we produced by removing connected parts. The tests were performed on a computer with a 2.2 Ghz processor and 8 GB of memory. Taking around one minute for the largest model and not showing an exponential slowdown, the performance of the proposed conflict detection technique seems adequate for practical use on models of the considered size.

---

<sup>13</sup>Our example model is available online at <https://github.com/QRamadan/MoPrivFair-ConflictsDetection/blob/master/README.md> (accessed: 17/12/2019)

Moreover, as in any practical software system, we cannot rule out the possibility of implementation defects in the implementation of our conflict detection technique. Our technique is based on declarative conflict patterns, which are automatically evaluated by a query engine. This pattern-based approach aims to reduce cognitive complexity during implementation, and thus may be less error-prone than hard-coding the technique in a general-purpose language. Yet, giving proof of this intuition requires additional work.

**We outline four important directions for future work:** First, while we made a systematic effort to assure completeness and correctness, we did not provide a formal validation. To address this gap, we suggest relying on algebraic graph transformations in the following way:

- (i) Specify the semantics of each data-protection requirement using one or several graph transformation rules. The rules would describe forbidden and intended flows of information between different actors.
- (ii) Apply a formally based conflict detection technique [76, 77], which can discover all possible conflicts arising from a set of rules.

Other possibilities for future work are:

- First, to extend our approach to support the resolution of conflicts. Although a fully automated process would be appreciated, the resolution of conflicts may require human intervention [42, 57], a further challenging task that involves reasoning on the privacy impact of different solution strategies [6, 87].
- Second, to refine our technique to consider the specified enforcement technologies during conflict detection. Similar to the work in [106], our approach allows specifying enforcement technologies for data protection annotations. Consider the specified enforcement technologies during conflict detection will reduce the number of reported potential conflicts by our technique. This requires as input a database with up-to-date information about enforcement technologies and their abilities to preserve data protection requirements.
- Third, to automate the tasks of maintaining our data protection patterns. Specifically, a possible future work is to provide an automated approach for: (1) mapping textual requirements to existing domain-specific patterns; (2) creating patterns for those data protection requirements that do not have corresponding patterns. Automating these two tasks can benefit from existing research work (e.g., [127]), and will allow avoiding badly modeled and duplicated patterns in the pattern repository.

## 2.12 Related Work

Having given an overview of existing BPMN security-oriented extensions and their limitations in Section 2.2.5, we now provide a comprehensive discussion of other related work in the field of security and privacy engineering.

### 2.12.1 Conflicts between data protection requirements

To our knowledge, no existing approach supports conflict detection between security, data-minimization and fairness requirements. For example, Hansen et al. [54] define six privacy and security goals for supporting the privacy needs of users. The authors considered a subset of the data-minimization concepts in [103], namely *anonymity* and *unlinkability*, and discuss their relationships. However, conflicts are discussed at the conceptual level, abstracting from concrete systems, while we show that the specific conflicts arising in a system can be identified by analyzing the data-system's minimization and security requirements.

The perspective papers of Ganji et al. [50] and Alkubaisy [10] highlight the importance of detecting conflicts between security and privacy requirements, specifically for data-minimization requirements. Both papers discuss the components required for a potential approach, however, without providing a complete solution. Ganji et al. [50] envision a realization for an automated conflict detection approach based on the SecureTropos framework as future work. However, to our knowledge, an automated approach that realizes the research work in [50] is still missing.

### 2.12.2 Model-based conflict detection approaches

We are not the first to use models for detecting conflicts between data protection properties. Paja et al. [99] propose an extension of Socio-Technical Systems (STS), a security goal-oriented approach, to allow checking if the stakeholders have conflicting authorization requirements such as who allowed to read, delete, modify and transfer data and for which purpose. Different from our approach, this approach does not support conflict detection between security, data-minimization and fairness requirements. However, our approach can be seen as complementary to this one due to the following reasons: First, the prototypical implementation of our work is developed on top of the STS environment, which supports the BPMN extension SecBPMN2 [129]. Second, a technique that supports the transformation from STS models to SecBPMN2 models is available [127].

Elahi et al. [42] propose an extension to a goal-oriented modeling language called *i\** to model and analyze security trade-offs among competing requirements of multiple users. The main goal is to allow identifying an optimal set of security mechanisms that together can achieve a good-enough security level without violating usability issues. However, this research work does not consider conflicts analysis between security, data-minimization and fairness requirements. Moreover, the authors considered the design of catalog reusable knowledge about security trade-offs for automating the trade-offs analysis as part of future work.

Saadatmand et al. [123] propose an approach that automatically, based on fuzzy logic extension of the TOPSIS decision-making method, analyzes UML class models annotated with non-functional requirements in order to evaluate different design alternatives and identify which one leads to the better overall satisfaction of non-functional requirements. For the same purpose, Pasquale et al. [100] propose to use the KAOS goal-oriented approach to study interactions between security requirements such as confidentiality and other organizational and non-functional requirements such as cost budget and performance, respectively. The proposed approach uses a Satisfiability Modulo Theories (SMT) solver to interpret the KAOS models and automate the execution of the trade-off analyses. This helps the security engineer in selecting security mechanisms and configurations that can maximize security without violating other goals such as usability. Different from our work, the works do not support conflict analysis between data protection related requirements. Instead, they support trade-offs analysis between security requirements and other quality or organizational related requirements such as the usability and the cost budget, respectively. Furthermore, they do not consider the context of how the requirements interact with each other. Models are used to support conceptual modeling for security and quality concepts and relationships between them, without considering the underlying behavior of the target system.

### 2.12.3 Other conflict detection approaches

The literature is rich with approaches that rely on textually specified requirements to identify conflicts between them. A comprehensive overview of these approaches can be found in [9]. In what follows, we discuss the approaches that consider security or privacy as part of their supported concepts.

Like we do, Egyed et al. [39] argue that detecting conflicts between the requirements requires an understanding of their dependencies. To overcome this challenge, the authors propose an approach that takes as input a set of textual non-functional requirements. The approach then automatically finds out the requirements that have trace dependencies between each other. The trace dependencies

are created among requirements if their test scenarios execute the same or similar lines of code. In a final step, the approach uses a matrix of conflicts to decide whether two dependent requirements are conflicting or not. Mairiza et al. [84] propose a catalog of conflicts between non-functional concepts. The catalog is a two-dimensional matrix that shows conflicts between the non-functional concepts at the conceptual level without considering their context. The same authors have proposed an ontology-based framework that aims to manage the relative conflicts among non-functional requirements, particularly those between security and usability requirements [83]. However, the work does not provide technical details about how the framework can be automated. Technical support and framework evaluation are proposed as future work.

Poort et al. [105] propose a framework called Non-Functional Decomposition, which aims to detect conflicts between non-functional requirements and suggest strategies to resolve them. For conflict detection, the framework suggests to group the requirements based on their functionalities and then study trade-offs between the non-functional requirements that belong to the same functional requirement. However, this framework provides only a methodological solution for conflict detection at a high-level of abstraction without supporting a concrete solution for how trade-offs between the requirements can be automatically detected.

Different from our work, the works in [39, 83, 84, 105] do not support conflict detection between data protection concepts. They aim to detect conflicts between non-functional requirements such as security, usability, efficiency, and maintainability. Surprisingly, although the proposed matrix in [84] consider the privacy and the security as two non-functional concepts, it does not show that privacy and security can be (potentially) in conflict. Moreover, these approaches rely on informally textually specified requirements to detect conflicts between them, which may lead to inaccurate results due to inexact semantics.

#### **2.12.4 Data-minimization-aware approaches**

Various works in data-protection requirements engineering aim to specify privacy requirements using the data-minimization concepts proposed in [103]. In Deng et al.'s LINDDUN framework [32], both misuse cases and data-minimization requirements can be identified by mapping predefined threat-tree patterns to the elements of a data-flow diagram. Kalloniatis et al. [65] propose the Pris methodology, which maps data-minimization and other security concepts to a system's organizational goals to identify privacy requirements. Pris introduces privacy-process patterns that describe the effect of privacy requirements to organizational processes.



Mouratidis et al. [95] present a conceptual framework that combines security and data-minimization concepts, and show its use to specify details about privacy goals such as the involved actors and threats. Beckers et al. [16] propose a privacy-threats analysis approach called ProPAn that uses functional requirements modeled in the problem-frame approach to check if insiders can gain information about specific stakeholders. Diamantopoulou et al. [34] provide a set of privacy process patterns for security and data-minimization concepts, aiming to provide predefined solutions for different types of privacy concerns in the implementation phase. In addition to textual description of the patterns, BPMN design patterns were provided to guide operationalization at the business process level.

Since none of these approaches considers conflicts between data-protection requirements, our approach can be seen as complementary: Their output can be used as input for our approach to allow the enrichment of the business process models with data protection requirements and then to perform conflict detection.

### 2.12.5 Fairness-aware approaches

Fairness is currently dealt with as an algorithmic problem at the implementation phase. Approaches in this field can be classified into *detection* and *prevention* approaches. Approaches of the former type aim to test whether a given software discriminates against protected characteristics. Approaches of this type can be further classified into while-box (e.g., [7]) and black-box (e.g., [29, 52, 145]) approaches.

The trade-offs between fairness and privacy needs are recently considered as a challenge in the field of algorithmic fairness [29]. However, so far there exists no approach that allows modeling fairness requirements and detecting conflicts between them and other data protection requirements as early as during the design phase of the targeted system. The need for integrating fairness in the early design stages has been highlighted by our position papers in [109, 112].

## 2.13 Conclusion

We proposed an BPMN-based framework for supporting the detection of conflicts between security, data-minimization, and fairness requirements. To this end, we first proposed an extension of BPMN that permits the specification of these requirements in business process models, based on existing security annotations from the SecBPMN2 [129] and new data-minimization and fairness annotations.

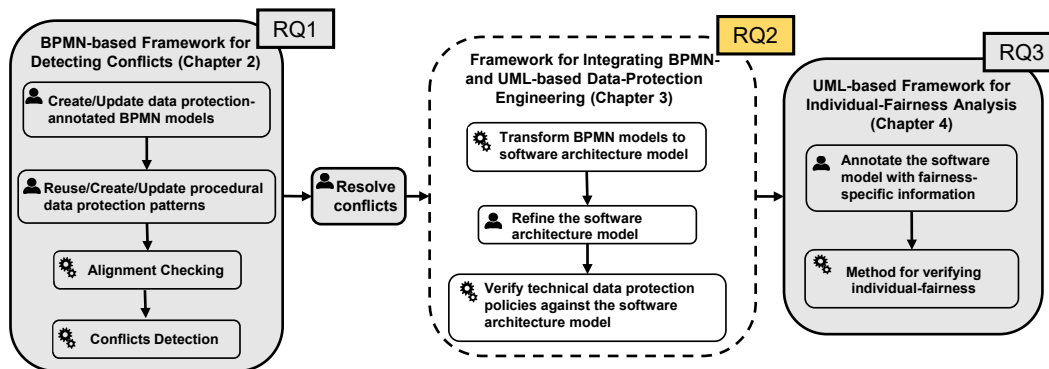
Based on this extension, first, we enabled checking the alignment between the security, data-minimization and fairness requirements and their specifications in the BPMN models. We automated this process by extending a graphical query language called SecBPMN2-Q to allow formulating the requirements as reusable procedural patterns. The patterns can be matched to BPMN models from the same domain. Second, we introduced a technique for conflict detection between the specified requirements in the BPMN models. Our technique analyses enriched models using a catalog of a domain-independent anti-patterns, which was created using our SecBPMN2-Q extension as well. Alignment checking is required to avoid conflicts arising from changes to the requirements during their specifications in the business process models, which if detected later will make the process to find their root causes more difficult.

We validated the feasibility and usability of our conflict detection technique based on a case study featuring a healthcare management system, and an experimental user study, respectively.

## Chapter 3

# Integrating BPMN- and UML-based Data-Protection Engineering

This chapter presents a sub-framework of the proposed MoPrivFair (*Model-based Privacy & Fairness*) methodology in this thesis. An overview of the MoPrivFair methodology is provided in Section 1.2 in the first chapter of this thesis.



**Figure 3.1:** The highlighted part (dashed lines) denotes how Chapter 3 contributes to the overall workflow of the MoPrivFair methodology.

Tracing and integrating data protection requirements throughout the system development process is a key challenge in the field of data protection engineering. A main reason for that is, in a system development process, data protection requirements for organizational and technical aspects of a system are currently dealt with separately, giving rise to substantial misconceptions and errors.

In this chapter, we present a model-based framework<sup>1</sup> for supporting the design of a data protection-aware system on both organizational- and technical-level. The key idea is to allow involved experts in the system design phase to specify data protection requirements in languages they are familiar with: business analysts use the Business Process Modeling Notation (BPMN) for procedural system's descriptions; system architects/developers use the Unified Modeling Language (UML) to design and specify a software architectural model. Data protection requirements are captured via the language extensions SecBPMN2 and UMLsec. To specify a UMLsec architecture model based on a SecBPMN2 model, we provide a model transformation technique. Using UMLsec policies, various data protection properties of the resulting software architecture model can be verified. We applied our framework to a case study featuring an air traffic management system.

### 3.1 Introduction

The vast majority of today's software systems are part of Socio-Technical Systems (STSs). These systems involve a rich interplay of organizational (humans and organizations) and technical (software and hardware) components to accomplish shared objectives [138]. Examples include air traffic management, smart cities, and healthcare systems. Socio-technical systems open up a new class of data protection challenges. Socio-technical systems are decentralized and their components are autonomous and loosely controllable. Hence, a violation of a data protection requirement in a single component may affect other components, leading to unpleasant consequences such as privacy violations, data misuse, law infringements, and safety risks. For instance, in an air traffic management system, unauthorized modification of the flight plan of a flying plane can threaten the safety of passengers, with severe consequences for the airline and airport companies.

For the effective data protection engineering in a socio-technical system, it is important to consider both organizational and technical data protection requirements right from the start of the system development process. Therefore, various approaches have been proposed to incorporate data protection requirements into the design phase of the system development life cycle.

A number of *BPMN-based approaches* (e.g., SecBPMN2 [111, 129], SecureBPMN [17]) rely on business process modeling for organizational data protection requirements such as security and privacy requirements. These approaches abstract from technical details to allow the specification of *high-level* data protection requirements by non-technical stakeholders, such as business analysts [126]. Moreover, a number

---

<sup>1</sup>This chapter shares material with the MoDELS'17 paper "From Secure Business Process Modeling to Design-Level Security Verification" [108] and the SE'18 paper "Integrating BPMN- and UML-based Security Engineering via Model Transformation"[110].

of *UML-based approaches* (e.g., UMLsec [62], SecureUML [81]) allow system developers to design a software model while considering data protection requirements. This software model is enriched with technical details at a *low-level* of abstraction to support the validation against pre-defined technical data protection policies. The software model is a cornerstone for further development stages, such as generating code for the implementation [62, 67].

In system development process, the business process models represent an important source of requirements for the software architecture construction [31, 72, 117, 120]. For example, the business process modeling represents an essential first phase in the rational unified process, which provides a disciplined software engineering approach [73]. Since building systems that preserve data protection requirements is a sensitive task, it is important to manage data protection requirements in a system's business process model and its software architecture consistently.

### 3.1.1 Problem statement and research questions

Data protection requirements are dealt with separately. This is because existing BPMN- and UML-based approaches address data protection needs in distinct development phases and from the perspectives of different stakeholders. As a consequence, an alignment of data protection requirements across the business process- and software architecture-models is not guaranteed [11, 118].

A non-alignment of data protection requirements may lead to security and privacy vulnerabilities. A main source of vulnerabilities are misunderstandings between expert stakeholders, as triggered by their implicit knowledge about terminology [11]. For instance, one of the most common uses of *swimlanes* in BPMN is to express internal roles in an organization, while the BPMN standard leaves their semantics undefined [1]. Without this implicit knowledge about the usage of swimlanes, system developers may neglect the use of an appropriate role enforcement mechanism (e.g., role-based access control), resulting in security loopholes. Worse, such loopholes may be notoriously hard to detect due to the lack of a traceability mechanism for data protection requirements across the different modeling phases. According to Yu et al. [152], "*a single inaccurate requirement traceability link assumed by developers may already be exploited by malicious attackers*". Ensuring traceability of data protection requirements is therefore important.

Model transformation [134] is a promising direction to address this problem. The use of different kinds of models during the system design phase leads to the challenge of keeping the models consistent with each other. At this point, model transformation techniques play a central role. Thus we need a model transformation from business process models to software architecture models which ensures traceability of data protection requirements.

Earlier automated transformation approaches used UML as sole modeling language throughout the entire process [55], thereby leaving the role of business analysts unaddressed, or focused on representing data protection requirements “*at the business analyst views*” ([120], p.2), leaving technical security concerns and the verification of security aspects to future work [120]. The specification of a UML software model based on business processes represented by BPMN or UML activity models has been subject to previous works [31, 72, 117, 120]. However, except for the discussed work in [120], none of these works considered data protection aspects during the transformation. Support for integrated management of organizational and technical data protection requirements throughout the entire development process is generally missing. Motivated by this, we investigate the following research question in this chapter:

**RQ2.** *How to support the integrated management of modeling business process and software architecture models, such that the traceability of data protection requirements is guaranteed?*

Our objective is to support the management of data protection requirements from the views of the involved stakeholders, in particular, business analysts and systems engineers, consistently in an integrated manner in order to: First, avoid the introduction of vulnerabilities during the development process due to the conceptual gap between the involved stakeholders. Second, ensure traceability for data protection requirements across business process- and software architecture-models.

### 3.1.2 Contribution

To address research question RQ2, we propose a framework for integrating BPMN- and UML-based data-protection engineering while ensuring traceability. In particular, our framework suggests to iteratively:

- (i) create business process model enriched with organizational data protection requirements using the SecBPMN2 modeling language,
- (ii) transform the SecBPMN2 model to a preliminary system architecture model enriched with data protection policies using UMLsec [62],
- (iii) refine the generated UMLsec model manually with additional design decisions, and
- (iv) verify the resulting UMLsec model against its contained data protection policies by using an automated tool, called CARiSMA [4].

The novelty of our framework is that we automatically establish traceability between high-level data protection requirements and verifiable technical data protection policies. In doing so, we integrate the views of business analysts and system developers, the two main kinds of expert stakeholders during the development of the targeted system. Our contributions in this chapter are:

1. a *semi-automated process* for enforcing an integrated data-protection management throughout the system development process.
2. a *model transformation* supporting the translation of business process models annotated with data protection requirements to system architectural models while establishing traceability for data protection requirements.
3. a *case study* featuring an air traffic management system, showing how our framework can be used to establish integrated management and traceability of data protection requirements.

This chapter is structured as follows. Section 3.2 describes the necessary background. Section 3.3 describes our proposed framework for supporting the integration between BPMN- and UML-based data-protection engineering approaches. Section 3.4 describes the model transformation. Section 3.6 demonstrates the integrated approach on a case study describing an air traffic management system. Section 3.7 discusses limitations of our current approach and presents future work. Section 3.8 and 3.9 survey related work and conclude, respectively.

## 3.2 Background

In this section, we describe necessary background for understanding this chapter, focusing on the approaches SecBPMN2 and UMLsec, which we illustrate with example models from our case study.

### 3.2.1 SecBPMN2 (Revisit)

In this section, we revisit the background information of SecBPMN2. A detailed account of SecBPMN2 is already provided in Section 2.2.6. SecBPMN2 [111, 126] is a data protection-oriented modeling language that allows business analysts to express and reason with organizational data protection requirements using the BPMN [1]. Similar to other BPMN-based data protection-oriented extensions, SecBPMN2 hides technical details of the targeted system, while permitting non-technical stakeholders to specify high-level data protection requirements in an intuitive way [126].

We selected SecBPMN2 [129] as a basis for our framework due to its expressiveness and the ability to model both business processes and data protection requirements.

To specify security-specific requirements in business process models, SecBPMN2 provides 10 security annotations, such as *Accountability* (🔒), *Confidentiality* (🔒), and *Integrity* (📄) that can be added and linked to BPMN 2.0 elements such as *tasks*, *data objects*, and *message flows*. In Chapter 2 of this thesis, we extended the SecBPMN2 modeling language with 4 data-minimization-specific annotations and 1 fairness-specific annotation. A number of alternative approaches have been proposed in the literature where specific annotations are introduced to extend BPMN with data protection aspects [17, 89, 119]. However, different from the SecBPMN2 these approaches are not designed to express requirements [89], or they permit to represent only a restricted set of data protection aspects [17, 119]. More details about the 10 security concepts that are supported by the SecBPMN2 and our extension are provided in Section 2.2.6 and Section 2.4, respectively.

### SecBPMN2 example model

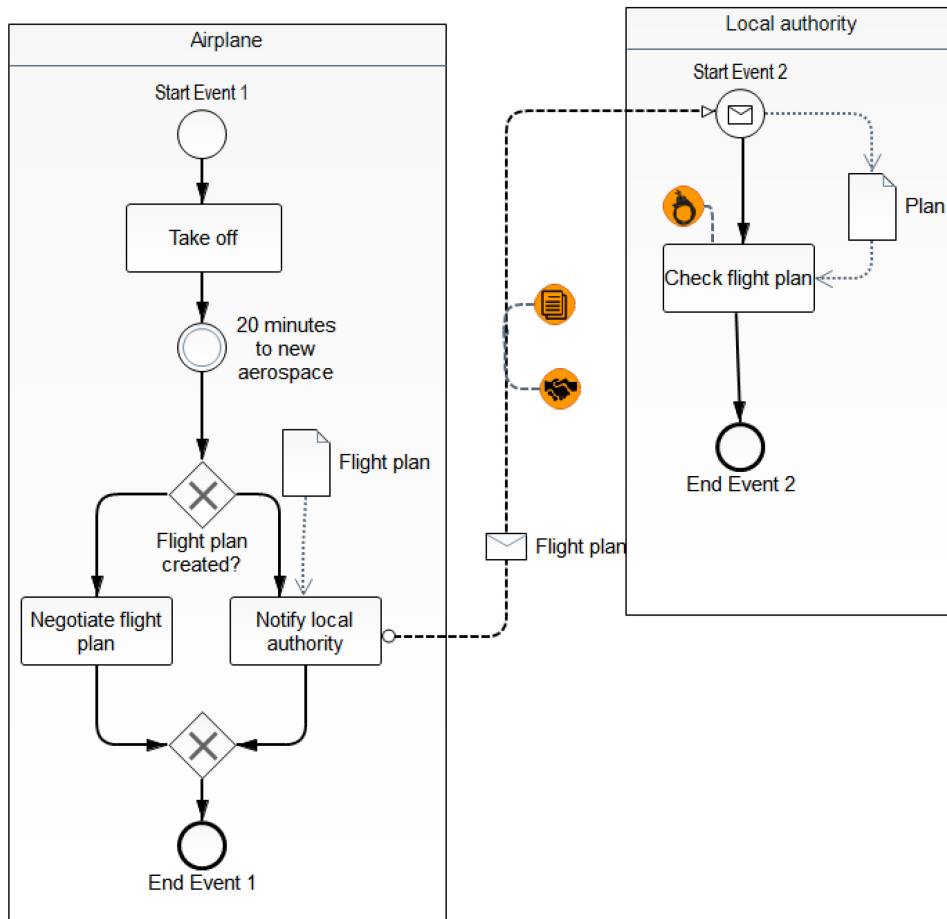
In the following, we introduce a SecBPMN2 example model, which will be used through this chapter to explain our contribution<sup>2</sup>. Figure 3.2 shows a SecBPMN2 model representing a business process for flight plan negotiation in an air traffic management system. Executors of a business process are represented by *Pools* such as “Airplane” and “Local authority”. Communications between pools are represented by *Message Flows*; the content of such communications are *Messages*: “Notify local authority” sends the message “Flight plan” to “Local authority”. Atomic activities in Figure 3.2 are represented with *Tasks*, for example “Take off”. The sequence of elements executed is represented with thick arrows, which specify that the element attached to the source of an arrow is executed before the element attached to its tip.

Events are represented with circles. *Start events* and *End events* mark the initial and terminal points of business processes. *Catch events* represent points in a business process where an event needs to happen, for example “20 minutes before new aerospace”. Event that receive or send message is represented by a circle with a message data item. For example, “Start Event2” denotes that the “Local authority” will start after receiving a “Flight plan” message. Gateways are used to specify deviations of the execution sequence: *Exclusive Gateway* specify decisions points: the gateway “Flight plan created?” allows the upper or lower branch to be executed, depending on whether the question is answered “Yes” or “No”.

The orange solid circles are security-specific concepts. In particular, *Integrity* (📄) and *Confidentiality* (🔒) are associated to the message flow meaning respectively that the content of the message is preserved and it will not be accessed by unauthorized users. *Accountability* (🔒) is associated to “Check flight plan” meaning that the task’s executor must be monitored.

<sup>2</sup>In this chapter, we use *italic* for concepts and “sans serif font” for examples.





**Figure 3.2:** Example model: SecBPMN2 model representing a business process for flight plan negotiation.

### 3.2.2 UMLsec profile

UMLsec [5, 62] is a UML *profile* that can be used to enrich UML diagrams with data protection information. A profile is a generic extension mechanism that permits refining the meta-model of the UML to be tailored for specific domains or platforms. UMLsec extends UML with security- and privacy-specific «*stereotypes*» and {*tags*}, that permit checking whether the architectural and the behavioral aspects of an annotated UML model preserve specific security or privacy policies. Verifying UMLsec policies can be done automatically by a tool support called CARiSMA<sup>3</sup> [4].

<sup>3</sup>CARiSMA is available online at <https://rgse.uni-koblenz.de/carisma/> (accessed: 05/12/2019). Details description of how to install and use CARiSMA is provided in Appendix (B) of this thesis.

UMLsec has shown its usefulness in several industrial applications [61, 64, 133], in-house training courses exist at several companies. This section focuses on UML deployment and class diagrams, which are typically used to define a software architecture. The UMLsec provides three main security-specific data-protection policies that parts of software architecture are supposed to obey. These policies are:

1. «*secure links*» to ensure that security requirements on the communication such as «*secrecy*» and «*integrity*») are met by the physical layer [62].
2. «*secure dependency*» to ensure that dependent parts in the architecture model preserve the security requirements relevant to the part they depend on [62].
3. «*abac*» to define central elements of the Role Attribute-Based Access Control (RABAC) mechanism such as roles and permissions and verify them against the designed architecture [5].

In the following, we explain these policies through the use of UML architecture models from an air traffic management system. The architecture models reflect the parts that are relevant to the flight plan negotiation process, which is discussed in Section 3.2.1. An account and a formal foundation of the UMLsec stereotypes and tags is given in [62]; the RABAC extension was introduced in [5].

### Secure links

The «*secure links*» is a UMLsec policy that aims to ensure that security requirements on the communication are met by the physical layer [62]. The physical layer of a system is modeled by a deployment diagram. Figure 3.3 is a deployment diagram annotated with the «*secure links*» policy and its related annotations.

A *deployment diagram* aims at capturing the relationship between the *physical element* of a modeled system and the software *Artifacts* ([3], p. 653). A physical element is represented as a *Node* and shown as a perspective view of a cube labeled with the name of the physical element. For instance, the “Airplane\_Client” and “Airplane\_Subsystem” in Figure 3.3 are examples of nodes. An artifact represents an information asset that is used or produced by a software process ([3], p. 656).

An artifact is deployed on a node and represented as a rectangle with the stereotype «*artifact*». For instance, in Figure 3.3, the “airplane\_GUI” represents a Graphical-User-Interface artifact. Artifacts are used to manifest systems’ components and they may have operations that can be performed on its instances.

The nodes may be connected by solid lines representing *Communication Paths*. A communication path is an association between two nodes, through which they may

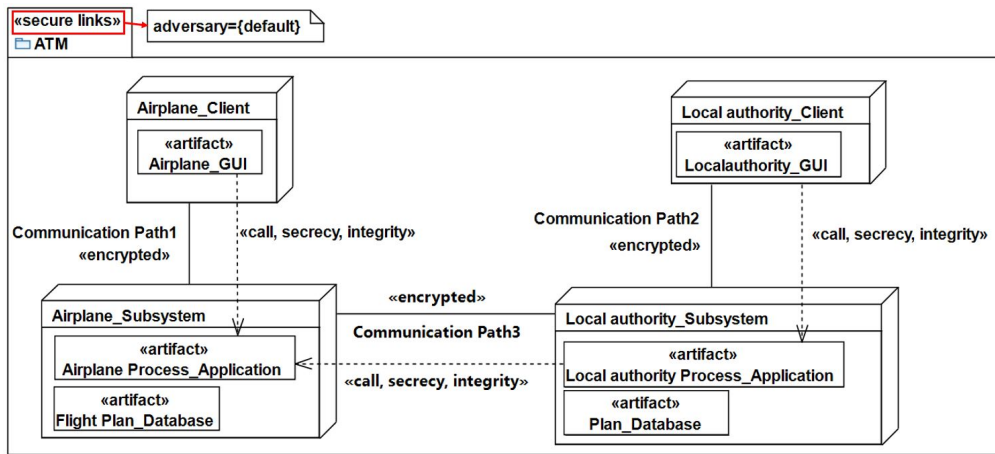


Figure 3.3: UML deployment diagram of SecBPMN2 example

exchange signals and messages. Examples of communications paths in Figure 3.3 include “Communication Path1” and “Communication Path2”. In UMLsec a communication path can be stereotyped with *«Internet»*, *«encrypted»*, *«LAN»*, or *«wire»*. These stereotypes on communication paths denote the respective kinds of communication paths. Based on the UMLsec specifications, each communication path should be stereotyped with at most one of these stereotypes ([62], p. 56).

The *«secure links»* policy is specified in relation to a specific adversary pattern, showing the potential threats that can be posed by certain types of attackers with respect to the type of the communication path (e.g. *«Internet»* and *«encrypted»*). Table 3.1 shows the threats posed by two examples adversaries, called default (i.e., outsider) and insider (i.e., one of the involved roles, such as sender). For a given adversary of type  $A$ , the function  $Threat_A(s)$  returns which kinds of actions the adversary can apply to a communication path annotated with the stereotype  $s$ . For example, considering an *«Internet»*-annotated communication path, the  $Threat_{default}(Internet)$  returns that a default adversary can delete, read and insert messages over this *«Internet»*-annotated communication path.

Table 3.1: UMLsec adversary patterns.

Stereotype	$Threat_{default}(s)$	$Threat_{insider}(s)$
<i>«Internet»</i>	{delete,read,insert}	{delete,read,insert}
<i>«encrypted»</i>	{delete}	{delete,read,insert}
<i>«LAN»</i>	$\emptyset$	{delete,read,insert}
<i>«wire»</i>	$\emptyset$	{delete,read,insert}

Artifacts may be connected by broken arrows representing communication *Dependencies*. A dependency may be annotated with the stereotype *«call»* or *«send»*. A dependency stereotyped *«call»* between two artifacts indicates that instances of the

source artifact may call operations of instances of the target artifact. A dependency stereotype «*send*» between two artifacts indicates that instances of the source artifact may send signals to instances of the target artifact. Different from the communication paths, dependencies describe the connection between the artifacts. In UMLsec a dependency can be also annotated with «*secrecy*», «*integrity*», and «*high*». These stereotypes are used in the constraint for the stereotype «*secure links*».

The «*secure links*» when label a deployment diagram enforces that for each dependency  $d$  with stereotype  $r \in \{\text{«secrecy»}, \text{«integrity»}, \text{«high»}\}$  between two artifacts deployed on two nodes  $n, m$ , we have a communication path  $l$  between  $n$  and  $m$  annotated with stereotype  $s$  such that:

- in case of  $r = \text{«high»}$ , we have  $\text{Threat}_A(s) = \emptyset$ ,
- in case of  $r = \text{«secrecy»}$ , we have  $\text{read} \notin \text{Threat}_A(s)$ , and
- in case of  $r = \text{«integrity»}$ , we insert  $\notin \text{Threat}_A(s)$ .

For example, if a communication path  $l$  between two nodes  $n, m$  is annotated with «*Internet*», and the dependency between two artifacts  $a1$  and  $a2$  that are deployed on  $n$  and  $m$ , respectively, is annotated with «*secrecy*», then the security constraint associated with the stereotype «*secure links*» with respect to the *default* adversary is violated. Specifically, the dependency annotated with «*secrecy*» requires that a default adversary should not be able to read messages over the communication path between  $n$  and  $m$ . However, the communication path is annotated with «*Internet*», meaning that the adversary is capable of reading messages over the communication path. As a result, the security requirement of the communications is not supported.

### Secure dependency

As mentioned earlier, «*secure dependency*» is a UMLsec policy that aims to ensure that dependent parts in the architecture model preserve the security requirements relevant to the part they depend on [62]. In the «*secure dependency*» policy, the «*critical*» stereotype labels classes with sensitive data and operations. The associated tags with the «*critical*» stereotype such as  $\{\text{secrecy}\}$  and  $\{\text{integrity}\}$  specify security requirements on these data and operations. Specifically, each class specifies the *stipulated* requirements of its own members and the *fulfilled* requirements of other classes' members. Verifying «*secure dependency*» entails checking whether: First, all requirements stipulated by a class are fulfilled by all dependent classes. Second, the «*call*» and «*send*» dependencies between classes respect the stipulated security requirement by the classes on the data that may be communicated between them.

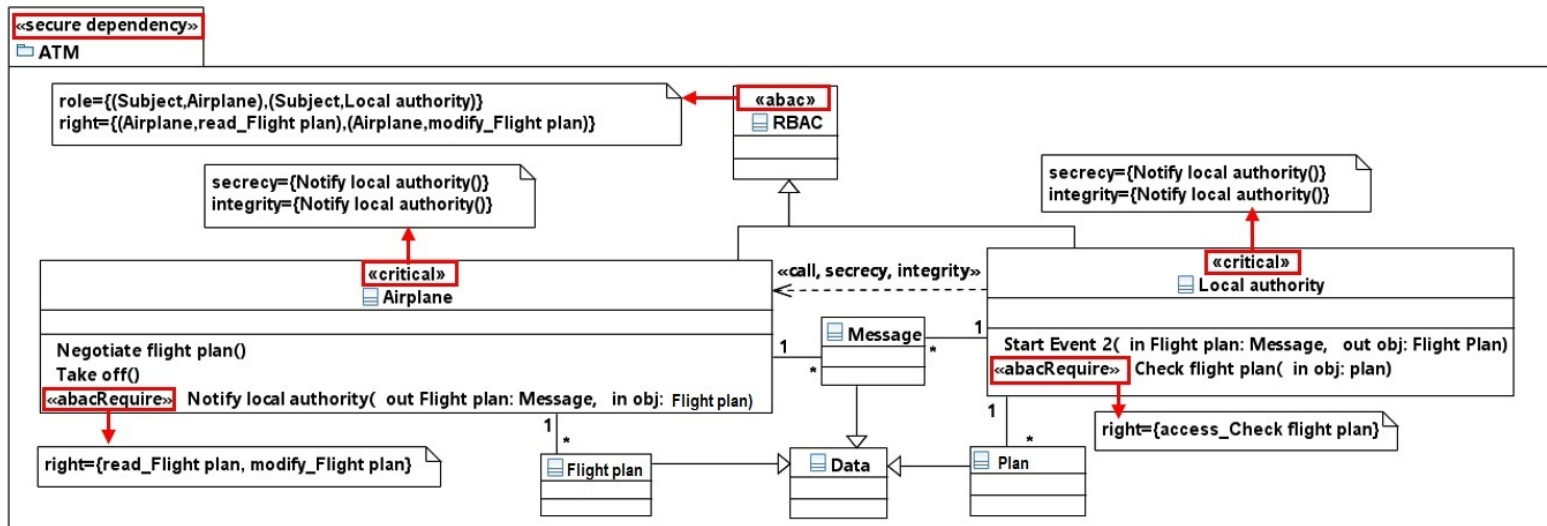


Figure 3.4: UML class diagram of SecBPMN2 example.

Figure 3.4 is a *class diagram* annotated with the «*secure dependency*» along with its related annotations. A class diagram describes the structure of a software by showing its classes, their attributes, operations, and the relationships among classes. Figure 3.4 shows a call dependency from the “Localauthority” class to the “Airplane” class. Specifically, the “Localauthority” class does not implement the “Notifylocalauthority()” operation to get the “Flight plane”, therefore, it calls the “Airplane” class.

In the “Localauthority” class, the tags *{secrecy}* and *{integrity}* are stated for the operation “Notifylocalauthority()”, denoting that the “Notifylocalauthority()” should be protected from unauthorized access (secrecy) and manipulation (integrity). Since both the «*call*» dependency and the “Airplane” class provide similar security requirements, we say that the secure dependency between the “Localauthority” class and the “Airplane” class is preserved.

### RABAC (Role attribute-based access control)

The purpose of RABAC is to check the access rights of each role and the access constraints assigned to specific operations based on predefined attributes. UMLsec implements the RABAC access control model via the policy «*abac*». The stereotype «*abac*» uses two tags called *{role}* and *{right}* to assign roles to subjects and rights to roles, respectively. For instance, in Figure 3.4, consider the “RABAC” class which is annotated with «*abac*» stereotype. The *{role}* tag of «*abac*» specifies two roles namely, the “Airplane” and the “Local authority”. The *{right}* tag specifies multiple rights to the “Airplane” as follows: “{(Airplane,read\_Flight plan), (Airplane,modify\_Flight plan)}”. This means that the Airplane has the right to read and modify the flight plan.

In «*abac*», operations in need of an access restriction can be annotated with the «*abacRequire*» stereotype along with its associated *{right}* tag. For example, in Figure 3.4, the “Notifylocalauthority()” operation is annotated with «*abacRequire*» stereotype. This stereotype specifies that the “Notifylocalauthority()” operation can be accessed by roles that have the rights to “(read\_Flight plan, modify\_Flight plan)”. In our example, the output result for the RABAC check will show that the Airplane role has access to Notify local authority operation.

## 3.3 Framework for Integrating BPMN- and UML-based Data-Protection Engineering

Business processes are mainly about behavior, tasks, and flows, which specify how a system achieves its goals, while architectural deployment and class diagrams focus on structural elements such as components, classes, and operations, which specify how a software architecture in a system should be built to achieve its goals.

Therefore, it is not possible to automatically specify all the architectural details based on a given business process. For example, while a task in the business process can be transformed to an operation in the architectural model, a one-to-one mapping is not always possible, as it might be preferable to represent the task by a set of operations rather than one. For this reason, our framework proposes the semi-automated process shown in Figure 3.5.

The proposed framework in Figure 3.5 aims at supporting the management of data protection requirements from the views of the involved expert stakeholders, in particular, business analysts and systems engineers, in an integrated manner. To this end, as shown in Figure 3.5, we integrate the two well-known model-based security approaches SecBPMN2 and UMLsec via model transformation. In the following we explain: First, the roles that should be involved in the proposed framework. Second, the input and the output of the our framework. Third, the phases of our proposed framework.

**Roles:** At a minimum, the process of the proposed framework involves a team of *SecBPMN2-trained business analysts* and *UMLsec-trained system developers*. The assumptions on the skills of these involved stakeholders are light-weight, as they do not have to be experts in data protection. Still, to ensure the correct use of SecBPMN2 and UMLsec, some additional instruction on top of their regular training is appropriate. With these assumptions, our aim is to address the common situation in which experts of data protection are absent [58]. In this situation, we aim to make the software more reliable than it would be when ignoring data protection from the start. However, even in presence of experts in data protection—which is clearly the preferable situation—the proposed framework can still be helpful, as the involved expert and non-expert stakeholders may benefit from the traceability of data protection requirements across development phases.

**Input/Output:** As input, the process receives a *requirements document* containing organizational, technical, and data protection requirements. During the requirements elicitation, the business analysts produce this document in interaction with the customer. In our work, we assume that the requirements document is given. The output is a data-protection-aware software architecture and a verification report, showing the results of validating the architectural model against the data protection policies of the UMLsec.

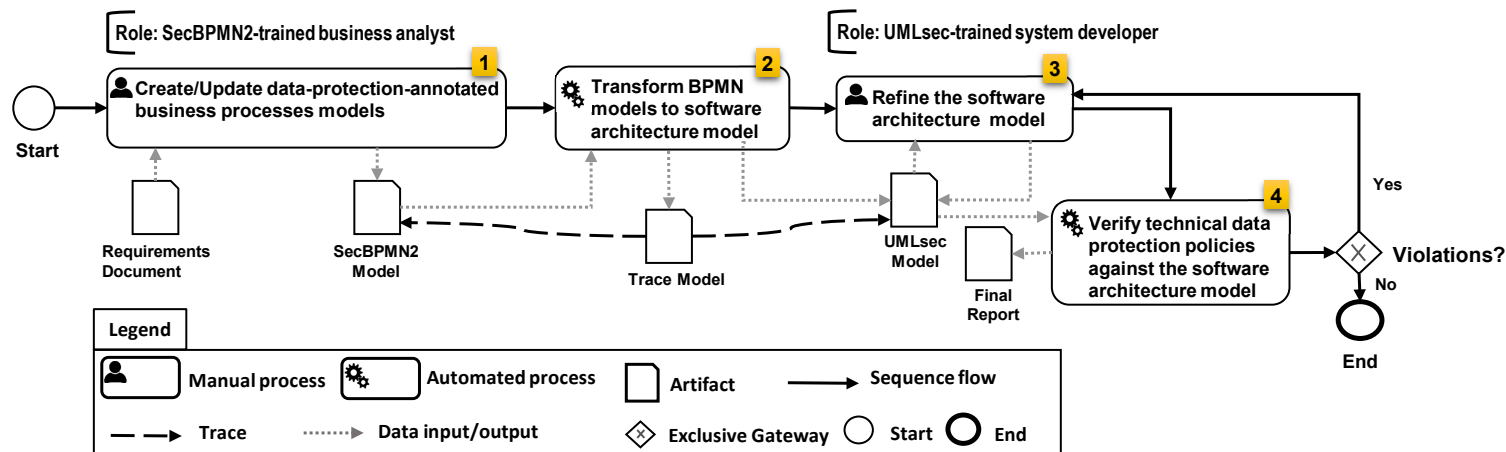


Figure 3.5: The proposed integrated management framework.



**Phase 1.** In this phase, business analysts design the business processes of the system in question with respect to data protection needs. As a first step, they derive business processes from the provided requirements document to create a BPMN 2.0 model. In a second step, they use SecBPMN2 to specify organizational data protection requirements, again based on the requirements document. The output of this phase is a *SecBPMN2 model*. The SecBPMN2 model is a BPMN 2.0 model annotated with data protection requirements.

**Phase 2.** Business process models deal with organizational aspects in a high level, abstracting from technical details. Consequently, they are not sufficient for generating software implementation directly. To this end, business processes and their included data protection requirements are now transformed into a data protection-aware architecture model. We provide an automated model transformation from SecBPMN2 to UMLsec models (i.e., UML class and deployment diagrams annotated with the UMLsec profile). As a byproduct, this transformation creates a *trace model* of mappings to source and target elements, allowing us to keep processes model and a software architecture aligned and bridge the gap between organizational requirements and technical ones. Section 3.4 elaborates on the transformation implementation using the Henshin transformation language [12]. Since it is infeasible to foresee all desirable architectural details from the input business process, the generated output from this phase can be seen as a preliminary *architecture model* that needs to be revised by the system developers.

**Phase 3.** In this phase, system developers can refine the preliminary architecture model in two ways. First, missing details can be inserted, such as the associations' names, attributes, and permissions of certain roles generated during Phase 2. Second, a UML element can be refined into multiple ones. For instance, classes with many contained operations can be split into several ones. Third, using the trace models, system developers can check whether a UMLsec data protection-related stereotype is in place for each data protection-related annotation specified in the SecBPMN2 model.

**Phase 4.** The architectural model in phase 3 can be verified against the included data protection policies using CARISMA [4]. The considered data protection policies (i.e., «*secure links*», «*secure dependency*» and «*abac*») are explained in Section 3.2.2. The output *final report* contains the result of the check of each verified policy. If the architectural model does not satisfy all data protection policies, the architectural model needs corrections; the process then jumps back to phase 3. If all data protection policies are correctly enforced, the involved stakeholders have evidence that the architecture meets the organizational data protection requirements specified in the business processes model. Therefore, one can guarantee that the architecture model is aligned with the business processes and can serve as the basis for an implementation that preserves data protection requirements.

### 3.4 SecBPMN2 to UMLsec Transformation

The specification of a software architectural model from a business processes model is not straightforward in real-world systems, which are large and complex. To address this challenge, we define an automated model transformation from SecBPMN2 models to UMLsec architectural diagrams (i.e., deployment and class diagrams), using the model transformation language *Henshin* and its associated toolset [12, 142]. Henshin is based on the Eclipse platform and the Eclipse Modeling Framework (EMF). It supports a graph-based specification of model transformation rules for EMF-based meta-models. The rationale for using Henshin was its convenient application to our setting with respect to our goals, including the possibility to create a trace model during the transformation to manage traceability. The trace model has a single class *Trace*. The *Trace* class has two non-containment n-ary references of type EObject called *source* and *target*. In our transformation, the source is a SecBPMN2 model element while the target is a UMLsec model element. We focus on the part of SecBPMN2 that can be translated to an architectural model expressed using deployment and class diagrams. These diagrams enable the specification of data protection policies in different design views of the system.

#### 3.4.1 Mapping schema from SecBPMN2 to UMLsec elements

To specify the transformation rules systematically, we first define a mapping schema from SecBPMN2 to UMLsec elements. In Table 3.2, the considered business model elements are linked to suitable UML elements in the deployment and class diagrams as follows:

**Table 3.2:** SecBPMN2 elements to UML elements.

BPMN Model	UML Deployment Diagram	UML Class Diagram
Process	Node	—
Pool	Node	Class
SwimLane	Node	Class
Data Object	Artifact	Class
Task	—	Operation
Event	—	Operation
Message Flow	Communication Path	Association
Data Association	—	Association
Security Association	Dependency	Dependency




\*\* The symbol (—) indicates that the BPMN element has no mapping to a UML element in the corresponding UML diagram.

- **[Process To Node]:** Since one cannot automatically identify from the SecBPMN2 model whether a set of processes is to be run on one node in the UML deployment diagram or a set of nodes, we assume that each *Process* is running on a separate node, and therefore, is mapped to a *Node*. The node name is the name of the process, followed by "\_Subsystem".
- **[Pool/SwimLane To Node]:** A role played by a participant (i.e., *Pool* or *Swimlane*) in the SecBPMN2 model is mapped to a *Node* in the UML deployment diagram. The node has the name of the participant, followed by "\_Client".
- **[Message Flow To Communication Path]:** A *Message Flow* in SecBPMN2 is used to pass messages between two processes. Since the processes are mapped to nodes in the deployment diagram, a *Message Flow* is mapped to a *Communication Path* that carries the communications between the corresponding nodes in the UML deployment diagram.
- **[Data Object To Artifact]:** Each *Data Object* in the SecBPMN2 model, identified by a name, is mapped to an *Artifact*<sup>4</sup> in the UML deployment diagram. The artifact name is the name of the data object followed by "\_Database".
- **[Pool/SwimLane To Class]:** A role played by a participant (i.e., *Pool* or *Swimlane*) in the SecBPMN2 model is mapped to a *Class* in the UML class diagram. The name of each class will be the name of the corresponding participant.
- **[Security Association To Dependency]:** In SecBPMN2, a data-protection annotation is linked to a specific element by using a *Security Association*. In UMLsec, a dependency between the communicated nodes or classes is used to show the corresponding data protection requirements. Therefore, *Security Association* in SecBPMN2 is mapped to a dependency in the UML class and deployment diagrams.
- **[Data Object To Class]:** Each *Data Object* in SecBPMN2 model is mapped to a *Class* of the same name in the class diagram. SecBPMN2 uses data objects to specify dataflow. To support a technical realization of this dataflow, the class diagram needs to provide appropriate classes, which are instantiated by objects in the running system. There can be multiple of these objects as repeated executions of the process require fresh objects.
- **[Task/Message Event To Operation]:** Each *Task* or *Message Event* owned by a participant in the SecBPMN2 model is mapped to an *Operation* in the corresponding class of the participant in the class diagram.

---

<sup>4</sup>In earlier UMLsec versions, this information was expressed using components in deployment diagrams, which is not supported in UML2. We mapped the data objects to artifacts which are used to manifest system components.

**Table 3.3:** SecBPMN2 security annotations to UMLsec security policies.

SecBPMN2 Security Annotations		UMLsec Security Policies	
		Deployment diagram	Class diagram
	Accountability	—	«abac» «abacRequire» {right}
	Confidentiality	«secure links» {adversary} «encrypted» «secrecy»	«abac» «abacRequire» {right} «secure dependency» «critical» {secrecy} «secrecy»
	Integrity	«secure links» {adversary} «encrypted» «integrity»	«abac» «abacRequire» {right} «secure dependency» «critical» {integrity} «integrity»

\*\* The symbol (—) indicates that the SecBPMN2 security annotation has no mapping to a UMLsec security policy in the corresponding UML diagram.

- **[Data Association To Association]:** A *Data Association* in SecBPMN2 is used to link a task or an event to a data object. The *Data Association* is mapped to an (1 : n) association between the class that represents the participant owning the task/event, and the class representing the data object.

The underlying background for our mapping schema benefits from related works aiming to relate BPMN with UML structural diagrams [31, 72, 117, 120]. However, we extended the mapping specifications proposed in these works for supporting the transformation to other UML elements that are needed for specifying data protection requirements, such as dependencies, communication paths, and artifacts. More details about the mapping specifications are provided as part of our discussion for the transformation rules in Section 3.4.2. In what follows, we explain our rationale for the mapping of SecBPMN2 security-specific data-protection annotations to UMLsec architectural security-specific policies, as shown in Table 3.3.

*Confidentiality/Integrity (Data Object/Message Flow).* In the SecBPMN2, the *Confidentiality* and the *Integrity* annotations can be attached to a data object or a message flow. The *Confidentiality*, when attached to a data object or a message flow, denotes that the data object or the message can be accessed by only authorized users [129]. The *Integrity* annotation, when attached to a data object or a message flow, denotes that the data object or the message can be modified by only authorized users [129]. A UMLsec model aligned with the *Confidentiality* and the *Integrity* annotations of the SecBPMN2 needs to include three policies:

- First, a «secure links» policy in the UML deployment diagram, to check if confidentiality and integrity of the data are preserved during transmissions. As

per Table 3.1, encrypting the data on the communication links can guarantee the confidentiality and integrity of the data against default adversary, but it does not shield against insider attackers.

- Second, the integrity and confidentiality of the corresponding data can be ensured via access control. To this end, the «*abac*» stereotype enforces the use of *Role-centric Attribute-Based Access Control*.
- Third, classes in a UML class diagram have dependencies via operation calls. To ensure that the dependencies between UML classes respect the security-specific data-protection requirements on the data communicated between them, the UMLsec «*secure dependency*» policy can be used.

*Accountability/Integrity (Task)*. The *Accountability* and *Integrity* annotations of SecBPMN2 can be attached to tasks. In this case, they express the need of monitoring a set of users when executing the task, and that the functionality of task should be protected from intentional corruption, respectively. These annotations can be enforced by employing an access control mechanisms. Therefore, we transformed the *Accountability* and *Integrity* annotations of SecBPMN2 to the UMLsec «*abac*» policy.

### 3.4.2 Transformation rules from SecBPMN2 to UMLsec

Given the mapping schema and the existing EMF implementations of SecBPMN2 and UMLsec, we define a set of Henshin transformation rules. The rules are defined graphically and applied to the input model via an interpreter engine provided by Henshin. Each rule is specified in terms of a graphs pattern, where nodes and edges represent source and target models elements and their connections, respectively. Figure 3.6 shows one of these rules, in which a *Confidentiality* SecBPMN2 annotation attached to a data object is transformed into a «*abacRequire*» UMLsec stereotype.

Each rule element has an associated action, such as «*preserve*» or «*create*». Elements with a «*preserve*» action must be matched in the model to trigger the rule's application; elements with a «*create*» action describe modifications. *Trace* elements allow correspondences between source and target models elements to be maintained, as is key for establishing traceability of data protection requirements. Rule *AddAbacRequire*, in Figure 3.6, adds a «*abacRequire*» UMLsec stereotype and its tagged value *right* to a given operation if a match for the whole *preserved* part is found. The trace element created together with the «*abacRequire*» is stored as part of a trace model, thus establishing traceability between the models.

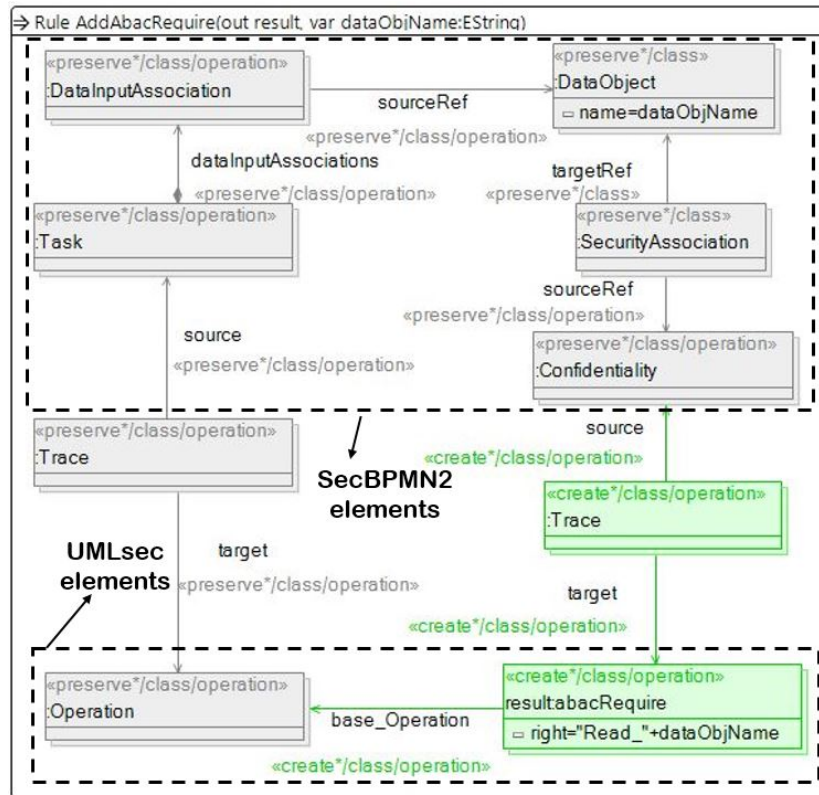


Figure 3.6: Henshin rule for adding «abacRequire» UMLsec stereotype.

Due to the large variability of possible SecBPMN2 models, we proposed 255 Henshin transformation rules. The proposed rules are divided into a set of groups, being devoted to particular goals. In the following, an abbreviated account of all transformation rules is given in textual form<sup>5</sup>. An application for the transformation rules on our example model in Figure 3.2, is provided in Section 3.6.

**A. Deployment diagram transformation rules (DR):** Based on the mapping specifications in Table 3.2 and Table 3.3, the transformation to deployment diagrams includes the transformation to deployment diagram elements, and to stereotypes related to the UMLsec «secure links» policy. For better readability, the rules are divided into a set of groups:

**Nodes and Communication paths (DR1).** Based on the mapping schema in Table 3.2, this set is responsible for producing nodes, their deployed artifacts and communication paths from the respective SecBPMN2 elements.

<sup>5</sup>A full account of our Henshin transformation rules is provided as part of our transformation tool support, which is available online at <https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec> (accessed: 31/12/2019).

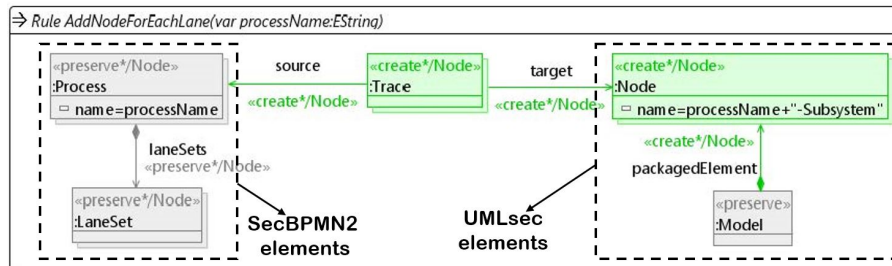


Figure 3.7: Henshin rule for adding UML nodes.

- **DR1.1** Since one can not automatically identify from the SecBPMN2 model whether a set of SecBPMN2 processes is running on one UML node or a set of UML nodes, we assume that each SecBPMN2 *Process* is running on a separated UML node, and therefore, is transformed into a *Node*. The UML node name is the name of the SecBPMN2 process contacted with "\_Subsystem". Each participant in a SecBPMN2 process has an objective that can be accomplished by one or a set of system components, therefore, each resulted UML subsystem-node has a number of artifacts based on the number of participants in the SecBPMN2 process. The name of each UML artifact is the name of the corresponding SecBPMN2 participant contacted with "\_Application".

Figure 3.7 is an example of the transformation rules that belong to **DR1.1** group. The rule *AddNodeForEachProcessLane*, in Figure 3.7, adds a node to the deployment diagram for each SecBPMN process that has swimlanes. Based on the rule, each resulted node has the name of the corresponding process, followed by "\_Subsystem".

- **DR1.2** A role played by a participant (i.e., SecBPMN2 *Pool* or *Swimlane*) is transformed into a UML *Node*. The name of each node is the name of the corresponding SecBPMN2 participant contacted with "\_Client". Each client node has a deployed artifact called *GUI* (Graphical User Interface).
- **DR1.3** A UML *Communication Path* is added between each resulted client node and its corresponding subsystem node.
- **DR1.4** If two participants, in a SecBPMN2 model, are communicating with each other via a *Message Flow*, the *Message Flow* is transformed into a UML *Communication Path* between the corresponding UML subsystem-nodes of the communicated participants in the deployment diagram.
- **DR1.5** Each SecBPMN2 *Data Object*, identified by a name, is transformed into a UML *Artifact* deployed on the corresponding subsystem node. The artifact name is the name of the SecBPMN2 data object contacted with "\_Database". If two SecBPMN2 data objects have the same name, it is assumed that they are identical, so they will be represented by the same artifact.

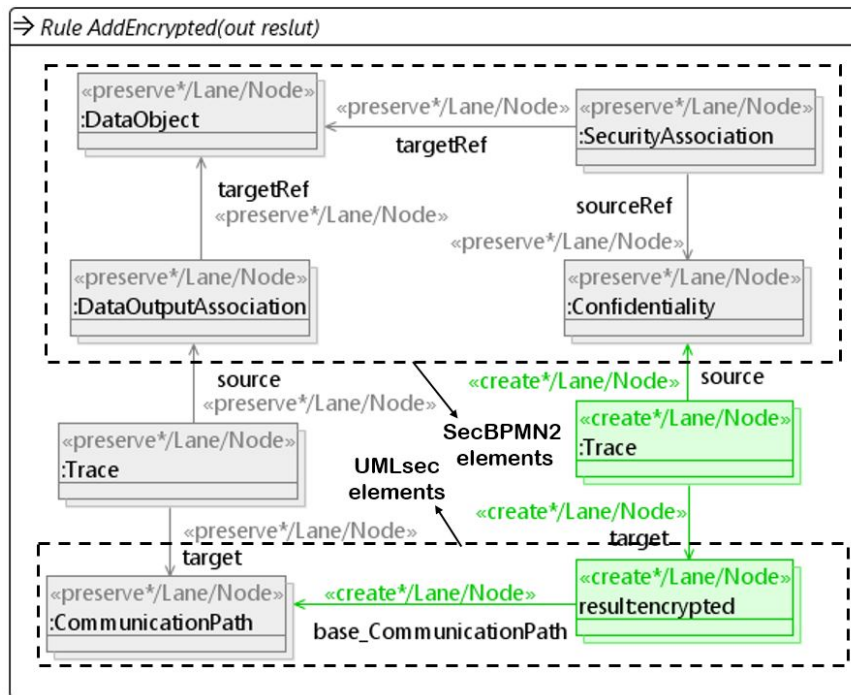


Figure 3.8: Henshin rule for adding «encrypted» to communication paths.

**Security (DR2).** This set is responsible for transforming SecBPMN2 security-specific data-protection annotations to «*secure links*» policy based on the mapping schema in Table 3.3.

- **DR2.1** If a participant, in a SecBPMN2 model, carries out a task or an event that manipulates a *Confidentiality*- or *Integrity*-annotated data object, the communication path between the corresponding UML client and subsystem node is stereotyped as «*encrypted*». For example, rule *AddEncrypted*, in Figure 3.8, shows that the *Confidentiality* SecBPMN2 annotation attached to a data object via a security association will be transformed to «*encrypted*» UMLsec stereotype. Based on the rule, the «*encrypted*» UMLsec stereotype will annotate the UML communication path in the deployment diagram that corresponds to the SecBPMN2 data output association whose target is the *Confidentiality*-annotated data object in the SecBPMN2 model.
- **DR2.2** If a participant, in a SecBPMN2 model, carries out a task manipulating a data object that is linked with a security association to a *Confidentiality* or *Integrity* SecBPMN2 annotation, the *Security Association* is transformed into a UML «*call, secrecy*»- or «*call, integrity*»-annotated *Dependency* from the GUI artifact of the participant to the corresponding artifact of the data object. For example, rule *AddIntegrity*, in Figure 3.9, shows that the *Integrity* SecBPMN2



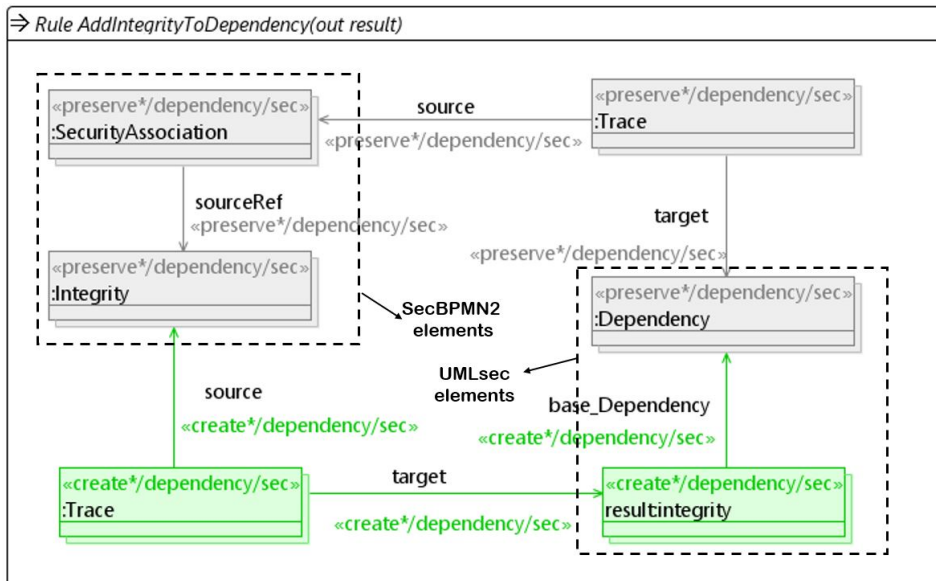
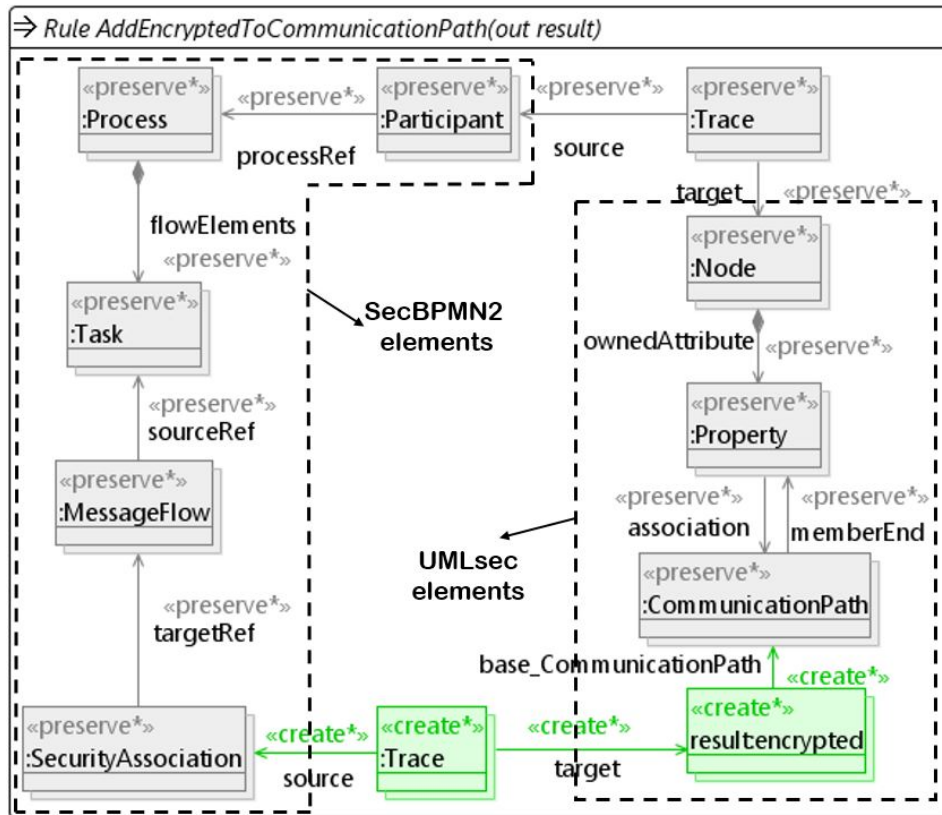


Figure 3.9: Henshin rule for adding «integrity» to dependencies.

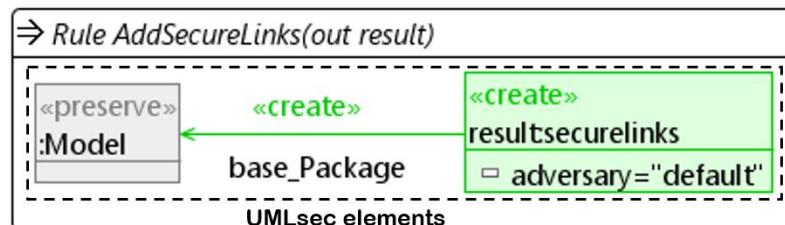
annotation attached to a data object in the SecBPMN2 model will be transformed to «integrity» UMLsec stereotype on the corresponding UML dependency to the security association. The UML «call» stereotype is added to the dependency by performing separate transformation rules.

- **DR2.3** If two participants, in a SecBPMN2 model, are communicating with each other via a *Confidentiality*- or *Integrity*-annotated *Message Flow*, the communication path between the corresponding UML subsystem-nodes of the communicated participants is stereotyped as «encrypted». In addition, the UML communication path between the client and the subsystem node corresponding to each SecBPMN2 participant is stereotyped as «encrypted». For example, rule *AddEncryptedToCommunicationPath*, in Figure 3.10, shows that each *Confidentiality*-annotated *Message Flow* whose source in the SecBPMN2 model is a task will be transformed into an «encrypted»-annotated *Communication Path* in the UML deployment diagram.
- **DR2.4** If two participants, in a SecBPMN2 model, communicate via a message flow that is linked with a security association to a *Confidentiality* or *Integrity* SecBPMN2 annotation, the *Security Association* is transformed to a «call,secrecy»- or «call, integrity»-annotated UML *Dependency* between the application artifacts which are deployed on the corresponding UML subsystems-nodes to the SecBPMN2 participants. In addition, a «call,secrecy»- or «call,integrity»-annotated dependencies are added between the *GUI* artifact and the application artifact corresponding to each participant.



**Figure 3.10:** Henshin rule for adding *«encrypted»* to communication paths correspond with SecBPMN2 message flows.

- **DR2.5** The *AddSecureLinks* rule, in Figure 3.11, shows that a generated UML deployment model is annotated with *«secure links»* stereotype being tagged with *{adversary=default}*. This rule is applied if the generated UML deployment model has at least two nodes communicated over an *«encrypted»*- or *«Internet»*-annotated *Communication Path* with *«secrecy»*- or *«integrity»*-annotated *Dependency* between their artifacts.



**Figure 3.11:** Henshin rule for adding *«secure links»* stereotype.

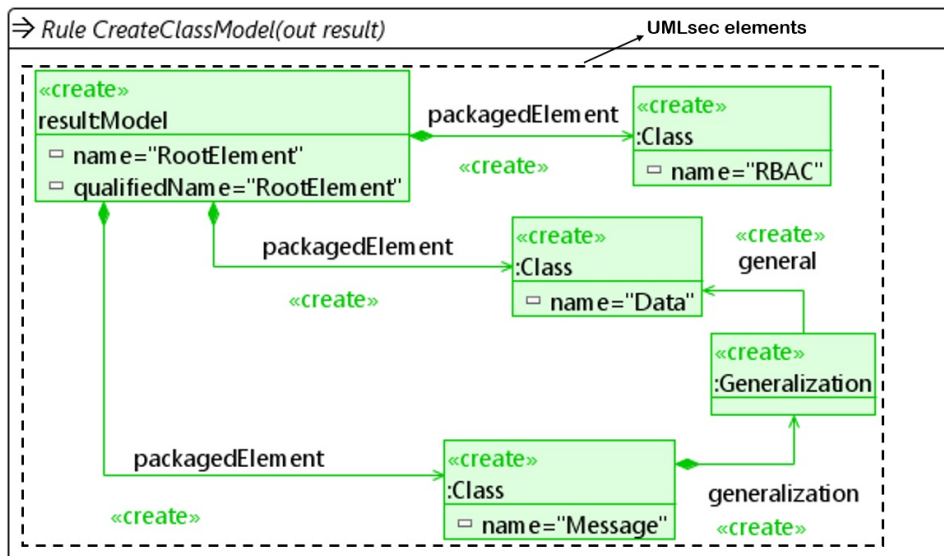


Figure 3.12: Henshin rule for importing core UML classes.

**B. Class diagram transformation rules (CR):** Based on the mapping schema, the transformation from SecBPMN2 models to UML class diagrams includes the transformation to class diagram elements, and to stereotypes related to UMLsec's *«abac»* and *«secure dependency»* policies. The rules are divided into a set of groups:

**Initialization (CR1).** This set is responsible for creating the UMLsec-annotated class diagram with some important core classes to aggregate information recurring across data protection requirements, thus improving readability. The set contains the following rule groups:

- **CR1.1** Add a class with the name *RBAC*. This class will be a super-class for all classes that play specific roles in the target system.
- **CR1.2** Add a class with the name *Data*. This class will be a super-class for all classes that are generated from transforming SecBPMN2 data objects.
- **CR1.3** Add a class with the name *Message*. Each message being sent between different participants (e.g., Pools) in a SecBPMN2 model becomes an instance of this class in the UML class diagram.

For example, following **CR1.1-CR1.3**, the rule *CreateClassModel* in Figure 3.12 adds three classes to the UML model namely, *RBAC*, *Data*, and *Message*. In addition, the rule specifies the data class as a generalization for the message class.

**Classes (CR2).** Based on the mapping schema in Table 3.2, this set is responsible for producing classes and their operations from the respective SecBPMN2 elements.

- **CR2.1** A role played by a participant (i.e., *Pool* or *Swimlane*) in a SecBPMN2 model, is transformed into a *Class* in the UML model. The name of each class is the name of the corresponding participant.
- **CR2.2** Each *Data Object* in a SecBPMN2 model is transformed into a *Class* in the UML model. The class name is the name of the SecBPMN2 data object. If two data objects in a SecBPMN2 have the same name, it is assumed that they are identical, so they will be represented by the same class in the UML model.

Following **CR2.2**, the rule *CreateClassFromDataObject* in Figure 3.13 shows that a SecBPMN2 *Data Object* will be transformed to a *Class* in the UML model. The class has the same name of the data object and specified as a subclass to the *Data* class which is added to the UML model by performing the rule *CreateClassModel* in Figure 3.12.

- **CR2.3** Each *Task* or *Message Event* owned by a participant in a SecBPMN2 is transformed into an *Operation* in the corresponding UML class of the participant. If the SecBPMN2 *Task* or *Message Event* represents the source for an *Output Data Association* or a *Message Flow*, it is transformed to an *Operation* with a return parameter of type *Data Object* or *Message*, respectively. Otherwise, if the SecBPMN2 *Task* or *Message Event* represents the target for an *Input Data Association* or a *Message Flow* it is transformed into an *Operation* with an input parameter of type *Data Object* or *Message*, respectively.

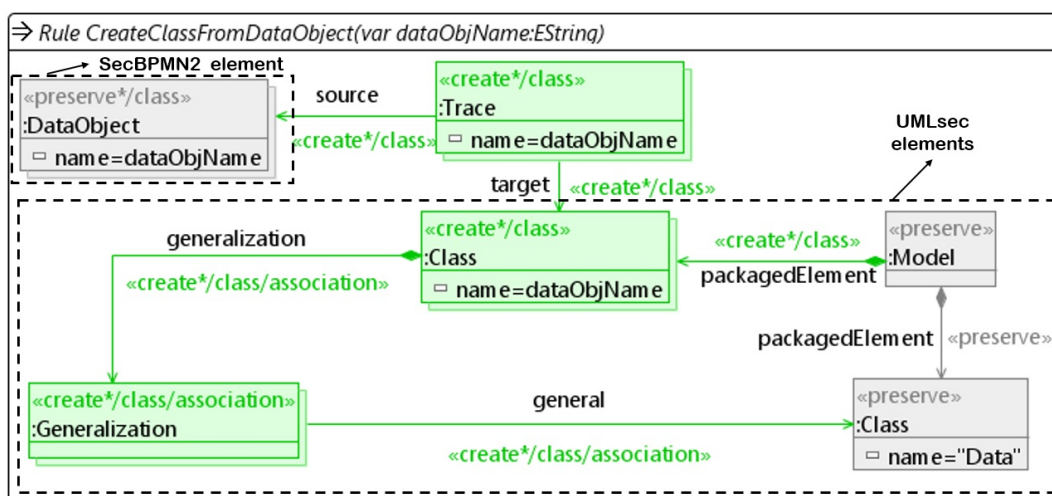


Figure 3.13: Henshin rule for transforming SecBPMN2 data object to UML class.

**Relationships (CR3).** This set is responsible for producing the UML relationships between the created classes, based on the mapping schema defined in Table 3.2.

- **CR3.1** A UML class that is corresponding to a SecBPMN2 Pool divided into multiple Swimlanes is set as super-class for the UML classes representing the Swimlanes. While the semantics of both Pools and Swimlanes are not defined by the BPMN 2.0 standard [1], we assume that a company will use them to represent main roles and internal sub-roles, which is also one of their most common usages in the field of business process modeling.
- **CR3.2** A UML class that is corresponding with a SecBPMN2 Pool is set as sub-class of the class *RBAC* in the class diagram.
- **CR3.3** If a participant, in a SecBPMN2 model, is responsible for a task or an event that manipulates a data object, the UML class that represents the participant is related to the class of the data object. Since the task or the event can be triggered several times either directly by the participant or indirectly by the system, the association between the class for the participant and the class for the data object will be (1 : n). The rationale behind this is that BPMN uses the concept of data objects to specify dataflow. To support a technical realization of this dataflow, the software design (which is specified using class diagrams) needs to provide appropriate classes, which are then instantiated by objects in the running software. There can be multiple of these objects because repeated executions of the process require a "fresh" object.
- **CR3.4** If a participant in a SecBPMN2 model is responsible for a task/event that is the source/target for a message flow, the UML class for the participant is related to the *Message* class. Since a task or an event can be triggered several times, the association between the class for the participant and the *Message* class will be (1 : n).

**Security (CR4).** This set is responsible for transforming SecBPMN2 data-protection related annotations to UMLsec «*abac*» and «*secure dependency*» policies based on the mapping specification in Table 3.3.

- **CR4.1** If a SecBPMN2 task is *Accountability*- or *Integrity*-annotated, the corresponding UML operation to the task is stereotyped with «*abacRequire*», being tagged *{right=access\_taskName}* or *{right=modify\_taskName}* in case of *Accountability* or *Integrity* SecBPMN2 annotation, respectively. For example, Figure 3.14 shows that an *Accountability*-annotated task in a SecBPMN2 model will be transformed into «*abacRequire*»-operation being tagged *{right=access\_taskName}*.

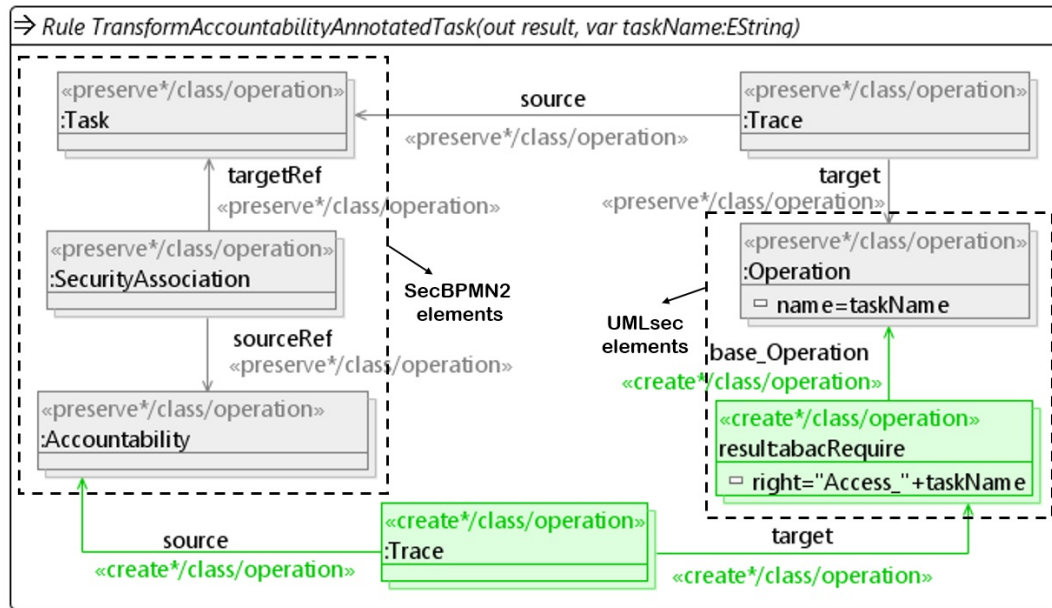


Figure 3.14: Henshin rule for transforming Accountability-annotated task.

- **CR4.2** If a participant, in a SecBPMN2 model carries out a task or an event manipulating a *Confidentiality*- or *Integrity*-annotated data object, (i) the corresponding UML classes for the participant and the data object are stereotyped as `«critical»` along with `{secrecy=TaskName}` or `{integrity=TaskName}`, (ii) the UML operation representing the SecBPMN2 task is stereotyped with `«abacRequire»`, tagged `{right=read_DataObjectName}` or `{right=modify_DataObjectName}` in case of *Confidentiality* or *Integrity*, respectively, and (iii) a `«call,secrecy»`- or `«call,integrity»`- annotated *Dependency* between the classes for the data object and the participant is created.

Following **CR4.2**, the rule *AddAbacRequire* in Figure 3.6 shows that a *Confidentiality*-annotated task in a SecBPMN2 model will be transformed to `«abacRequire»`-stereotyped operation with the `{right=read_DataObjectName}` tag.

- **CR4.3** If a participant carries out a task or an event being the source or the target for a *Confidentiality*- or *Integrity*-annotated message flow, the class for the participant is stereotyped with `«critical»` along with `{secrecy=TaskName}` or `{integrity=TaskName}`, (ii) the operation representing the task is stereotyped with `«abacRequire»`, tagged `{right=read_MessageName}` or `{right=modify_MessageName}` with respect to the SecBPMN2 security annotation, and (iii) a *Dependency* stereotyped with `«call,secrecy»` or `«call,integrity»` between the classes for the participants is created.
- **CR4.4** If the generated class model has at least one `«abacRequire»`-annotated operation, `«abac»` stereotype is attached to the generated class "RBAC", along



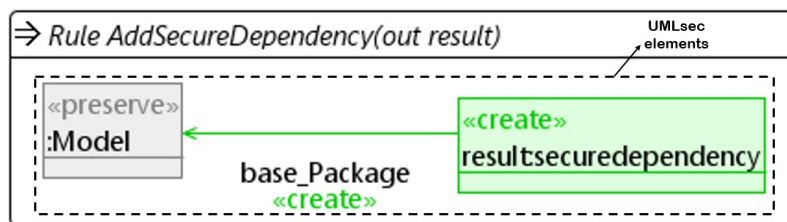


Figure 3.15: Henshin rule for adding «secure dependency» stereotype.

with two tags *{role}* and *{right}*. The *{role}* tag specifies a set of roles, where roles are the names of all subclasses for "RBAC" class. The *{right}* tag needs to be specified by the user after the transformation; it specifies the permissions associated with each role.

- **CR4.5** The *AddSecureDependency* rule, in Figure 3.15, shows that the generated UML class model will be annotated with «secure dependency» UMLsec stereotype. This rule is applied, if the generated class model has at least one a «critical»-annotated class that is being the source for a «secrecy»- or «integrity»-annotated dependency.

### 3.5 Tool Support

We developed a prototypical implementation of our work in this chapter<sup>6</sup>. In Figure 3.16, we show an artifact-centric representation of the process for applying our framework in Figure 3.5, including two automated tasks<sup>7</sup>.

The first task in Figure 3.16 is an automated model transformation process from SecBPMN2 models to corresponding UMLsec structural diagrams (i.e., deployment and class diagrams), using the model transformation language Henshin. This task is implemented using a set of transformation rules (.henshin files) and some Java code for rules orchestration. The rules are defined graphically and applied to the input models (i.e., SecBPMN models) via an interpreter engine provided by Henshin. The output of this task is a UMLsec model, and a trace model. The trace model links the SecBPMN2 and UMLsec models. Using the trace models, one can check whether a UMLsec security stereotype is in place for each security annotation specified in the SecBPMN2 model.

<sup>6</sup>Our implementation is available online at <https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec> (accessed: 31/12/2019).

<sup>7</sup>Details description of how to install and use our transformation tool and CARiSMA is provided in Appendix (B) of this thesis.

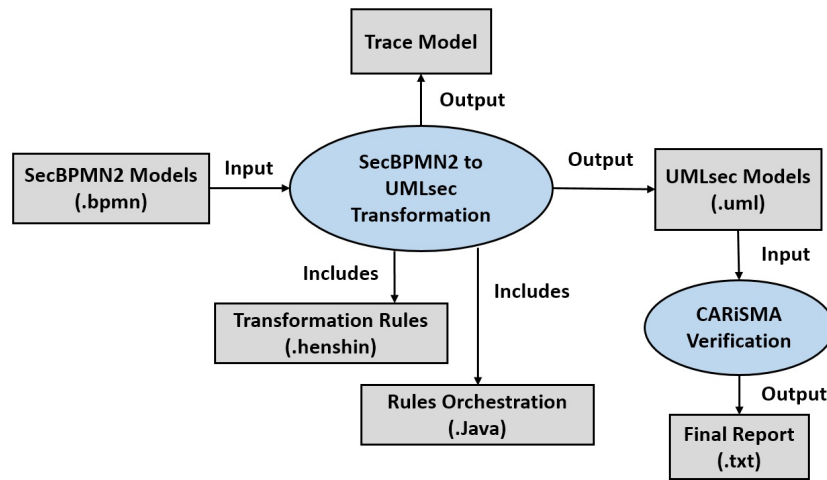


Figure 3.16: Process with involved tasks and artifacts.

In the second task of Figure 3.16, we use CARiSMA to automatically verify the generated UML models against UMLsec policies. The output of this process is a text file that summarizes the results of the verification process.

### 3.6 Case Study

To study if the proposed approach satisfies the specified goals of *integrated management* and *traceability*, we applied it in a real case study featuring the *System Wide Information Management*<sup>8</sup> (SWIM) of the Federal Aviation Administration of the United States. SWIM is a technology program focusing on information sharing for *Air Traffic Management* (ATM) systems. ATM systems consist of a large number of autonomous and heterogeneous components that interact with each other to enable ATM operations: pilots, airports, national airspace managers, weather forecast services, radars, etc. In such a complex information system, ensuring data protection is critical. For example, the leakage of secure data may result in severe consequences on safety and confidentiality. Below, we first describe and exemplify our application of the approach and then discuss if the goals are satisfied.

<sup>8</sup>Information about SWIM is available online at [https://www.faa.gov/about/office\\_org/headquarters\\_offices/ato/service\\_units/techops/atc\\_comms\\_services/swim/documentation/](https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/atc_comms_services/swim/documentation/) (accessed: 02/12/2019)



### 3.6.1 Applying the integration framework

In this section, we explain how we applied the proposed framework in Figure 3.5 on the ATM case study.

**Phase 1: Create SecBPMN2 models.** In the first phase of the framework in Figure 3.5, we assumed the role of a business analyst in our process. The task was to create SecBPMN2 models from the given requirements description. However, the documentations were already analyzed by the authors of the work in [128] for modeling and verifying of Air Traffic security requirements with SecBPMN2. Therefore, instead of creating SecBPMN2 models, we reused the created SecBPMN2 models in [128]. These SecBPMN2 models focus on three aspects<sup>9</sup>:

1. *Flight plan negotiation.* Every time an airplane enters a new aerospace, a new flight plan is negotiated with the local authority. To model the procedures which regulate such negotiation, the authors in [128] defined a SecBPMN2 model with 74 elements (data objects, tasks, events and data associations), 3 participants, and 22 security-specific annotations (accountability, confidentiality, and integrity).
2. *Landing.* Landing procedures are executed to negotiate the last part of the flight plan, which includes the approach to the airport and the waiting trajectory. The defined SecBPMN2 model contains 84 elements, 4 participants, and 19 security-specific annotations (accountability, confidentiality and integrity).
3. *External services.* The new ATM SWIM architecture permits to use external services for providing information such as the weather forecast. Services are selected based on a trust value which is updated every time a service is used and evaluated. For better readability, the authors in [128] created two separate SecBPMN2 models for this aspect, containing 164 elements, 9 participants, and 30 security-specific annotations (accountability, confidentiality, and integrity) in total.

**Phase 2: Transform SecBPMN2 models to UMLsec architecture models.** In the second phase of the framework in Figure 3.5, we automatically applied our transformation to the three aspects of the ATM case study described above. As described above, the *Flight plan* and *Landing* aspects are each represented by a SecBPMN2 model, the *External services* aspect is represented by two models for readability. Based on the rules in Section 3.4.2, a UMLsec deployment and class models are automatically generated for each SecBPMN2 model.

<sup>9</sup>The SecBPMN2 models created for the ATM case study are provided online at <https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec> (accessed: 31/12/2019)

To illustrate the outcomes, Figure 3.3 and Figure 3.4 show the produced UML deployment and class diagrams, respectively, from the input SecBPMN2 model in Figure 3.2. In the following, we explain the utilized transformation rules for producing the deployment and class diagrams.

The deployment diagram in Figure 3.3 is produced from the SecBPMN2 model in Figure 3.2 as follows:

1. Following the specifications in **DR1**, the SecBPMN2 processes have become UML subsystem-nodes each with an application deployed artifact, while the SecBPMN2 participants are now UML client-nodes, each with, GUI as a deployed artifact. For example, “Airplane”, a SecBPMN2 process with one participant, is transformed into two UML nodes; the first is the “Airplane\_Subsystem” node with a deployed artifact called “Airplane Process\_Application”, while the other is the “Airplane\_Client” node with “Airplane\_GUI” as a deployed artifact.
2. Following the same specifications in **DR1**, three communication paths between the resulted nodes are added and the SecBPMN2 “Flight plan” data object has become a “Flight plan\_Database” artifact deployed on the “Airplane\_Subsystem” node.
3. Based on **DR2.3**, since the “Airplane” and the “Local authority” in the SecBPMN2 model are communicating via *Confidentiality*- and *Integrity*-annotated message flow, the resulted UML communications paths (i.e., “Communication Path1”, “Communication Path2” and “Communication Path3”) are annotated with the UMLsec stereotype *«encrypted»*.
4. Following the specification of **DR2.4**, a *«call,secrecy,integrity»*-annotated dependency is added between the UML artifacts namely, the “Local authority Process\_Application” and “Airplane Process\_Application”. Accordingly, two *«call,secrecy,integrity»*-annotated dependencies are added, one is between the artifacts namely, “Airplane\_GUI” and “Airplane Process\_Application”, while the other is between the artifacts namely, “Local authority\_GUI” and “Local authority Process\_Application”.
5. As a final step for the transformation to the deployment diagram a *«secure links»* with *{adversary=default}* is added to the diagram based on **DR2.5**. Due to the fact that most UML modeling tools do not display tagged values of stereotypes in the diagram view, we show the tagged value as a note linked to the corresponding stereotype. The resulted diagram can be automatically verified against the *«secure links»* policy.

The class diagram in Figure 3.4 is produced from the SecBPMN2 model in Figure 3.2 as follows:

1. Following the specifications in **CR2.1**, each participant in the SecBPMN2 model has become a class in the class diagram. For example, the “Airplane” SecBPMN2 participant is transformed to a UML class with the same name.
2. Following the specification in **CR2.2**, each data object in the SecBPMN2 has become a class in the class diagram. For example, the “Flight plan” data object is transformed into a class with the same name.
3. Following the specification **CR2.3**, each task or Message Event owned by a participant in a SecBPMN2 is transformed into operation in the corresponding UML class of the participant. For example, the “Notify local authority()” task is transformed into operation in the “Airplane” class.
4. Classes are annotated with security-specific annotations. For instance, as per **CR4.3**, both the *Confidentiality* and *Integrity* SecBPMN2 annotations linked to the *Flight plan* message flow in Figure 3.2 are transformed into «*abac*» and «*secure dependency*» UMLsec security policies, as follows:
  - (i) the transformation to «*abac*» involved the attachment of «*abacRequire*» stereotype along with a tag {*right=read\_Flight plan, modify\_Flight plan*} to “Notify local authority()” operation. Again, we show the tagged values as notes connected to the corresponding stereotype.
  - (ii) following the same specification in **CR4.3**, the transformation to «*secure dependency*» stereotype involved the attachment of «*critical*» stereotypes to the corresponding classes of the participants along with {*secrecy=Notify local authority()*} and {*integrity=Notify local authority()*} as tags. Subsequently, the *Message Flow* is transformed to UML «*call*» *Dependency* stereotyped with «*secrecy*» and *integrity*.
  - (iii) based on **CR4.1** the transformation of SecBPMN2 *Accountability* security annotation involved the attachment of «*abacRequire*» along with {*right=access\_check flight plan*} to the “check flight plane()” operation.
  - (iv) based on **CR4.4-CR4.5** «*abac*» and «*secure dependency*» stereotypes are add to the generated *RBAC* class and to class diagram respectively.

**Phase 3: Refine the resulted architecture model.** Since the BPMN-based approaches do not contain the information required to generate complete UML models, our framework in Figure 3.5 suggests a refinement phase where the system developers can split large classes and specify missing details, such as attributes and association names. Also, classes with many contained operations can be split into several ones.

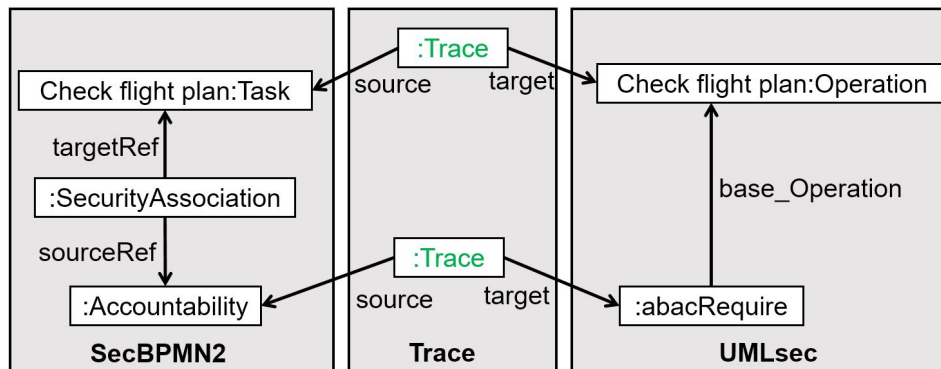


Figure 3.17: Example of the generated trace models (excerpt).

*Trace model.* As a prerequisite for managing traceability, our transformation rules create a trace model, consisting of traceability links between the source (i.e., input) and target (i.e., output) models. The generated trace model is an Eclipse Modeling Framework (EMF) model which permits generic and flexible support for traceability in Henshin. The trace model has a single class *Trace*. The *Trace* class has two non-containment n-ary references of type *EObject* called *source* and *target*. The main important utilize of the *Trace* model is the *exogenous model transformations*. Exogenous transformations are transformations between models expressed using different modeling languages such as our proposed transformation in this chapter. In our transformation, the source is SecBPMN2 model while the target is a UMLsec architecture model.

Figure 3.17 represents the trace model generated as a result of applying the Henshin transformation rule in Figure 3.14. The trace model links the SecBPMN2 and UMLsec models. Using the trace models, one can check whether a UMLsec data protection-related stereotype is in place for each data protection-related annotation specified in the SecBPMN2 model. For example, in Figure 3.17 one can see that the *Accountability*-annotated task in the SecBPMN2 (i.e., “Check flight plan”) model is transformed to *«abacRequire»*-annotated operation in the UMLsec model.

**Phase 4: Verify technical data protection-related policies.** Once the refinement phase is completed, the system developers can verify the designed UMLsec class diagram against the specified security policies using CARiSMA. For instance, the deployment diagram in Figure 3.3 allows the *secrecy* and *integrity* of the data communicated between “Airplane\_Client” and “Local authority\_client” nodes to be preserved against the *default* attacker pattern, see Table 3.1.

Moreover, as shown in Figure 3.4, using the *«abac»* policy, system developers can specify a list of rights for each role, and thereby, they can define the set of operations that can be accessed by a certain subject who plays a specific role in the system.

For example, a given subject plays an airplane role has access to the “Notify local authority()” operation. One can also observe in Figure 3.4 that the “Airplane” class and «call» dependency provide the security levels (i.e., secrecy and integrity) on the *Notify local authority()* operation that are required by “Local authority” class, and thus, the security dependency is preserved. Per comparison between the source model in Figure 3.2 and the target models namely, the deployment diagram in Figure 3.3 and the class diagram in Figure 3.4, one can guarantee that the considered security requirements are correctly traced and both models are aligned.

### 3.6.2 Transformation results

We summarize the transformation results in terms of the number of the elements in the source and the target models<sup>10</sup>. Table 3.4 denotes the source and target models of each transformation<sup>11</sup>. Each row represents one of the four model pairs *Flight plan*, *Landing*, *External services 1* and *2*. For example, the *Flight plan* model contains 3 processes, each with one participant, and 5 *Accountability*, 7 *Confidentiality* and 10 *Integrity* security annotations.

In the *Flight plan* SecBPMN2 model, the *Confidentiality* and *Integrity* annotations are linked to 10 out of 14 message flows, where the sources of message flows were generally tasks, while the targets were message-receive events. To transform the security annotations, a «*encrypted*» stereotype is assigned to the UML communication paths between the corresponding communicated nodes. Based on our expectation, the target model should include 6 nodes, and 5 «*encrypted*»-annotated communication paths as it is matched by the number in Table 3.4.

In the class diagram, a «*abacRequire*» stereotype is assigned to the linked task (in case of *Accountability*-annotated task **CR4.1**) and to the source and target task (in case of *Confidentiality*- or *Integrity*-annotated message flow **CR4.3**). In contrast to the tasks, the message-receive events are not human-controlled and, therefore, are not subject to access control in our transformation rules. 10 out of 23 tasks are a source of a *Confidentiality*- or *Integrity*-annotated message flow, and 5 of them are also *Accountability*-annotated tasks. Therefore, we should have 10 «*abacRequire*» stereotypes for the tasks and 1 «*abac*» stereotype attached to *RBAC* class, as it is matched by the numbers in the Table 3.4.

<sup>10</sup>All artifacts used for the evaluation are archived at <https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec> (accessed: 31/12/2019).

<sup>11</sup>Readers who are interested in reproducing the summarized results in Table 3.4 can follow the provided description in Appendix B of this thesis.

**Table 3.4:** Detailed Summary for the transformation results

Model	SecBPMN2						UMLsec								
	<i>Pa.</i>	<i>DO.</i>	<i>MF.</i>	<i>Ac.</i>	<i>Cf.</i>	<i>Ig.</i>	<i>No.</i>	<i>Cl.</i>	<i>EP.</i>	<i>Ab.</i>	<i>AR.</i>	<i>Cr.</i>	<i>ST.</i>	<i>IT.</i>	<i>Dp.</i>
<i>Flight plan</i>	3	6	14	5	7	10	6	12	5	1	10	3	17	26	11
<i>Landing</i>	4	5	15	5	5	9	8	12	7	1	10	4	16	24	16
<i>External services 1</i>	4	4	5	3	2	4	8	11	7	1	6	4	6	11	10
<i>External services 2</i>	5	5	17	4	3	14	10	13	10	1	18	5	8	42	21

\*\* SecBPMN2 models with the numbers of: Participants (Pa), Data Objects (DO), Message Flows (MF) and Accountability (Ac), Confidentiality (Cf), and Integrity (Ig) annotations.

\*\* UMLsec models with the numbers of: Nodes (No), Classes (Cl), and Encrypted Paths (EP), and Abac (Ab), AbacRequire (AR), Critical (Cr), Secrecy (ST), and Integrity (IT) annotations and Dependencies (Dp).

**Table 3.5:** The overall transformation results and the execution time.

Model	SecBPMN2		UMLsec		Time[ms]
	Element	Security annotation	Elements	Security annotations	
Flight plan negotiation	77	22	349	83	20481
Landing procedures	88	19	366	78	24465
External services (1+2)	173	30	818	150	49945

Again based on **CR4.3**, for each *Confidentiality* or *Integrity*, two «critical» stereotypes must be generated each with same tag type (i.e., *secrecy* or *integrity*) and values; the first will be assigned to the corresponding class of the sender while the second will be assigned to the recipient class. Since we have in total 3 classes for the participants, we expect the target model to have 3 «critical» stereotypes each with one *secrecy* and one *integrity* as tags. In total, the *secrecy* tags should have 17 values, while the *integrity* tags should have 26. Table 3.4 shows that this is the case.

In the input model, the annotated message flows can be grouped into four sender-receiver pairs: (1,2), (2,1), (2,3), and (3,2). Following the specifications of **CR4.3**, the number of «*secrecy*»-and «*integrity*»-annotated dependencies in the target class diagram depends on the number of these pairs: we should have 4 dependencies in the class diagrams. Conversely, via **DR2.4**, the deployment diagram should include 7 «*secrecy*»-and «*integrity*»-annotated dependencies; 4 between the subsystem nodes and 3 between the clients and subsystem nodes. Therefore, our target model should have in total 11 «*secrecy*»-and «*integrity*»-annotated dependencies, as is the case in the table. Therefore, we can conclude that the SecBPMN2 security requirements are correctly transformed into verifiable UMLsec policies.

This chapter does not include a systematic performance evaluation of the transformation technique. To still offer first insights into the scalability of our technique, we performed a preliminary assessment based on our case study. Table 3.5 shows a summary of the overall transformation results and the execution time of our transformation technique. The first column shows the name of the model, the second and the third columns show the number of the BPMN elements and the security annotations in the SecBPMN2 model, respectively. The fourth and fifth column show the number of UML elements and the security annotations in the corresponding UMLsec model. The last column shows the time needed by our technique for transforming the corresponding SecBPMN2 model to the UMLsec model. The time does not include the initialization time for our transformation. The initialization time for transforming each model is 17402 [ms]. The tests were performed on a computer with a 2.2 Ghz processor and 8 GB of memory.

Based on the results in Table 3.5, the performance of the proposed transformation technique seems adequate for practical use on models of the considered size. For example, our technique requires in total around 70 seconds for transforming both

the “External services 1” and “External services 2” SecBPMN2 models to UMLsec models that consists of 818 UML elements and 150 security annotations. Taking 70 seconds for the transformation represents a very low time comparing with the needed time needed for a manual transformation process, which may take hours.

### 3.7 Discussion and Future Work

In Section 3.6, we demonstrate through a case study of an air traffic management system how our proposed framework in this chapter allows security-specific data protection requirements to be managed and traced throughout the different design phases of the system development process.

*Integrated management* of data protection requirements across different phases is established via our automated transformation that integrates the involved languages. Our proposed framework for integrating SecBPMN2 and UMLsec models focuses on a subset of SecBPMN2 of those security-specific data-protection annotations with an equivalent on the architectural level (i.e., class and deployment diagram). We did not consider the transformation of BPMN models to UML behavioral models such as activity and sequence diagrams because the BPMN model and these UML behavior diagrams provide the same level of information and almost share the same semantics. For example, generating UML activity diagrams from BPMN models would be a straightforward transformation.

As business analysts, we focused on expressing the organizational requirements and data protection requirements. As system engineers, we could concentrate on designing and implementing the data protection-aware software architecture, without having to learn the specifics of business process modeling. From Table 3.5, we can infer that the automated transformation saved us from re-implementing a large number of security annotations, which would have been daunting and error-prone.

*Traceability* is established via the trace models generated by our transformation rules. The trace models contain traceability links from source to target model elements. As one possible use for these links, an interested stakeholder could check whether a UMLsec data-protection policy is in place for each data-protection annotation in the SecBPMN2 model. Therefore, trace models are a promising means to increase trust in the produced models. Altogether, our case study highlights the potential benefits of integrating SecBPMN2 and UMLsec via our framework.



### 3.7.1 Threats to validity and limitations

Our study for the applicability of the proposed framework in this chapter is subject to a number of threats to *external* and *internal* validity. Two main threats to *external validity* are: First, we emulated the involved stakeholders in the process of our proposed framework, rather than involving actual ones. Second, we focus on a single case study instead of applying the process of our proposed framework to a broader selection of cases. While a key benefit of our process is its reliance on notations familiar to the involved users, an empirical usefulness study is left to future work. We also aim to apply our process to a broader selection of cases.

A threat to *internal validity* is the lack of a formal validation of the correctness of our transformation. Our transformation benefits from the capability to check the output models against UMLsec data-protection policies, which, however, does not ensure that the intention of the business analyst is accurately reflected. Due to the large variability of SecBPMN2 models, our transformation for integrating SecBPMN2 and UMLsec models could be affected by errors. However, by applying our proposed 255 transformation rules to the SecBPMN2 model of the air traffic management system case study, we did not report errors. Although we studied this on a single case study, we assume that the same result will be reported when applying our transformation rules to other cases because the used SecBPMN2 model in our case study is large and it consists of several corner cases.

### 3.7.2 Future Work

A possible future work is to extend our proposed framework to legacy situations, in which the UML design models are already given, rather than developed from scratch. Our mapping between SecBPMN2 and UMLsec security concepts can provide a foundation for addressing such legacy scenarios.

Systems are continuously evolving due to changes at the organizational level or at the technical level [44]. Any change at the organizational or the technical level of a system may be accompanied by changes in data protection requirements. Therefore, a possibility for future work is to extend our framework in order to support traceability of data protection requirements in case of changes to a business process model of a system or its architecture models.

Other possibilities for future work are: First, to generalize our approach to the remaining SecBPMN2 annotation types by mapping them to various UMLsec diagrams. Second, to extend the framework to incorporate the semantics of both languages, which would allow us to formally guarantee the correctness of our transformation. Third, to perform a user study to study the usability of our framework.

## 3.8 Related Work

There has been a significant amount of model-based analysis approaches that aim to reason about security- and privacy-specific data protection requirements in the early phase of system design models. An overview can be found in [78], which reviews existing approaches for security analysis of model-based object-oriented software designs, and identifies ways in which these approaches can be improved and made more rigorous. Some research addresses linking the model to the code level within model-based security engineering [85, 86, 102]. Other work addresses the model-based use of security-specific patterns [66, 97]. Further research makes use of aspect-oriented modeling for model-based security [51]. [56] proposes an approach for model-based security verification.

The specification of a software architecture model based on business processes represented by BPMN or UML activity models has been subject to previous works [31, 72, 117, 120]. However, except for the following ones, none of these works considered data protection aspects such as security during the transformation. The fact that different modeling languages are extended to cover data protection aspects such as security and privacy, attracted many researchers to study the traceability of these aspects via the integration between these languages. Approaches in this direction can be classified into automated and manual approaches. In the following, we highlight the differences between our work in this chapter and these studies.

### 3.8.1 Automated transformation

In [120], the authors used QVT specification<sup>12</sup> (Query/View/Transformation) to specify a mapping from UML activity models annotated with security-specific annotations to UML class diagrams. The mapping is of a one-to-one kind, where each security requirement is transformed into a class or annotation with the same name. Hence, as stated by the authors, the resulting security policies remain at a high level of abstraction, representing the view of business analysts. As a result, security experts are needed for manually refining the security annotations to a technical security aspects. In contrast, our framework:

- (i) supports the transformation to two UML structural diagram types in which data protection requirements at different design-level views of the system are captured, and
- (ii) does not require the involvement of data protection experts, since high-level data protection needs are transformed to verifiable data protection policies that encapsulate data protection knowledge. However, the presence of a data protection expert is still preferable.

<sup>12</sup><https://www.omg.org/spec/QVT/1.3/PDF> (accessed: 08/12/2019).

Conversely, the authors in [55] proposed a method to systematically develop UMLsec design models from requirements models annotated with security-specific requirements. The requirements models in their approach were UML models with *secure problem frame* annotations. While this approach makes the formal validation of UMLsec applicable, it is not tailored to business analysts, one of the main stakeholder types in business process analysis of system-to-be. In [41], the authors presented the transformation from misuse-cases [137] to mal-activity models [136]. Different from UMLsec, mal-activity models are not verifiable against the specified activities and represents the view of business analysts.

### 3.8.2 Manual transformation

Different from our work, other works have provided informal guidelines for addressing data protection requirements while transitioning between stages in the development process.

For example, the authors in [94] proposed an approach for designing secure software system models, using UMLsec [62], starting from organizational security requirements, using Secure Tropos [93]. For the same purpose, the authors in [79] proposed an approach to generate a UML class diagram represented by SecureUML [81] from organizational requirements represented by the KAOS [147] method. In [58], the authors connect UMLsec security-specific policies with elicited requirements based on heuristics. This approach takes as input a set of requirements to predict a suitable UMLsec security policy for each security requirement. In these approaches, the software models must still be built and annotated with security annotations manually, which is a non-trivial and error-prone task.

## 3.9 Conclusion

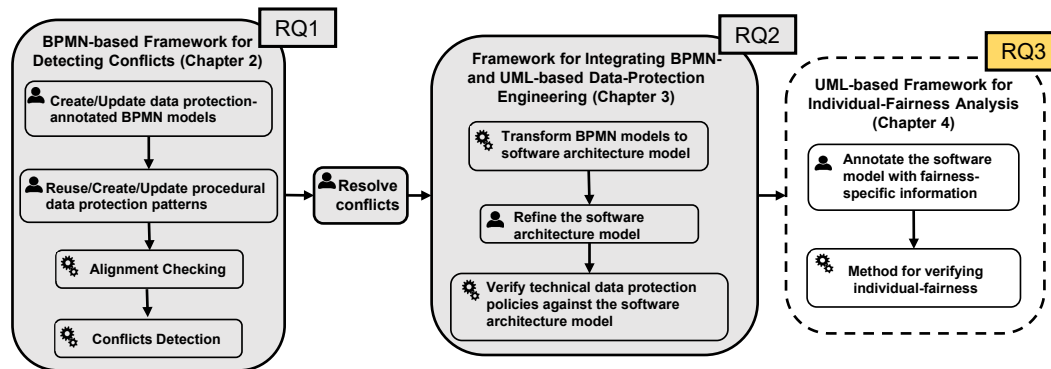
In this chapter, we present a framework for tracing high-level data protection requirements to verifiable technical data protection policies and, as a result, for bridging the conceptual gap between them. The main benefit is the management of data protection requirements from the views of the involved expert stakeholders, in particular, business analysts and systems engineers, in an integrated manner. To this end, we integrate the two well-known model-based security approaches SecBPMN2 and UMLsec via model transformation. We illustrated the aforementioned benefits in a case study, in which our framework was suitable to render the early development stages of an air traffic management system less error-prone and more systematic. Our results are not restricted to any particular data protection-oriented extension of BPMN or UML, but can be applied to other ways of using BPMN and UML to address data protection requirements.



## Chapter 4

# Individual Fairness Analysis based on Software Design Models

This chapter presents a sub-framework of the proposed MoPrivFair (*Model-based Privacy & Fairness*) methodology in this thesis. An overview of the MoPrivFair methodology is provided in Section 1.2 in the first chapter of this thesis.



**Figure 4.1:** The highlighted part (dashed lines) denotes how Chapter 4 contributes to the overall workflow of the MoPrivFair methodology.

Decision-making software is prone to undesired discrimination against individuals based on protected characteristics such as gender or ethnicity. Considering the individual fairness of software after implementing it raises substantial difficulties in the identification and explanation of the discriminatory behavior of the software.

In this chapter, we propose a semi-automated model-based framework that supports the analysis for individual fairness already in the software design phase, thereby avoiding the possibility of discrimination from the onset of the software

development process<sup>1</sup>. Specifically, we proposed a UML profile called UMLfair to annotate a software model with fairness-specific information. Using UMLfair, we enabled the generation of temporal logic claims, which can be verified against the model to uncover discriminatory behavior. We have applied our framework to three case studies featuring a school management system, a delivery management system and a loan management system.

## 4.1 Introduction

Automated decision-making software became responsible for sensitive decisions with far-reaching societal impact in many areas of our lives. However, the risk that a falsely developed decision-making software may lead to unlawful discrimination against persons, illegally exploiting their *protected characteristics*, has raised public and legal awareness on software *fairness* [140, 153]. For example, Article 22, Paragraph 4 of the European General Data Protection Regulation (GDPR, [2]) forbids decisions based on special categories of data as defined in Article 9 of the GDPR, such as ethnicity, religion, and gender. These data are known in the state of the art as protected characteristics [49].

Although there is no single definition of what fairness means [149], a distinguished type of fairness is called *individual fairness*. A decision-making software provides the individual fairness if it produces the same decision for every two individuals whose data that are given as input to the decision-making software are identical except for the protected characteristics [37]. One should note that not every discrimination concerning a protected characteristic is forbidden. For example, although the *age* is a protected characteristic, it might be legal that insurance companies differentiate based on the *age* for various life insurance rates. Data whose effects on sensitive decisions can be justified are called *explanatory data* [145]. Details description of fairness-related concepts is provided in Section 2.2.3 of this thesis. An overview of discrimination from the legal perspective is provided in Section 2.2.4.

Only avoiding protected characteristics in a decision-making software does not prevent discrimination. Due to data correlations, other data may act as proxies for protected characteristics, thereby causing *indirect discrimination* [49]. For example, in 2016, a delivery software by Amazon<sup>2</sup> excluded some neighborhoods with

---

<sup>1</sup>This chapter shares material with the paper "Analyzing Individual Fairness based On System Design Models" [113], the SE/SWM'19 "Explaining Algorithmic Decisions with respect to Fairness" [112] and the FairWare@ICSE'18 workshop paper "Model-Based Discrimination Analysis: A Position Paper" [109].

<sup>2</sup>Detailed description for the discriminatory behavior of Amazon's software is available at <https://www.bloomberg.com/graphics/2016-amazon-same-day/> (accessed: 05/12/2019).

African American communities in the United States from being able to participate in a free-delivery service, although the software did not explicitly use the ethnicity of the customers for making the decisions. There are two main explanations for data correlations [112]:

- (i) **societal fact.** For instance, it turns out that the discriminatory behavior of Amazon's delivery software results from the use of the zip-code which is highly correlated with the *ethnicity* of local area residents in the USA.
- (ii) **data flow.** The actual input of a decision-making software may contain data resulting from processed protected characteristics. For example, assume for a moment that Amazon uses *age* to decide about a *prime membership* application. If the delivery decision-making software depends on the prime membership status, the delivery software indirectly discriminates against age.

#### 4.1.1 Problem statement and research question

The data flow issue is typical for the problem that Dwork et al. [36] observed: "*fairness is not a property of software in isolation, fairness is a property of the software composed into its system*". However, existing fairness analysis approaches do not analyze the software *ex-ante* but *ex-post*, namely during the testing phase of software development life cycle (e.g, [29, 49]) or at the run-time of the software (e.g., [7, 8]). To reduce difficulties in the later stages of software development, it is important to deal with fairness from the early stages of software development. According to Brun et al. [18] "*as with software security and quality, [...], fairness needs to be a first-class entity in the software engineering process*". Considering support for individual fairness at the software design phase, a *software model-based* approach that permits detecting discriminations during the design of software models is missing.

The *software model-based development* is a promising direction in the software engineering field and it has been widely used in the literature for reasoning about critical issues such as security and privacy [5, 62, 81]. In many application domains (including finance, health, automotive etc) there exist regulations which require detailed documentation of the software to support the needed certification, and this requirement can be fulfilled using software models. Modeling languages such as the Unified Modeling Language (UML [3]) allow for designing software models in a high-level specification in which its components with their inputs and outputs can be analyzed. Although the use of software models in practice varies between different software domains, they can be a key enabler for important tasks of high business value, such as the fairness analysis. The need for engineering fairness-aware software based on software models has been motivated by Brun et al. [18]. However, the authors did not provide an approach that supports their idea.

In this chapter, we address the following research question:

**RQ3.** *How to detect undesired discriminatory behavior, that violates individual fairness, by analyzing software design models?*

Our objective is to support software developers/analysts with an approach that permits uncovering discrimination against individuals during the software design time in order to reduce the effort needed for uncovering discrimination in the later stages of development.

### 4.1.2 Contribution

To address the research question RQ3, we present a *semi-automated* framework that supports the analysis of UML-based software designs with regard to individual fairness. The analysis of individual fairness in our framework is established by generating temporal logic claims, whose verification against the targeted software model enables reporting on the individual fairness of the software. Specifically, our framework includes the following contributions: (1) a UML profile *UMLfair* for annotating UML models with fairness-specific information; (2) a *method for verifying individual fairness* of annotated UML models; and (3) *three case studies*, that show the applicability of our method.

Our framework is novel in the sense that it is the first that permits fairness analysis during the design phase of software models. It is worth noting that, our framework is not concerned with analyzing the individual fairness of a software that uses a machine learning algorithm. Fairness is not a property that should be preserved by machine learning-based software only. Even IF-THEN-ELSE rule-based decision software can be unfair. Algorithmic systems without machine learning still make up a vast percentage of real-life software, especially in new systems where a sufficient amount of relevant training data is not available. For example, many of today's insurance systems still use preplanned IF-THEN-ELSE rules, instead of machine-learning algorithms. Hence, the fact that previous studies focus on machine learning-based software does not mean to underestimate fairness in non-machine learning based software.

The remaining sections of this chapter are organized as follows. Section 4.2 provides the running example that will be used throughout the chapter to explain our contribution. Section 4.3 presents our proposed framework. Section 4.4 presents our proposed UMLfair profile. Section 4.5 presents the generation of temporal logic claims. Section 4.6 describes how we use model checking to verify the claims. In Section 4.7 we describe how the verification results are mapped to discrimination issues. Section 4.8 presents an optional to support uncovering proxies of protected characteristics based on a database when expert knowledge about proxies of protected characteristics is not available. Section 4.9 presents the tool support. Section



4.10 shows the applicability of our proposed framework based on three case studies. Section 4.11 presents the limitations and future work. Section 4.12 and Section 4.13 discuss related work and conclude, respectively.

## 4.2 Running Example

Consider a school interested in leveraging automatic decision-making to decide about different kinds of applications, that can be submitted by its students, such as: (i) application for a basketball team membership, or (ii) application for a scholarship. Deciding about a scholarship application is sensitive. In particular, the school policies disallow discriminating among applicants to a scholarship based on their protected characteristics such as *gender* and *physical health status*.

**The software model.** Figure 4.2 shows an excerpt from the school’s software model. It focuses on the parts that are related to the basketball and the scholarship applications. Figure 4.2(a) is a UML *class diagram* [3]. A class diagram describes the structure of software by showing its classes, attributes, operations, and the relationships among classes. A central class in the class diagram is the “StudentProfile”.

The “gender” is a boolean data attribute. It is “true” if the gender is *male* and “false” if the gender is female. The “height” is an integer attribute. It shows the height of a student in *cm*. The “outstandingEducation” attribute is “true” if a student has a very good education level, else “false”. The “normalWeight” is “true” in case a student is not overweight, else “false”. The attributes that are denoted with “/” symbol are *derived attributes*. A derived attribute is a data item that is not given by the user but resulted from processing other user’s data. The derived attributes in the “StudentProfile” class are described as follows: The “bbMembership” is a boolean attribute, that is “true” if a student is a member in the basketball team, else “false”. The “membershipNum” is an integer attribute, that shows the number of memberships that a student owns. The “scholarshipStatus” is “true” if a student is admitted to a scholarship, else “false”.

An example of an operation is the “verifyScholarApp()”. The «Signal»-annotated “done” represents a received signal that an object of the “StudentProfile” class may receive during its life-cycle. An object reacts to the receiving of a signal according to the specified behavior by its class ([3], p.169). Figure 4.2(b) is a UML *state machine diagram* [3] that describes the behavior of the “StudentProfile” class. UML state machines describe the sequences of states that an entity, such as an object or a component, can go through in response to events such as call operations or received signals, together with its responding actions. A state machine contains states and transitions. A *state* is “a situation in the execution of a state machine behavior during which some invariant condition holds” ([3], p.308). States are denoted by boxes. Examples of states in Figure 4.2(b) are “Idle” and “VerifyBbApp”.

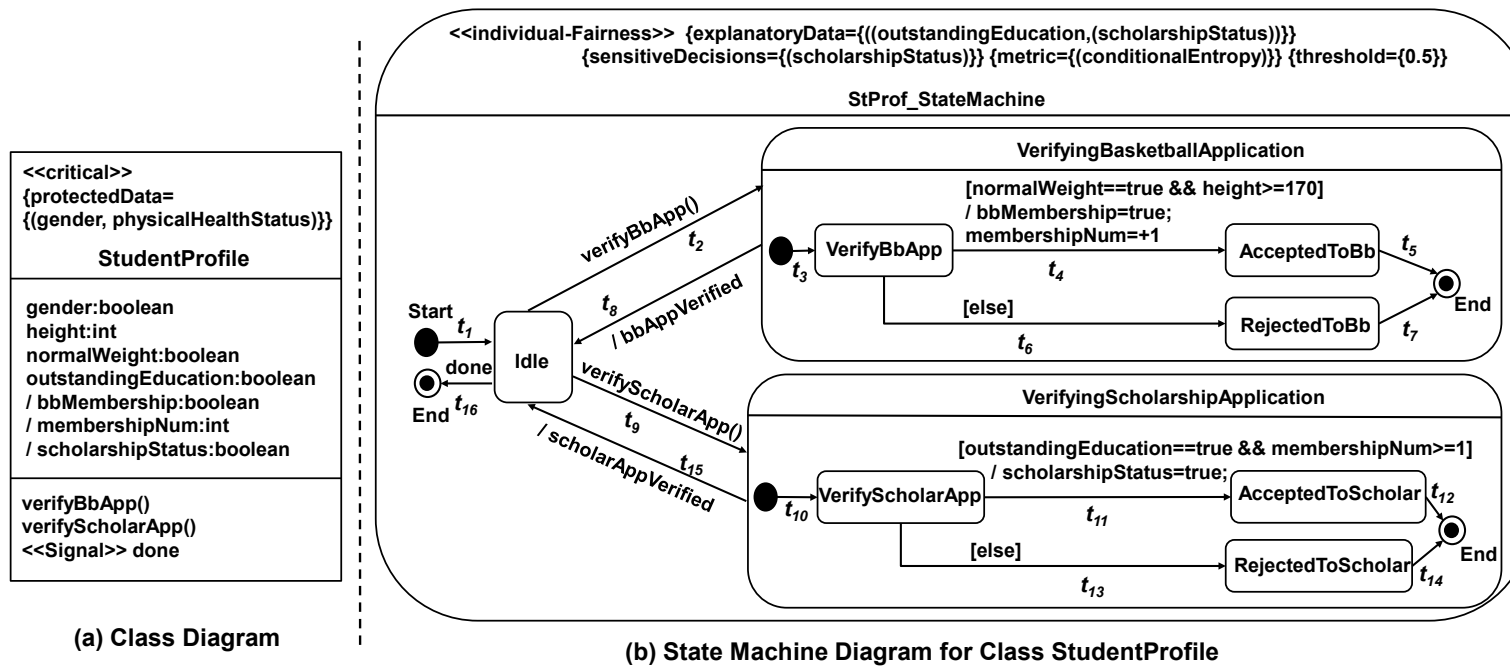


Figure 4.2: Example Model: Excerpt from the class and the state machine diagrams of the school software.

Transitions are denoted by arrows between the states. A transition with label  $e[g]/a$  indicates that an object in the first state will perform the action  $a$  and enter the target state when the event  $e$  occurs and the guard condition  $g$  is satisfied ([3], p.314). The event can be either a call operation or a received signal. The guard represents a logical formula, involving, for example, equality and inequality between data expressions and logical connectives such as conjunctions. The action can be a data assignment for an attribute, a call for an operation or a send signal. For example, in Figure 4.2(b), if an object is in the “verifyScholarApp” state and the guard condition “[outstandingEducation==true && membershipNum>=1]” is true, first the transition “ $t_1$ ” will be fired, then the “scholarshipStatus” will be set to true and the “AcceptedToScholar” state will be entered.

**Proxies for protected characteristics.** The software design uses personal information about students to decide about applications. In this example, experts identify that “outstandingEducation” and “height” can act as proxies for “gender”, and “normalWeight” can act as a proxy for “physicalHealthStatus”.<sup>3</sup>

**The challenge.** Given the UML model in Figure 4.2 and the definition of proxies, our question is: Does the behavior of the “StProf\_StateMachine” violate individual fairness with respect to for example, “gender” when deciding about the “scholarshipStatus”? If it does, then how could this happen? In the following sections, we will show that it is possible to discriminate between two scholarship applicants whose data are identical except for the “height”, which is a proxy of the “gender”.

### 4.3 Framework for Analyzing Individual Fairness

Manually analyzing a UML software model to uncover violations of individual fairness is an error-prone task because information about how data is propagated in the software model and how data items depend on each other are hidden and distributed in multiple diagrams. In addition, the software model alone is not sufficient for identifying the proxies of protected characteristics. A data item is a proxy for a protected characteristic if the two are strongly correlated with respect to a threshold [29]. For detecting indirect discrimination, it is necessary to identify proxies for protected characteristics. This information can be provided by an expert, who takes this information from laws, compliance documents, and experience. In systems where personal data is available or can reliably be generated, this data can be used to identify proxies for protected characteristics by performing statistical analysis, using metrics such as the conditional probability [145].

---

<sup>3</sup>This is an example use case and the proxies should be adapted to be used in other contexts.

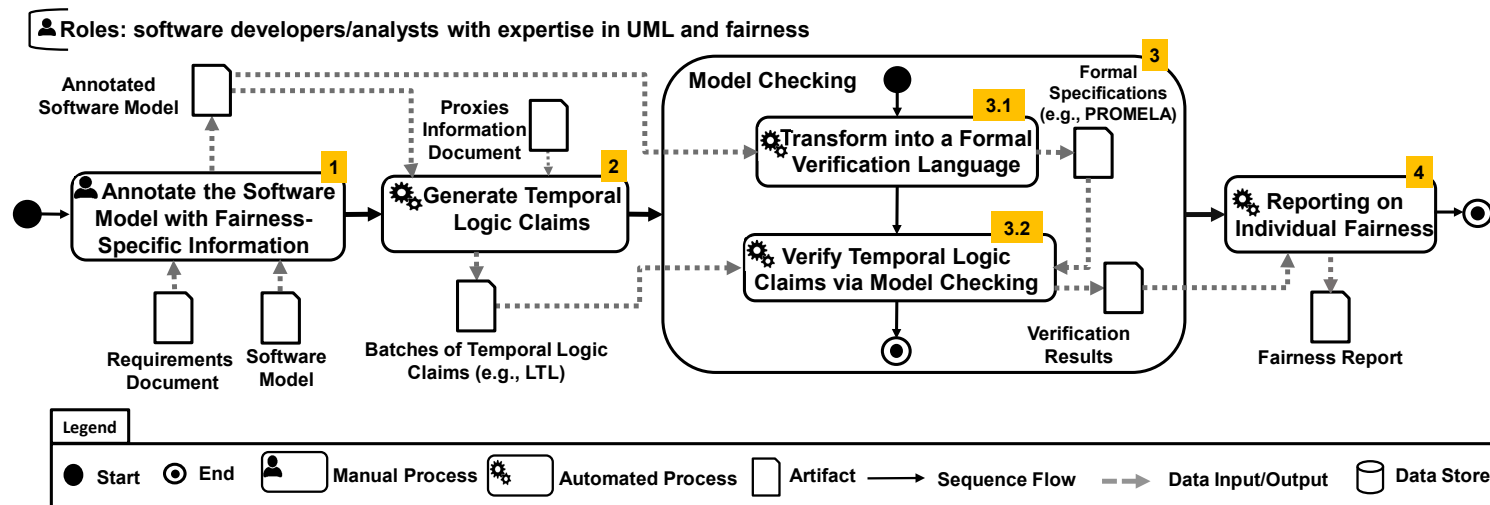


Figure 4.3: The semi-automated, model-based process for analyzing individual fairness

To address the challenges above, we propose a semi-automated framework that applies *formal model checking* techniques to a given UML model in order to support the analysis for individual fairness<sup>4</sup>. An overview of our framework is shown in Figure 4.3. Formal verification techniques are best applied in the early stages of software design when the cost is relatively low and the benefits are high [59]. The goal of model checkers such as NuSMV<sup>5</sup> and SPIN<sup>6</sup> is to determine whether given *temporal logic* claims are violated by exploring the entire state space of a model.

Temporal logic is a symbolic logic, which permits specifying claims (i.e., propositions) that have truth values. The claims can be described using an abstract concept of time [59]. For example, consider the following temporal logic claim:  $(gender == female \rightarrow \Diamond scholarshipStatus == true)$ . This claim expresses that the “scholarshipStatus” will *eventually* (i.e., has to hold at some point in any execution path) be true given the “gender” is female. The model type in this chapter is a state machine, which uses an abstract concept of time. State machines are a way to express decision-making algorithms that use if-then-else rules. We are interested in checking whether a protected characteristic will eventually directly or indirectly influence the decision. Therefore, with a collection of temporal logic claims, we can uncover whether specific values (or value intervals) of protected characteristics or proxies later affect the *sensitive decision*. In our work, a sensitive decision is to set a specific data attribute value or to call an operation. These sensitive decisions should not discriminate on the basis of protected characteristics.

In the following, we provide an overview of our proposed framework in Figure 4.3. The overview covers information about the involved roles in the process of our framework, the input and output of our framework, and a brief description for the phases of the framework. Latter each phase will be explained in more details through the use of the running example in Figure 4.2.

**Involved Roles.** The process of our framework can be executed by software *developers/analysts* who are responsible for designing the software model and annotating it with details such as protected characteristics, sensitive decisions in the software, explanatory data that their effects on decisions can be justified (e.g., admitting applicants to a basketball team based on their physical health status is allowed), and correlation metrics and the thresholds that have to be used for identifying proxies of protected characteristics. The involved roles should have expertise in UML and

---

<sup>4</sup>Worth noting that term fairness was already used as a constraint in the formal model checking area. Different from what fairness means in our work, in the model checking the fairness constraint is imposed on (the scheduler of) the system that it fairly selects the process to be executed next. For example, if an activity is infinitely often enabled then it has to be executed infinitely often [107].

<sup>5</sup>The NuSMV model checker and detailed account on its usage are available online at <http://nusmv.fbk.eu/> (accessed: 05/12/2019).

<sup>6</sup>The SPIN model checker and information on its usage are available online at <http://spinroot.com/spin/whatispin.html> (accessed: 05/12/2019).

fairness. Still, to ensure correct use for the proposed framework, the involved roles should be trained on the usage of our framework.

**Inputs.** As shown in Figure 4.3, three inputs are considered:

- (i) a *Software Model* in UML. Structural aspects of software are specified by class diagrams. Behavioral aspects are specified by state machine diagrams. An overview of the class and state machine diagrams is provided in Section 4.2.
- (ii) a *Requirements Document*. During the requirements elicitation, domain experts, based on organizational policies and equality acts such as the UK Equality Act<sup>7</sup> 2010, can identify the protected characteristics and the purposes in which discrimination on the basis of a protected characteristic is allowed.
- (iii) a *Proxies Information Document*. Domain experts can provide information about the proxies of protected characteristics. The provided proxies will help in uncovering indirect discrimination against protected characteristics. Optionally, in the absence of knowledge about proxies of protected characteristics, historical personal data can be used to identify proxies. Therefore, the domain experts will define metrics, thresholds, and a database of historical personal data in the requirements document. Statistical analysis, using the correlation metric and the threshold, will be used to identify proxies.

**Outputs.** as shown in Figure 4.3, the final output of the process is a *Fairness Report*. The report summarizes the overall analyzes result. For each detected violation of individual fairness, the report shows:

- (i) the effected sensitive decision by the violation of individual fairness,
- (ii) the source of the individual fairness violation (i.e., the data item that caused the discrimination, which might be a protected characteristic or a proxy of a protected characteristic),
- (iii) all the *side effects* of the source of the violation on the sensitive decision. The side effects here refers to all the possible changes on the sensitive decision with respect to all possible values that can be assigned to the data attribute, which represents the source of the violation, and
- (iv) references to counterexamples that explain how some of the side effects on the sensitive decision occurred. A counterexample can also help in locating the source of discrimination in the model.

---

<sup>7</sup>The Equality Act 2010 is an Act of Parliament of the United Kingdom and it aims at protecting people from discrimination. The Act is available online at [http://www.legislation.gov.uk/ukpga/2010/15/pdfs/ukpga\\_20100015\\_en.pdf](http://www.legislation.gov.uk/ukpga/2010/15/pdfs/ukpga_20100015_en.pdf) (accessed:05/12/2019).

In the following, we explain the phases of our framework. The numbers in Figure 4.3 represent the phases, as follows:

**Phase 1.** Software developers manually design a UML model and annotate it with fairness-specific information, based on the *requirements document*. Hence, we propose a UML profile called *UMLfair* to allow annotating a UML model with details such as the protected characteristics, and the explanatory data. The output, as shown in Figure 4.3, is an *annotated software model*. A detailed description of the UMLfair profile is provided in Section 4.4.

**Phase 2.** Batches of temporal logic claims are generated with respect to the targeted decision-making state machine in the software model. Each batch contains claims analyzing the effects of a protected characteristic or a proxy of it on the behavior of a sensitive decision in the software model. The inputs to this phase, as shown in Figure 4.3, are an *annotated software model* and *proxies information document*. The output is a set of *batches of temporal logic claims*.

Although different kinds of temporal logic are available such as Linear Temporal Logic (LTL) [104], Computation Logic Tree (CTL) [26], and  $\mu$ -calculus [71], they all share the basic capabilities and semantics we use in our work. Hence, our proposed framework is not restricted to a specific kind of temporal logic. To explain our contributions, in what follows, we use LTL to express claims. A detailed description on the generation of temporal logic claims is provided in Section 4.5.

**Phase 3.** The generated claims in *phase 2* are automatically verified against the software model. This phase is divided into two sub-phases:

- *Phase 3.1.* The annotated software model is transformed into a formal verification language, using a transformation tool such as Hugo/RT<sup>8</sup> UML-VT<sup>9</sup>, or SMUML<sup>10</sup>. As shown in Figure 4.3, the input to this phase is an *annotated software model* while the output is a *formal specifications* file that the model checker accepts as an input. For example, the model checker SPIN accepts a software model described by the Process Meta-Language (PROMELA) language while the model checker NuSMV accepts a software model described by the Symbolic Model Verifier (SMV) language.

---

<sup>8</sup>Hugo/RT is available online at <https://www.informatik.uni-augsburg.de/en/chairs/swt/sse/hugort/> (accessed: 05/112/2019).

<sup>9</sup>UML-VT is available online at <https://www.cs.umd.edu/~rance/projects/uml-vt/> (accessed: 02/01/2020).

<sup>10</sup>SMUML is available online at <http://www.tcs.hut.fi/Research/Logic/SMUML.shtml> (accessed: 02/01/2020).

- *Phase 3.2.* The claims from *phase 2* are automatically verified against the formal specifications of the UML model using a model checking technique. If a claim is violated, the model checker will generate a counterexample in the form of a trace of events that explains how the violation occurred. As shown in Figure 4.3, the inputs to this phase are *batches of claims* and a *formal specifications* file while the output is a report, that contains the verification results of the claims. A detailed description of the verification of claims against the software model is provided in Section 4.6.

**Phase 4.** The claims' verification results of each batch of claims are analyzed to check whether the software model preserves individual fairness. As shown in Figure 4.3, the input to this phase is the *claims' verification results* while the output is a *fairness report*. The *fairness report* is introduced earlier and its contents describe the detected violations for individual fairness. A detailed description on how the verification results of the temporal logic claims permit reasoning on individual fairness is provided in Section 4.5.

## 4.4 Annotating the UML Model with Fairness Information

In this section, we describe *phase1* of the proposed framework in Figure 4.3. In this phase, the UML model is annotated with fairness information. We propose a UML profile called *UMLfair*. *UMLfair* is an extension of UML and it permits the involved stakeholders to specify fairness information in a UML model. We propose the *UMLfair* as a sub-profile for the UML security extension profile *UMLsec* [62]. Details on the *UMLsec* profile are provided in the previous chapter in Section 3.2.2. In what follows, we describe the *«stereotypes»* and the *{tags}* of the *UMLfair* profile. The *«stereotypes»* and the *{tags}* of *UMLfair* are provided in Table 4.1.

**Table 4.1:** The *UMLfair* profile

Stereotype	Tag
«critical»	{protectedData={{(string[1...*])}} //required
«individual-Fairness»	{explanatoryData={{(string[1],string[1...*])}} //optional
	{sensitiveDecisions={{(string[1...*])}} //required
	{metric={string[0]}} //optional
	{threshold={double[0]}} //optional

Data in UML, in this context, is specified as attributes in UML class diagrams. For specifying protected characteristics, our profile has a stereotype called *«critical»*. We reused this stereotype from the *UMLsec* profile [62]. In *UMLsec* this stereotype annotates classes that may contain confidential data. In our work, we use this



stereotype to annotate classes that may contain protected characteristics or proxies to protected characteristics. As shown in Table 4.1, we extended this stereotype with a *{protectedData}* tag to permit defining the protected characteristics with respect to a class annotated *«critical»*. For example, in Figure 4.2(a), the *{protectedData}* tag of the annotation *«critical»* on class “StudentProfile” specifies the “gender” and the “physicalHealthStatus” as protected characteristics.

The *«individual-Fairness»* stereotype annotates a state machine, that describes the behavior of a class, that is annotated *«critical»*. As provided in Table 4.1, this stereotype has several tags:

- The *{sensitiveDecisions}*. This tag is required and it allows defining that setting values of specific data attributes or calling operation events are decisions that should not discriminate on the basis of protected characteristics or proxies of protected characteristics. For example, the state machine annotated with *«individual-Fairness»* in Figure 4.2(b), specifies the “scholarshipStatus” data attribute as a sensitive decision.
- The *{explanatoryData}* tag. This is an optional tag and it allows defining that it is acceptable to use these data attributes to differentiate in sensitive decisions. For example, in Figure 4.2(b), the “outstandingEducation” is defined as an explanatory data attribute for the “scholarshipStatus”.
- The *{metric}* tag. This is an optional tag and it allows specifying the correlation metric to use for measuring correlations. Various correlation metrics have been used in the literature to identify proxies such as the conditional probability, information gain, and others [145]. For a flexibility issue, the specification of the *{metric}* tag is not restricted to a specific set of metrics. The *{metric}* tag can take any string as an input as long it has a mapping to a function in the implementation.
- The *{threshold}* tag. This is an optional tag and it allows to specify the numeric threshold that is needed to determine based on the correlation results whether two pieces of data are strongly correlated so one can act as a proxy for another.

The *{metric}* and *{threshold}* can be used to identify proxies of protected characteristics based on a database of historical data in case the information about proxies is not available. As explained in Section 4.3, the correlation metric and the threshold that have to be used for identifying proxies of protected characteristics can be retrieved from the *requirements document*. In our running example, information about proxies is available. A detailed account on the generation of proxies based on a database is provided in Section 4.8.

## 4.5 Generating Temporal Logic Claims

In this section, we describe *phase2* of our framework in Figure 4.3. We first show through an example how we analyze individual fairness of software models using temporal logic and then we propose the generalization of our proposed technique.

From a *technical view*, a software model preserves individual fairness if it produces the same decision for any two feasible traces of events, whose *configuration data* (i.e., values used in guard conditions) are identical except for the protected characteristics or their proxies. This notion of individual fairness is derived from the *no down-flow* policy in the field of information security [33]. Intuitively, no down-flow means, that a public output of a system does not depend on secure inputs. Generating all possible traces of events is a difficult task as the number of possible traces may be infinitely large. Therefore, we introduce the use of *temporal logic* to formulate the side effects of protected characteristics and their proxies on sensitive decisions as claims, while leaving the process of exploring all the possible traces of events in a software model to a model checking technique.

**Example 1.** In our running example, the “height” is identified as a proxy of the “gender”. To check whether the “StProf\_StateMachine” in Fig. 4.2 preserves individual fairness with respect to the “height” when deciding about the “scholarshipStatus”, one needs to: First, express all the possible effects of the “height” on the “scholarshipStatus” in pairs of claims, where each pair, in isolation, considers claims about one possible side effect. Second, verify if the claims of exactly one pair are satisfied at some point in any execution path (i.e., eventually true) while the claims of the other pairs are violated (i.e., eventually false). That is the “scholarshipStatus” always has the same value independent of the “height”.

Based on the “(height>=170)” condition in Fig. 4.2(b), there might be two side effects for the “height” on the value of the “scholarshipStatus”. These side effects are expressed as LTL claims in the pairs namely, *pair1* and *pair2*. The claims of *pair1* check whether the “scholarshipStatus” is true, independently of the “height”, while the claims of *pair2* check whether it is false, independently of the “height”.

$$\left( \begin{array}{l} (\text{height} \geq 170 \rightarrow \diamond \text{scholarshipStatus} == \text{true}, \\ \neg(\text{height} \geq 170) \rightarrow \diamond \text{scholarshipStatus} == \text{true} \end{array} \right) \quad (\text{pair1})$$

$$\left( \begin{array}{l} (\text{height} \geq 170 \rightarrow \diamond \text{scholarshipStatus} == \text{false}, \\ \neg(\text{height} \geq 170) \rightarrow \diamond \text{scholarshipStatus} == \text{false} \end{array} \right) \quad (\text{pair2})$$

Specifically, the first claim of *pair1* expresses that if the “height” is greater than or equal to “170”, the “scholarshipStatus” will eventually  $\diamond$  (i.e., at some point in any execution path) equal true. The second claim of *pair1* expresses that if the “height” is less than “170”, the “scholarshipStatus” will eventually equal true. The first and second claims of *pair2* are similar to the first and the second claims of *pair1*, respectively, but with claiming that the “scholarshipStatus” will eventually equal false instead of true. Since, in the end, a value has to be assigned to the “scholarshipStatus”, it is impossible that all the claims of *pair1* and *pair2* will be violated. We have individual fairness when the claims of *pair1* are satisfied while the claims of *pair2* are violated or vice versa.

In our work, we use the eventually  $\diamond$  operator instead of the *globally*  $\square$  operator to consider the effect of other data that are not part of the claims on the final decision. Also, to declare that the possible discrimination can happen at any latter moment in the software behavior. In other words, our analysis consider fairness to be violated already if it would be violated at some intermediate stage of the model execution. Different from the eventually operator, the *globally*  $\square$  operator does not take the effect of other data in the system on the final decision because it requires a condition to hold in any execution path until the end of the path execution. The following two claims clarify our argument:

$$height \geq 170 \rightarrow \square scholarshipStatus == true \quad (1)$$

$$height \geq 170 \rightarrow \diamond scholarshipStatus == true \quad (2)$$

By verifying the claims (1) and (2) against our running example, claim (1) is violated because based on the specifications of the state machine in Figure 4.2(b) the “scholarshipStatus” does not depend only on the “height”. In addition to the “height”, the “scholarshipStatus” depends on the “outstandingEducation” and the “normalWeight”. Therefore, given the height is greater than “170” does not mean that the “scholarshipStatus” will *always* be true at the end of the execution. In contrast, claim (2) is satisfied, meaning that if the “height” is greater than “170”, there is the possibility that the “scholarshipStatus” will equal “true” sometimes. Based on the state machine in Figure 4.2(b), this can happen when the “outstandingEducation” equals “true” and the “normalWeight” equals “true”.

**Generating batches of claims.** The pairs *pair1* and *pair2* in *Example 1* together represent a batch that analyzes the effects of the “height” on the “scholarshipStatus”. To check whether a state machine *sm* preserves the individual fairness, for each sensitive decision *s* in *sm* we generate a batch of claims for (a) each protected characteristic (in the *{protectedData}* tag) and (b) for each proxy of those.

To reduce the number of the claims to be generated, the generation of batches of claims with respect to a sensitive decision *s* has to consider every atomic guard

condition  $g$  whose data attribute:

- First, is not defined as an explanatory piece of data with respect to  $s$  in the  $\{explanatoryData\}$  tag of  $sm$ . This is to avoid generating claims with respect to data that are acceptable to differentiate based on them among individuals.
- Second, is defined as protected characteristic in the  $\{protectedData\}$  tag or is a proxy of a protected characteristic, that does not belong to the explanatory data of  $s$ . This is to ensure that only claims are generated with respect to protected characteristics or their proxies.
- Third, is not defined as a derived data attribute. This is to avoid generating claims with respect to generated proxies. For example, in our UML model, the “membershipNum” is a generated proxy because its value depends on the value of the “height” which is a proxy of the “gender”. In this case, the “membershipNum” is a mask for the “height”. Hence, knowing that discrimination happened because of using the “membershipNum”, will not uncover the original source for the discrimination.

An atomic guard condition  $g$  whose data attribute satisfies the properties above is called a *used condition* with respect to the sensitive decision  $s$ . For example, in our UML model, the “(height $\geq$ 170)” is a used condition with respect to the “scholarshipStatus”, because the height: is not a derived attribute, is not defined as explanatory with respect to the “scholarshipStatus”, and is a proxy of the “gender”, which does not belong to the explanatory data of the “scholarshipStatus”.

To explain the generation of batches of claims with respect to a state machine  $sm$  annotated with the «*individual-Fairness*» stereotype, we assume the following:

- $s_i \in SensitiveDecisions$ , where  $SensitiveDecisions$  is the set of all the data attributes and call operations in the  $\{sensitiveDecisions\}$  tag of  $sm$  and it is of the size  $n$ .
- $g_j \in usedConditions_{s_i}$ , where  $usedConditions_{s_i}$  is the set of the used conditions with respect to  $s_i$  and it is of the size  $k$ .
- $v_h \in V_{s_i}$ , where  $V_{s_i}$  is the set of all values in the range space of a sensitive data attribute  $s_i$  and it is of the size  $l$ .
- $event\_queues?[call\_s_i]$  is a logical condition. It returns true if the sensitive call-operation event  $s_i$  belongs to the  $event\_queues$  set, which includes the triggered events while executing the software model  $m$ .

Given the defined notations, Definition 1 shows how we define the set of all batches of claims  $B$ , which has to be generated with respect to a state machine  $sm$  annotated with the «*individual-Fairness*» stereotype. The definition shows that for each sensitive decision  $s_i$  with  $k$  used conditions,  $k$  batches of claims have to be generated, each batch with respect to a used condition  $g_j$  of  $s_i$ .

**Definition 1 (Batches of Claims).** Given  $sm$  is an «*individual-Fairness*»-annotated state machine in a software model  $m$ , the set  $B$  of all batches with respect to  $sm$  is defined as follow:

$$B = \left\{ \bigcup_{s_i=1}^{s_n} \bigcup_{g_j=1}^{g_k} batch_{s_i g_j} \right\}$$

where, in case  $s_i$  is a data attribute:

$$batch_{s_i g_j} = \left\{ \bigcup_{v_h=1}^{v_l} ((g_j \rightarrow \diamond s_i == v_h), (\neg g_j \rightarrow \diamond s_i == v_h)) \right\} \quad (\text{Rule 1})$$

while in case  $s_i$  is a call operation:

$$batch_{s_i g_j} = ((g_j \rightarrow \diamond event\_queues?[call\_s_i]), (\neg g_j \rightarrow \diamond event\_queues?[call\_s_i])) \quad (\text{Rule 2})$$

When  $s_i$  is a data attribute, a  $batch_{s_i g_j}$  has to be defined according to Rule 1 of Definition 1. In this case, the  $batch_{s_i g_j}$  is a set of pairs. According to Rule 1, the number of the pairs in a  $batch_{s_i g_j}$  is  $l$ , where  $l$  is the number of the possible values that can be assigned to  $s_i$ . Although this may theoretically be infinitely large, we assume, that in practice the number of the possible values that can be assigned to a decision data-attribute, is finite. Following Rule 1, each pair, that belongs to a  $batch_{s_i g_j}$ , consists of two claims: (1) a claim expresses that if the  $g_j$  condition is true, the value of the sensitive data attribute  $s_i$  will eventually equal  $v_h$ . (2) a claim expresses that if the  $g_j$  condition is false, the value of the sensitive data attribute  $s_i$  will eventually equal  $v_h$ . Where  $v_h$  is one possible value that can be assigned to  $s_i$ .

For example, recall the pairs of claims *pair1* and *pair2* in *Example 1*. By Rule 1 of Definition 1, *pair1* and *pair2* together are a batch of claims with respect to the “StProf\_StateMachine” in the example model shown in Figure 4.2(b).

When  $s_i$  is a call-operation event, a  $batch_{s_i, g_j}$  has to be defined according to Rule 2 of Definition 1. Following the specification of Rule 2, the  $batch_{s_i, g_j}$  consists of a pair of two claims: (1) a claim expresses that if the  $g_j$  condition is true, the call-operation event  $s_i$  will eventually be added to the *event\_queues* (i.e.,  $s_i$  will eventually be triggered). (2) a claim expresses that if the  $g_j$  condition is false, the call-operation event  $s_i$  will eventually be added to the *event\_queues*.

**Example 2.** Assume for a moment that our UML model has a transition labeled with “[membershipNum>=1] predict()”, where: (1) “predict()” is a call-operation event that, once triggered, will call a black-box operation that implements a scholarship decision-making algorithm. (2) “predict()” is defined as sensitive call-operation event with respect to the “scholarshipStatus”. Based on these assumptions, by Rule 2 of Definition 1, the following pair will be a batch with respect to the «individual-Fairness»-annotated state machine in Figure 4.2(b).

$$\left( \begin{aligned} &(\text{height} \geq 170 \rightarrow \diamond \text{event\_queues?}[\text{call\_predict}], \\ &\quad \neg(\text{height} \geq 170) \rightarrow \diamond \text{event\_queues?}[\text{call\_predict}]) \end{aligned} \right)$$

With this pair, we can check with respect to the software behavior whether the “predict()” call-operation event will be triggered for every two scholarship applicants which have identical data except for the “height”. This is to ensure that the scholarship decisions result from the same decision-making algorithm. Further analysis is required to ensure the fairness of the called algorithm.

#### 4.5.1 Algorithm for generating temporal logic claims

Algorithm 4.1 illustrates the generation of temporal logic claims. Our algorithm takes as input a UML model  $m$  annotated with the UMLfair profile and a list contains information about proxies of protected characteristics *proxList*. The algorithm returns  $B$  (the set of all batches of claims) with respect to  $sm$ , where  $sm$  is an «individual-Fairness»-annotated state machine in  $m$ .

First, in line 2 of the algorithm, the state machine that is annotated with the «individual-Fairness» stereotype will be retrieved. In the lines from 3-8 of the algorithm, the followings are declared and initialized:

1. the *protectedSet*. A set contains all the data that are defined as protected characteristics in the *{protectedData}* tags of the UML model  $m$ .
2. the *sensitiveSet*. A set contains all the data that are defined as sensitive decisions in the *{sensitiveDecisions}* of  $sm$ .
3. the *guardSet*. A set contains all atomic guards conditions in the model  $m$ .

---

**Algorithm 4.1:** Generating batches of claims from a UML model annotated with the UMLfair profile

---

```

1 generateBatchesOfClaims (m, proxList);
   Inputs : a UMLfair-annotated model m and a list of proxies proxList
   Output: a set of batches of claims B
2 sm ← getIndividualFairnessStateMachine(m);
3 protectedSet ← ∅;
4 sensitiveSet ← ∅;
5 guardSet ← ∅;
6 protectedSet ← getProtected(m);
7 sensitiveSet ← getSensitive(sm);
8 guardSet ← getGaurds(m);
9 B ← ∅;
10 foreach s ∈ sensitiveSet do
11   explanatorySet ← ∅;
12   explanatorySet ← getExplanatory(sm, s);
13   usedConditionsSet ← ∅;
14   usedConditionsSet ←
       getUsedConditions(m, s, proxList, protectedSet, explanatorySet, guardSet);
15   if s is a data attribute then
16     rangeSpaceSet ← ∅;
17     rangeSpaceSet ← getRangeSpace(s, m);
18     foreach g ∈ usedConditionsSet do
19       batch ← ∅;
20       foreach v ∈ rangeSpaceSet do
21         batch.add(
22           {g →<> s == v}, {!g →<> s == v});
23       end
24       B.add(batch);
25     end
26   end
27   else
28     foreach g ∈ usedConditionSet do
29       batch ← ∅;
30       batch.add(
31         {g →<> (event_queues?[call_s])},
32         {!g →<> (event_queues?[call_s])});
33       B.add(batch);
34     end
35   end
36 return B

```

---

In line 9 of Algorithm 4.1, an empty set  $B$  is declared. This set will be used to store all batches of claims that can be generated. In line from 10-14 of the algorithm, for each sensitive decision  $s$ , the followings will be performed:

1. in the lines from 11-12, the explanatory data with respect to  $s$  will be retrieved from the  $\{explanatroyData\}$  tag of  $sm$  are and stored in the  $explanatorySet$ .
2. in the lines from 13-14, all the used conditions in  $m$  with respect to  $s$  will be stored into the  $usedConditionsSet$ . This will be performed by calling the  $getUsedConditions(s, m, proxList, protectedSet, explanatorySet, guardSet)$  function. This function takes as input a UML model  $m$ , the sensitive decision  $s$ , the list of proxies  $proxList$ , the set of protected data  $protectedSet$ , the set of explanatory data  $explanatorySet$  and the set of the atomic guard conditions  $guardSet$  in the model. The function returns all used conditions in  $m$  with respect to the sensitive decision  $s$ . Details on how the properties of the used conditions are explained in Section 4.5.

In the line 15, for each sensitive decision  $s$  that is defined as a data attribute in  $m$ , the followings will be performed:

1. in the lines from 16-17, all the possible values that can be assigned to  $s$  will be retrieved from the UML model  $m$  and stored in the  $rangeSpaceSet$ .
2. in the lines from 18-24, for each used condition  $g$  with respect to the sensitive decisions  $s$ , a batch of temporal claims will be defined as follows: for each value belongs to the  $rangeSpaceSet$ , a pair of claims will be defined and added to the batch. Each pair should have the format of the temporal logic pair in line 22 of Algorithm 4.1. After iterating overall values in the  $rangeSpaceSet$ , the batch will be added to the set of all batches of claims  $B$ .

In the line 28, for each sensitive decision  $s$  that is defined as a call-operation event in  $m$ , the followings will be performed:

1. in the lines from 29-32, for each used condition  $g$  with respect to the sensitive decisions  $s$ , a batch of claims will be defined. Each batch must consist of two claims that conform to the format of the claim in line 31 of Algorithm 4.1.
2. Each generated batch will be added to the set of all batches of claims  $B$ .



## 4.6 Model Checking

In this section, we describe *phase3* of the proposed framework in Figure 4.3. After the batches of claims are generated, we verify the claims against the UML model. For this, the UML model has to be transformed into a formal verification language by using a transformation tool. The verification language depends on the model checker that will be used to verify the claims. The goal of any model checker is to search for a feasible trace of events in the targeted software model that violates a given claim. During the search, the model checker non-deterministically assigns values to the model’s data attributes, that are not part of the claim.

To verify the claims of *pair1* and *pair2* in *Example 1* against the UML model in Figure 4.2 by using the SPIN model checker, we first automatically transformed the UML model into PROMELA specifications by using the transformation tool Hugo/RT. The verification results are provided in Table 4.2. The result column shows “satisfied” if the corresponding claim is true at some point in any execution path. Otherwise, it shows “violated”.

**Table 4.2:** The verification results of the LTL claims

Pair	LTL Claim	Result
pair1	$height \geq 170 \rightarrow \diamond scholarshipStatus == true$	Satisfied
	$\neg(height \geq 170) \rightarrow \diamond scholarshipStatus == true$	Violated
pair2	$height \geq 170 \rightarrow \diamond scholarshipStatus == false$	Satisfied
	$\neg(height \geq 170) \rightarrow \diamond scholarshipStatus == false$	Satisfied

We have individual fairness if the claims of one pair are satisfied while the claims of the other pairs are violated. However, Table 4.2 shows that this is not the case. Specifically, Table 4.2 shows that the first claim of *pair1* is satisfied, meaning that if the “height” is greater than “170”, there is the possibility that the “scholarshipStatus” will eventually equal “true”. Based on the specifications of the state machine in Figure 4.2(b), this can happen when the “outstandingEducation” equals “true” and the “normalWeight” equals “true”.

In contrast, since in the UML model there is no other membership possibility except the basketball membership, when the “height” is less than “170”, it is impossible that the “scholarshipStatus” will eventually equal “true”. Therefore, the second claim of *pair1* is violated. Concerning the claims of *pair2*, they are both satisfied because the value of the “scholarshipStatus” in our UML model does not depend only on the “height”. Thereby, whatever the value of the “height” is, there is the possibility that the “scholarshipStatus” will eventually equal “false”. As a consequence, the “scholarshipStatus” will not always have the same value independent of the “height”. Thereby, there is the possibility to discriminate between scholarship applicants on the basis of their “height”, which is defined as a proxy for the “gender”.

[variable values]	model.pml:118 [state = Start]
outstandingEducation= true	model.pml:119 [transition = t <sub>1</sub> ]
normalWeighted = true	.....
membershipNum = 0	model.pml:450 [BbAppVerified = true]
scholarshipStatus = false	model.pml:453 [ScholarAppVerified = true]

**Figure 4.4:** Excerpt from the generated trace of events.

If we removed the guard condition “height $\geq$ 170” from transition “t<sub>4</sub>” in Figure 4.2, all the claims in Table 4.2 will be satisfied. This is because the “height” is not used in the model and, thereby, each claim will be satisfied because the “scholarshipStatus” depends on the values of the other attributes in the model. However, assuming that all the claims in Table 4.2 are satisfied, this does not mean that there is no possibility to discriminate between scholarship applicants on the basis of their “gender” because there might be other proxies for the “gender” in the model.

We have shown that the individual fairness regarding the “height” in *Example 1* is not preserved due to the verification results of *pair1* and *pair2*. We now show how to interpret the results to explain why the “height” can affect the “scholarshipStatus”, although it is only required for deciding about the basketball team membership.

To interpret the results, we rely on the counterexamples that the model checker generates for the violated claims. Figure 4.4 shows an excerpt from the trace of events that is generated by SPIN as a counterexample on violating the second claim of *pair1* in *Example 1*. Consider the two instructions in Figure 4.4: “bbAppVerified=true” and “scholarAppVerified=true”. The former indicates that transition “t<sub>8</sub>” in Figure 4.2 is fired while the later indicates that transition “t<sub>15</sub>” is fired. Based on the order of the executed instructions, transition “t<sub>8</sub>” is fired before transition “t<sub>15</sub>”. Thereby, there is a relation between being admitted to a scholarship and being admitted to a basketball membership.

## 4.7 Reporting on Individual Fairness

In this phase, we analyze the claim’s verification results to check if the individual fairness is preserved. Our proposed check for individual fairness is a function  $indFairness(B, R)$ , that takes as inputs: First, a set of generated batches of claims  $B$  with respect to an  $\langle\langle individual-Fairness \rangle\rangle$ -annotated state machine  $sm$  in a software model  $m$ . Second, a set  $R$  that contains the results of verifying the claims in  $B$  against the software model  $m$ . The function returns “ $sm$  preserves the individual fairness” if each batch in  $B$  has exactly one pair whose claims are satisfied while all the claims of the other pairs in that batch are violated.

The rationale behind our proposed check is that each batch in  $B$  consists of pairs of claims that together analyze the side effects of the data attribute of a used condition  $g$  on a sensitive decision  $s$ , where each pair, in isolation, considers claims about *one* possible side effect. Therefore, we have individual fairness with respect to the data attribute of  $g$  when deciding about  $s$  if the claims of only one pair in the batch are satisfied while the claims of the other pairs are not satisfied, that is, the data attribute of  $g$  always has the same side effect on  $s$ . For example, based on the verification results in Table 4.2, neither the claims of *pair1* are satisfied while the claims of *pair2* are violated, nor vice versa. Thereby, it is possible to discriminate between two scholarship applicants whose data are identical except for the “height”, which is a proxy of the “gender”.

#### 4.7.1 Algorithm for reporting on individual fairness

---

##### Algorithm 4.2: Individual-fairness-check

---

```

1 indFairness ( $B, R$ );
   Inputs : a set of batches of claims  $B$  and a set contains the claims'
            verification results  $R$ 
   Output: fairnessReport a report shows the result of the individual fairness
            analysis
2 create fairnessReport;
3 foreach  $batch \in B$  do
4   if  $\exists pair \in batch$  whose claims are satisfied while the claims of  $batch \setminus \{pair\}$ 
      are not satisfied then
5     | do nothing;
6   end
7   else
8     | create explainReport;
9     | explainReport.add(getResults(batch, R));
10    | fairnessReport.add("a violation for individual fairness in the model
      is detected"+ explainReport);
11  end
12 end
13 if fairnessReport is empty then
14  | fairnessReport.add("the analyzed state machine preserves individual
    | fairness");
15 end
16 return fairnessReport;

```

---

Algorithm 4.2 shows the proposed individual fairness-check. Algorithm 4.2 takes as inputs: (1) a set of generated batches of claims  $B$  with respect to a state machine  $sm$  annotated with «*individual-Fairness*» stereotype. (2)  $R$  a set contains the results

of verifying the claims in the set  $B$  against the targeted software model  $m$  by using a model checking technique. Algorithm 4.2 returns *fairnessReport* that shows whether the software model in question preserves individual fairness.

First, in line 2 of Algorithm 4.2, an empty *fairnessReport* will be created. In line 3, for each batch in  $B$  if there is only one pair in the batch whose claims are satisfied while the other pairs in the batch are all not satisfied, we do nothing. Otherwise, a violation will be reported in the fairness report together with all verification results of the claims in that batch. In line 13, if after iterating over all the batches in  $B$  the fairness report remains empty, the following sentence will be added to the fairness report: *"the analyzed state machine preserves individual fairness"*.

## 4.8 Optional: Generating Proxies From a Database

In our framework, when expert knowledge about proxies of protected characteristics is not available, a database that contains historical personal data of a true distribution can be a source for identifying such proxies. To support uncovering proxies of protected characteristics, the database needs to contain both protected characteristics and not protected characteristics. To find whether a piece of data in a software model is a proxy of protected characteristics, we assume a mapping from all not derived data attributes in the class diagram to the database schema, which could be given by the user or determined (semi-)automatically. Proxies will be statistically derived by using the correlation metric and the threshold that are specified in the software model. In the following, we show through an example how proxies of protected characteristics can be identified based on historical data.

Table 4.3: Student data

st_ID	gender	healthy	outstanding..	normalWeight	height	height*
st1	Female	True	True	True	169	False
st2	Female	True	True	True	173	True
st3	Female	True	True	True	165	False
st4	Female	False	True	False	167	False
st5	Female	False	True	False	160	False
st6	Male	True	False	True	178	True
st7	Male	True	False	True	183	True
st8	Male	False	False	False	185	True
st9	Male	True	False	False	175	True
st10	Male	True	True	False	180	True

Table 4.3 shows the "Student Data" table from the running example, the school database. The attributes namely "gender", "height", "outstandingEducation", and "normalWeight" are identical to those that have been described in the software model in Figure 4.2. The "healthy" is mapped to the "physicalHealthStatus" which is defined as protected in Figure 4.2. The "healthy" attribute is "True" if the student has no physical health issues and "False" otherwise.

**Table 4.4:** Correlations results

Conditional Entropy	Result
Entropy(gender   outstandingEducation)	0.39
Entropy(gender   normalWeighted)	0.97
Entropy(gender   height*)	0.39
Entropy(healthy   outstandingEducation)	0.875
Entropy(healthy   normalWeighted)	0.485
Entropy(healthy   height*)	0.79

Before measuring correlations, each data attribute in the database with a large range of possible values has to be transformed into a boolean data attribute based on its usage context in the model. For example, based on the “(height  $\geq$  170)” condition in Figure 4.2(b), the model will deal with any value of the “height” as either a value that can evaluate the “(height  $\geq$  170)” condition to a true or a false. Therefore, a new column called “height\*” is added to Table 4.3. It is “True” if the corresponding cell in “height” column satisfies the “(height  $\geq$  170)” condition and “False” otherwise.

In our model in Figure 4.2(b), the “conditionalEntropy” is specified in the *{metric}* tag as the correlation metric and the *{threshold}* tag sets the threshold value to “0.5”. The conditional entropy measures the *uncertainty* in a piece of data given another piece of data. The output of the conditional entropy is a value between “0” and “1”, where “0” means high correlation (i.e, low uncertainty) and “1” means no correlation (i.e, high uncertainty). Using the conditional entropy, we measured the correlations among the protected characteristics and other data attributes in the software model in Figure 4.2. The correlation results are summarized in Table 4.4. The results are calculated based on the data that are provided in Table 4.3. Each piece of data reduces the entropy in a protected characteristic to less than the specified threshold in the model (i.e., 0.5) is a proxy for that proctored characteristic. Based on Table 4.4, we consider the following data attributes as proxies: “outstandingEducation” and “height\*” each is a proxy for “gender” while “normalWeight” is a proxy for “healthy”.

## 4.9 Tool Support

Towards tool support, we developed<sup>11</sup>: (i) a Papyrus editor extension permits annotating UML models with the stereotypes and tags of the UMLfair profile, (ii) a pseudo-code describes our function to generate batches of LTL claims, (iii) a launch configuration to the transformation tool Hugo/RT, and (iv) a pseudo-code describes our check for individual fairness.

<sup>11</sup>The developed artifacts towards a tool support are provided online at <https://github.com/QRamadan/MoPrivFair-FairnessAnalysis> (accessed: 31/12/2019)

Based on our pseudo-codes, in the worst case, the number of the generated claims from a model will belong to  $\mathcal{O}(ns*ng*nv)$ , where:  $ns$  is the number of the sensitive decisions that are defined as data attributes,  $ng$  is the number of the used atomic guard conditions with respect to a sensitive decision and  $nv$  is the number of values in the range of a sensitive decision. In the following, we provide and explain the proposed pseudo-codes.

## 4.10 Case Studies

To study the applicability of our framework in Figure 4.3, we applied it to three case studies<sup>12</sup> featuring a school management system, a delivery management system and a loan management system.

*The School Management System* (shown in the running example) is an artificial case study, that describes situations where a student can apply for several activities organized by a school. Among the activities, deciding about a scholarship application is critical, since it should not discriminate between applicants based on their protected data such as their *gender* and their *physical health status*.

An excerpt from the resulted UML model is shown in Figure 4.2. A dataset of artificial data is created in order to uncover proxies for protected characteristics. The dataset contains information for 20 individuals and 6 data attributes for each individual. An excerpt of the dataset is shown in Table 4.3. The UML model and the dataset are created by master students in the context of a research seminar.

*The Delivery Management System* presents a real incident based on Amazon's delivery management system. The UML model of the delivery system is designed based on the incident's description<sup>13</sup>. In particular, Amazon's software offers a free-delivery service for prime customers whose orders exceed 35\$ and who live in zip-codes that are near to the locations of Amazon's stores. To uncover proxies of protected data, we created a dataset that contains data for 30 individuals and 5 data attributes for each individual. Our goal is to check if the delivery system model preserves individual fairness with respect to ethnicity when deciding about the free-delivery service.

---

<sup>12</sup>The artifacts used in this section are provided online at <https://github.com/QRamadan/MoPrivFair-FairnessAnalysis> (accessed: 31/12/2019). Our artifacts include: the analyzed UML models, the generated PROMELA specifications, the generated claims, and verification results of the generated claims.

<sup>13</sup>Detailed description for the discriminatory behavior of Amazon's software is available at <https://www.bloomberg.com/graphics/2016-amazon-same-day/> (accessed: 05/12/2019).

**Table 4.5:** An overview about the analyzed UML models

System Model	Total	Class	Operation + Signal	Attribute	State Machine	State	Fork Gate	Orthogonal State	Transition
School system model	55	2	6	8	2	15	2	1	29
Delivery system model	34	2	3	5	2	7	1	0	14
Loan system model	27	2	4	3	2	6	0	0	10

The *Loan Management System* is based on a real loan management business process model (BPMN) [132] which has been automatically generated from an event log recording the loan management process of a Dutch financial institute<sup>14</sup>. The BPMN model consists of two main processes, namely the *loan request management* and the *risk analysis management*. The former verifies whether a loan request will be accepted. The latter creates a loan proposal for each accepted loan request and performs a risk analysis to decide whether the proposal will be granted.

The UML model of the loan management system is generated from the business process model in [132] as follows: First, a class diagram is automatically generated, using our proposed transformation technique in Chapter 3. Second, for each subprocess, a state machine is manually created, based on the transformation rules in [116]. Due to the lack of details about the data used in the business process model, we assume the following: First, a loan request will be accepted if the applicant's account has no critical credit history and if the saving amount exceeds 1,000 Euro. Second, the risk analysis process is a black box decision-making component that preserves individual fairness. To uncover proxies, we derived data from the Statlog Credit dataset<sup>15</sup>, which stores 20 data attributes for 1,000 individuals. Among the 20 attributes, 3 attributes are considered as protected, namely gender, age, and marital status. Our goal is to check if the risk analysis component will be invoked for every two loan applicants who their data differ only in the protected characteristics and their proxies.

All the modeled procedures in our case studies are IF-THEN-ELSE rules that are almost similar to the example model in Figure 4.2. An overview of the analyzed UML models is provided in Table 4.5. The table shows the total number of UML elements in each of the analyzed models. It also provides an overview of the considered UML elements.

For example, the model of the loan system consists of 27 elements. The number of the elements in this model includes the number of the classes, attributes, operations, state machines, states, and transitions.

<sup>14</sup>The event log is available online at <https://www.win.tue.nl/bpi/doku.php?id=2012:challenge> (accessed:07/12/2019)

<sup>15</sup>The Statlog Credit is available online at [datasethttps://archive.ics.uci.edu/ml/datasets/statlog+\(german+credit+data\)](https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)) (accessed: 07/12/2019)

**Table 4.6:** The Detected Individual-Fairness Violations

Model	Num. Element	Num. Claims	Time (Sec)	Sensitive Decision	Num. Violations	Against Protected	Source	Through
School System	65	8	72	scholarshipStatus	2	gender	height	Data Flow
						physicalHealthStatus	normalWeight	Data Flow
Delivery System	34	4	36	freeDeliveryStatus	1	ethnicity	zipCode	Direct Usage
Loan System	27	4	36	call-operation riskAnalysis()	4	age (2 times)	creditHistoryStatus saving	Data Flow
						citizenship (2 times)	creditHistoryStatus saving	Data Flow

**Applying our framework.** We first annotated the UML models with individual fairness requirements, using our proposed UMLfair profile. To identify proxies for identified protected characteristics in the UML models, we used the conditional entropy with 0.5 as a threshold. We then generated batches of Linear Temporal Logic claims that conforms to Definition 1 by manually simulating Algorithm 4.1. The resulting claims were then automatically verified against the UML models, using the SPIN model checker.

Since the SPIN model checker accepts only PROMELA specifications, we first automatically transformed the UML models into PROMELA by using the Hugo/RT transformation tool. The verification results of the claims were then analyzed by manually simulating Algorithm 4.2 to check whether the designed UML models preserve individual fairness.

Table 4.6 summarizes the results of our analyses. The first column shows the total number of UML elements in the analyzed models. An overview of the considered UML elements in each model is provided in Table 4.5. The second and the third columns of Table 4.6 provide the number of the generated claims from the models and the time needed for their verification, respectively. For example, 4 claims were generated from the model of the loan system. The 4 claims took 32 seconds to be verified by the SPIN model checker. The tests were performed on a computer with a 2.2 Ghz processor and 8 GB of memory.

Table 4.6 also provides the number of detected fairness violations with respect to the defined sensitive decisions in the analyzed models. For each detected violation, the table provides: (1) the protected data, against which the violations occurred, (2) the source of the violation (i.e., the piece of data that caused the violation), and (3) whether the violation happened due to flow or a direct usage for the source of the violation. For example, the table shows four violations of individual fairness in the loan system model when deciding about triggering the “riskAnalysis()” call-operation. Two of the violations are against the “age” while the other two are against the “citizenship”. The violations against the “age” and “citizenship” happened due to flow for the “creditHistoryStatus” and the “saving”, which act as proxies for both the “age” and the “citizenship”.



## 4.11 Discussion

The proposed case studies demonstrate how our proposed framework allows for detecting violations against individual fairness. The *analysis for individual fairness* at design time is established via generating temporal logic claims that permit studying the effects of protected characteristics and their proxies on sensitive decisions.

In all case studies, proxies were identified using an underlying dataset to avoid using experts knowledge, because even when an expert for proxies in the domain is available, the approach using metrics and thresholds can assist the expert. As provided by Table 4.6, our approach helps in detecting violations for individual fairness at the design time of a software model, which avoids faulty implementations for the considered systems. For example, by applying our approach to the delivery system model, which is designed based on the description of Amazon’s incident, the same discriminatory behavior against the ethnicity is detected at design time. To check if our framework can detect discrimination in models that have concurrent executions, an orthogonal state machine is considered in the school system’s model. Applying our framework, all the expected discrimination are detected. Although the used models in our evaluation are small, our results may extrapolate to larger models. It is not necessary to always analyze a large model as a whole. There are techniques to analyze subsystems in separation such as the work in [98].

Software analysts (who are in charge of certifying the behavior of software with respect to individual fairness) can use our proposed framework without having software modeling expertise. Software developers (who are responsible for designing a fairness-aware software system) can use our proposed framework to detect discrimination early during the software design.

### 4.11.1 Using the framework at run-time

The usage of our proposed framework is not limited to the design time of the software system. As long as the software implementation is compliance with the software models, analysts can use our framework to reason about individual fairness at the run-time of the software system.

For instance, if the source code of a software system is derived from software models, the software models can be seen as an approximation to the software’s execution semantics, and therefore, the models can be used as a basis for the analysis. In legacy systems, models might be not available because of poor documentation for the system specifications. However, the models in a legacy scenario can be ex-

tracted from the code using reverse engineering techniques (e.g., [70, 144, 156]). Also there some work on model and code synchronization [19]. All of these aspects make it possible to use our framework for individual fairness analysis at the run-time of the software system.

Also, we would expect that professionals who apply our framework are not light-heartedly swayed that if there was no discrimination issue at a specific point of time, that no discrimination issues could happen in the future. For example, as discussed in this chapter, a source for discrimination on the basis of a protected characteristic is the use of a proxy for it (i.e., a data item that is highly correlated with the protected characteristic). It is possible that a data item that does not act as a proxy for protected characteristics at a specific point of time starts acting as a proxy at some point in the future. For this, our framework should be applied regularly from time-to-time using an updated database.

#### 4.11.2 Threats to validity, limitations and future work

Our study for the applicability of our proposed software model-based individual-fairness analysis is subject to a number of threats to *external* and *internal* validity. A threat to *external validity* is that detecting violations for individual fairness in our process highly depends on the robustness of the used model transformations and model checking tools. Therefore, a possibility for future work is to apply our process to a broader selection of cases using different model transformation and model checking tools to recommend which of them can perform best with our approach.

Threats to *internal validity* are: First, our approach searches only for a single attribute proxy while sometimes a proxy to a protected piece of data can be found in multiple attributes together. For example, maybe the *zip-code* alone is not a proxy for the *ethnicity* but the *zip-code* and *education level* together can do. A possible future work in order to addressing this limitation is to extend the process of generating claims to consider multi-attributes proxies. Second, since our framework considers fairness to be violated already if it would be violated at some intermediate stage of the model execution, false positives results may be reported (i.e. cases where our framework finds a violation of individual fairness although it is not really a violation). For example, if in Figure 4.2 “*scholarshipStatus*” is always set to true just before the final state (independently of whether “*height*≥170” holds), our analysis still would consider this model to violate individual fairness, because it does so at some state before the end (namely before it is set to true in all executions).

Another possibility for future work is to extend our framework to allow reasoning about other types of fairness such as *group fairness*. Different from individual fair-

ness, a decision-making software preserves the group fairness property if it produces equally distributed outputs for each protected group [49, 149]. Consider, for example, a loan decision-making software to decide whether loan applicants should be given loans. The software preserves group fairness with respect to *gender* if the outcome fractions of males and females who will get a loan are equal. Extending our framework to permit group fairness analysis is a further challenging task that requires, in the first place, extending our UMLfair profile to allow annotating system models with specific information regarding group fairness.

## 4.12 Related Work

Having proposed our framework for analyzing individual fairness based on software models, we now provide a discussion for the related work.

### 4.12.1 Software model-based development

Despite the availability of many model-based approaches [68], we have not found an existing approach described in the literature that would use software models for fairness analysis. The need for fairness analysis based on software models has been motivated by Brun et al. [18]. However, the work in [18] is not supported by an approach, that realizes the idea of analyzing fairness based on software models.

*From a conceptual perspective*, the closest model-based checks to our work are: the *no down flow-check* in [62] and the *purpose-check* in [5]. According to [62], the *no down flow-check* verifies whether a software model produces the same outputs for any two traces of events that differ only in their secret data. Although this check is very related to our work, different from our work, the *no down flow-check* does not consider proxies for secret data through the analysis. According to [5], the *purpose-check* is a static check that can be used to identify whether a software model processes users' data only for authorized purposes. With the *purpose-check*, we can avoid unauthorized direct discrimination, but not indirect discrimination, because it does not support the indirect effects of protected characteristics on the outputs.

*From a technical perspective*, the work in [63] is relevant to the proposed work in this chapter. The authors propose a software model-based approach that depends on temporal logic and the model checker SPIN to support reasoning about cryptography related issues in UML models, that are annotated with UMLsec profile.

### 4.12.2 Discrimination detection approaches

Most of the proposed discrimination detection approaches in the literature can be used as fairness testing components during the *testing phase* of the software development life cycle, which is only after the software is implemented. Most of the proposed approaches in this direction are *black-box* approaches [49, 101, 121, 122, 145, 154, 155]. These approaches apply different strategies and methods to study the changes in the overall software's output based on historical observations.

For example, FairTest [145] and Themis [49] are two methods that uncover discrimination in a software system by studying the correlations between its outputs and its inputs. Different from the FairTest[145], which requires a dataset of possible inputs to be given, Themis [49] automatically generates input test cases based on a schema describing valid software inputs.

The authors of [101, 121] proposed a method that extracts classification rules from a dataset of historical decision records. The classification rules are then directly mined to search for discrimination with respect to protected characteristics and their proxies. In [154, 155], the authors propose an algorithm for detecting discrimination by analyzing a causal network that captures the causal structure among the data stored in a historical decision records. In [122], the authors deploy privacy attack strategies to detect discrimination under hard assumptions about a given dataset of historical decision records, such as that the dataset is preprocessed to hide illegal discrimination from being detected. In [28] a domain-specific method called AdFisher is proposed. AdFisher aims to detect violations of individual fairness with respect to the ads that web users receive when visiting a web page. AdFisher monitors the changes in the ads based on web users' behaviors and privacy preferences specified in their web browsers.

The main drawback of black-box approaches is their inability to provide witnesses, that allow locating and explaining the source of detected discrimination. To address this challenge, in [29], the authors proposed a *white-box* approach that analyzes decision-making software and returns witnesses, that describe where and how discrimination happened. However, the work in [29] supports only the detection of discrimination in decision-making software, that make use of machine learning prediction models. FairSquare [7, 8] is a method to detect discrimination *at runtime*. The method verifies probabilistic assertions in the source code of decision-making software during its execution.

Different from our proposed work in this chapter, all the discussed approaches above require the software to be implemented. Hence, none of them can be used as a testing/verification component during the design phase of software models. However, since a UML model is an approximation to the actual behavior of its

corresponding software and since the UML language does not support modeling the behavior of machine learning algorithms well, our framework and existing discrimination detection approaches support each other, as follows:

- First, our framework can be used to avoid violations against individual fairness already during the design of software models. The goal is to reduce the difficulties of identifying and explaining the causes of detected discrimination in the later stages of the software development life cycle.
- Second, existing discrimination detection approaches can be used in the later stages of the software development process to check if the implementation of a decision-making software preserves individual fairness requirements.

### 4.13 Conclusion

In this chapter, we proposed a semi-automated, model-based framework that permits an analysis of individual fairness. The framework is based on UML software design models. We proposed a UML profile called UMLfair that allows annotating UML models with fairness-specific information. Using UMLfair, we enabled the generation of temporal logic claims, whose verification results against the UML models permit checking whether an individual fairness requirement is preserved by the software model. The novelty of our framework is that it is the first that permits fairness analysis during the design phase of software models. We validated the applicability of our framework using three case studies featuring a school management system, a delivery management system and a loan management system.



## Chapter 5

# Conclusions, Limitations, and Outlook

In this chapter, we summarize the key contributions of this thesis. We also discuss limitations of our work. Finally, we describe the directions for future research work.

### 5.1 Conclusion

We introduced a model-based methodology called MoPrivFair (**Model-based Privacy & Fairness**). The MoPrivFair methodology comprises three sub-frameworks to support: First, detecting conflicts between data protection requirements. Second, integrating BPMN- and UML-based data-protection engineering while supporting traceability for data protection requirements. Third, detecting discrimination against individuals on the basis of their protected characteristics.

In the following, we briefly describe the frameworks of the MoPrivFair methodology and then we elaborate on their evaluations.

**First, a BPMN-based framework for detecting conflicts.** This framework helps business analysts in detecting conflicts between security, data-minimization and fairness requirements during the business process modeling time. The framework supports: First, the specification of security, data-minimization and fairness requirements in business process models, based on security, data-minimization and fairness annotations. The security annotations in our BPMN extension were reused from an existing security-oriented BPMN extension called SecBPMN2, while the

data-minimization and fairness annotations are part of our contribution.

Second, checking the alignment between the security, data-minimization and fairness requirements and their specifications in the BPMN models. We automated this process by extending a graphical query language called SecBPMN2-Q to allow formulating the requirements as reusable procedural patterns. The patterns can be matched to BPMN models from the same domain. Third, detecting conflicts between the specified requirements in the BPMN models. Our technique for detecting conflicts analyses enriched models using a catalog of a domain-independent conflict anti-patterns, which was created using our SecBPMN2-Q extension as well. Alignment checking is required to avoid conflicts arising from changes to the requirements during their specifications in the business process models, which if detected later will make the process to find their root causes more difficult.

While the security annotations were reused from the SecBPMN2, our proposed framework is novel because it is the first to directly support modeling data-minimization and fairness requirements in BPMN models. It is also the first to support automatic conflict detection between specified security, data-minimization and fairness requirements in BPMN models.

**Second, a framework for integrating BPMN- and UML-based data-protection engineering.** This framework aims at integrating BPMN- and UML-based data-protection engineering while supporting traceability for data protection requirements. The main benefit is the management of data protection requirements from the views of the involved expert stakeholders, in particular, business analysts and systems engineers, in an integrated manner. To this end, we integrate the two well-known model-based security approaches SecBPMN2 and UMLsec via model transformation. Our framework suggests to iteratively: (i) transform business process models enriched with organizational data protection requirements using the SecBPMN2 to a preliminary software architectural model enriched with data protection policies using UMLsec [62]; (ii) refine the generated UMLsec architecture model manually with additional design decisions, and (iii) verify the resulting UMLsec architecture model against their contained data protected policies by using an automated tool called CARiSMA [4].

This framework is novel because it is the first that establishes an automatic traceability between high-level data-protection requirements and verifiable technical data-protection policies.

**Third, a UML-based framework for analyzing individual fairness.** This framework supports the analysis of UML-based software designs with regard to individual fairness. The analysis of individual fairness in our framework is established by generating temporal logic claims, whose verification against the targeted software



model enables reporting on the individual fairness of the software. Our framework includes the following contributions: (i) a UML profile *UMLfair* for annotating UML software models with fairness-specific information; (ii) a method for verifying individual fairness of annotated UML software models. Given a UML model annotated with fairness-specific information, our method for reasoning about individual fairness includes: First, generating temporal logic claims from the UML model. Second, verifying the generated claims against the UML model by using a model checking technique. Third, reporting on individual fairness based on the verification results of the claims.

Our framework for individual fairness analysis is novel in the sense that it is the first that permits fairness analysis based on software models at system design time.

### 5.1.1 Evaluation results

In the following, we provide the key evaluation results for our proposed work.

**Evaluating the BPMN-based framework for detecting conflicts.** We validated the applicability and usability of our conflict detection technique based on a case study of a healthcare management system, and an experimental user study, respectively.

*As part of the case study*, we designed a business process model featuring an over distance healthcare service. Using our framework, we were able to enrich the model with data-minimization and fairness requirements that represent data-protection preferences for patients. For conflict detection, we annotated the model with security requirements that represent security needs from the system point of view. Applied to the model, our conflict detection technique precisely detected 8 conflicts and 21 potential conflicts as expected (i.e, comparing to the manually detected conflicts and potential conflicts). Using models that differ in their sizes, we provided first insights into the scalability of our technique. By comparing the needed time for detecting conflicts in the different versions, we show that our technique does not show an exponential slowdown. The tests were performed on a computer with a 2.2 Ghz processor and 8 GB of memory.

*In an experimental user study*, we studied the usefulness of our technique for conflict detection between security, data-minimization and fairness requirements. The two main observations from our study can be summarized as follows: First, our technique supports the detection of conflicts that are hard to identify manually. Second, users generally perceive the output of our technique as helpful.

**Evaluating the framework for integrating BPMN- and UML-based data-protection engineering.** We validated the applicability of this framework based on a case study featuring an air traffic management system.

*As part of the case study*, we showed how our proposed framework allows data protection requirements to be managed and traced throughout the design of SecBPMN2 and UMLsec models. *Integrated management* of data protection requirements across different phases is established via our automated transformation that integrates the involved languages. *Traceability* is established via the trace models generated by our transformation rules. Based on the transformation results, we showed how the automated transformation saved us from re-implementing a large number of UMLsec annotations, which would have been daunting and error-prone. Our results are not restricted to any particular data protection-oriented extension of BPMN or UML, but can be applied to other ways of using BPMN and UML to address data protection requirements.

To offer insights into the scalability of our transformation technique, we performed a preliminary assessment based on our case study. The results showed that the performance of the proposed transformation technique seems adequate for practical use. For example, our technique took 7 seconds for transforming a SecBPMN2 model of 203 elements into a UMLsec model of 968 elements. Taking 7 seconds for the transformation represents a very low time comparing with the needed time for a manual transformation process, which may take hours. The tests were performed on a computer with a 2.2 Ghz processor and 8 GB of memory.

**Evaluating the UML-based framework for analyzing individual fairness.** We validated the applicability of our framework for analyzing individual fairness based on three case studies featuring a school management system, a delivery management system and a loan management system.

*As part of the case studies*, we demonstrated how our proposed framework allows for detecting violations against individual fairness at systems design time, which would avoid faulty implementations for the considered systems. For example, by applying our approach to the delivery system model, which presents a real incident based on Amazon's delivery management system, the same discriminatory behavior against ethnicity is detected at design time. To check if our framework can detect discrimination in models that have concurrent executions, an orthogonal state machine is considered in the school system's model. Applying our framework, 2 violations in the UML model of the school management system and 4 violations in the UML model of the loan management system were detected. The detected violations for individual fairness in the models precisely matched our exception (i.e, the manually detected violations).

### 5.1.2 Limitations

**Concerning our BPMN-based framework for conflict detection.** Our proposed extension to the SecBPMN2 allows specifying enforcement technologies for data protection annotations. However, we did not take into account the specified enforcement technologies during the conflict detection. While this information may be helpful to identify whether two requirements are conflicting or not, addressing available data protection mechanisms is not feasible because of the following two reasons: First, the number of available data protection mechanisms grows continuously [146]. This makes it difficult to have a database that contains updated information about all available data protection mechanisms and their abilities to preserve data protection requirements.

Second, the knowledge of security engineers on how to design and the knowledge of adversaries on how to break data protection mechanisms grows continuously as well [21]. A mechanism that today has the ability to preserve a specific data protection requirement might be broken tomorrow. Consequently, our catalog of conflict would be outdated very soon if it addressed this knowledge. To still raise awareness on possible conflicts, we model situations where the decision on whether two requirements are in conflict depends on the applied mechanisms, as potential conflicts. We made a systematic effort to assure completeness and correctness of our catalog of conflicts. We provided structured textual descriptions of all considered data protection requirements. The descriptions incorporate feedback from several rounds of revisions, and from the participants of an earlier experiment [111].

As mentioned earlier, we performed a user experiment to study the practical usefulness of our technique for conflicts detection between data protection requirements. Although that the used experimental material was based on one particular case study from the health care domain and relied to a large extent on students as participants in the study, our results can be generalized to case studies from other domains due to the following reasons: First, based on the results of previous study [125], students perform nearly equally well when faced with a novel software development approach, and are therefore suitable as stand-ins for practitioners.

Second, the considered security, data-minimization, and fairness requirements in our healthcare case study are applicable to other system from different domains. Consider, for example, a conference management system such as the easy-chair. In such systems, various security and data-minimization requirements can be considered. For instance, authors of a submitted paper and the reviewers of that paper may would like to be anonymous. In the same time, the system needs to prevent a paper submitter from being able to deny that she submitted the paper. Also, the system needs to prevent a reviewer from being able to deny that she wrote the review for a specific paper. However, since the usability of our conflicts de-

tection technique was assessed through a subjective questionnaire, performing another user experiment to assess the usability of our technique through an objective performance measures will provide us with more rigorous insights about the applicability of our approach in practice.

**Concerning our framework for integrating BPMN- and UML-based data-protection engineering.** Our proposed framework for integrating SecBPMN2 and UMLsec models focuses on a subset of SecBPMN2 of those security-specific data-protection annotations with an equivalent on the architectural level (i.e., class and deployment diagram). We did not consider the transformation of BPMN models to UML behavioral models such as activity and sequence diagrams because the BPMN model and these UML behavior diagrams provide the same level of information and almost share the same semantics. For example, generating UML activity diagrams from BPMN models would be a straightforward transformation.

Due to the large variability of SecBPMN2 models, our transformation for integrating SecBPMN2 and UMLsec models could be affected by errors. However, by applying our proposed 255 transformation rules to the SecBPMN2 model of the air traffic management system case study, we did not report errors. Although we studied this on a single case study, we assume that the same result will be reported when applying our transformation rules to other cases because the used SecBPMN2 model in our case study is large and it consists of several corner cases.

**Concerning our framework for analyzing individual fairness.** In our framework, we use Hugo/RT for transforming UML models into PROMELA specifications, which can be verified by the SPIN model checker. However, Hugo/RT does not transform any given UML model. It only allows the transformation of consistent UML diagrams. Also, some UML elements are not supported by the current version of Hugo/RT. For example, Hugo/RT does not allow the transformation of UML models that have classes of type *Enumeration*. As a result, potential violations of individual fairness might be undetected by our approach if Hugo/RT is used for the transformation without conforming to these restrictions. However, our framework does not rely on a specific transformation tool. It can use any other transformation tool such as UML-VT<sup>1</sup> and SMUML<sup>2</sup>.

Moreover, although that the used models in evaluating our framework for individual fairness analysis are small, our results may extrapolate to larger models. It is not necessary to always analyze a large model as a whole. There are techniques to analyze subsystems in separation such as the work in [98].

---

<sup>1</sup>UML-VT is available online at <https://www.cs.umd.edu/~rance/projects/uml-vt/> (accessed: 02/01/2020).

<sup>2</sup>SMUML is available online at <http://www.tcs.hut.fi/Research/Logic/SMUML.shtml> (accessed: 02/1/2020).

## 5.2 Outlook

Based on the outlined conclusions, it is apparent that the our proposed MoPrivFair methodology represents a step forward towards supporting data protection assurance by design. In addition to the discussed challenges in Section 1.1, a variety of research directions remains to be explored. This section mainly relies on the discussion and future work sections of previous chapters in this thesis. The scope of the proposed MoPrivFair methodology may be widened in several directions.

### 5.2.1 Conflict resolution

We aim to extend our BPMN-based framework for detecting conflicts, to support the resolution of conflicts. Once a conflict is detected, a solution strategy for solving the conflict should be decided. Although a fully automated process would be appreciated, the resolution of actual conflicts (e.g., between two requirements related to different views of system users) may require human intervention [42, 57], a further challenging task that involves reasoning on the security and privacy impacts of different solution strategies [6, 87]. For semi-automated support, we suggest a recommendation system that provides suggestions for resolving strategies based on historical decisions made by experts.

### 5.2.2 Tracing requirements in legacy and evolutionary systems

In the future, we aim to extend our proposed framework for integrating SecBPMN2 and UMLsec models to legacy situations, in which the UML design models are already given, rather than developed from scratch. Our mapping between SecBPMN2 and UMLsec security concepts can provide a foundation for addressing such legacy scenarios. Moreover, systems are continuously evolving due to changes at the organizational level or at the technical level [44]. Any change at the organizational or the technical level of a system may be accompanied by changes in data protection requirements. Considering the evolving nature of systems, we plan to extend our framework to support traceability of data protection requirements in case of changes to the business process model of a system or its architecture model.

### 5.2.3 Support group fairness analysis based on software models

As part of future work, we aim to extend our UML-based framework for analyzing individual fairness to allow reasoning about *group fairness*. Different from individual fairness, a decision-making software preserves the group fairness property if it produces equally distributed outputs for each protected group [49, 149]. Consider, for example, a decision-making activity in a bank to decide if loan applicants

should be given loans. We say the activity preserves group fairness with respect to gender if the outcome fractions of males and females who will get a loan are equal. Extending our framework to permit group fairness analysis requires, in the first place, extending our UMLfair profile to allow annotating system models with specific information regarding group fairness.

## Appendix A

# Conflict Detection: A Walk Through Artifacts and Tool Support

In this appendix, we present the artifact used in Chapter 2, which describes our proposed BPMN-based framework for detecting conflicts between security, data-minimization and fairness requirements. We also provide a manual for our prototypical implementation on top of STS, the supporting tool for the BPMN extension SecBPMN2, whose security requirements we reused. Our implementation supports: First, the modeling of security, data-minimization and fairness requirements in BPMN models, using a suitable model editor. Second, formulating security, data-minimization and fairness requirements as SecBPMN2-Q queries that can be verified against security-, data-minimization- and fairness-annotated BPMN model for alignment verification purpose. Third, automated conflict analysis in data-minimization-, security-, and fairness-annotated BPMN models, based on our catalog of anti-patterns.

**Resources.** We list the artifacts that are used in Chapter 2. One can use these artifacts to reproduce the conflict detection results that are reported in Chapter 2.

- **Artifact:** Mirrors of the STS-tool current versions:  
<http://www.sts-tool.eu/downloads/secbpmn-dm>
- **The Experimental Evaluation Artifacts:** for transparency we share all our experimental materials: <https://github.com/QRamadan/MoPrivFair-ConflictsDetection/blob/master/ExperimentalArtifacts.zip>

- **Artifact:** a healthcare case study and a catalog of conflicts specified as anti-patterns: <https://github.com/QRamadan/MoPrivFair-ConflictsDetection/blob/master/Telemedicine.exp>

**Performing automatic conflicts detection.** To view and analyze our designed models please follow the instructions:

- Install the STS tools from  
<http://www.sts-tool.eu/downloads/secbpmn-dm>
- Download the *telemedicine.exp* file to your desktop. This file is automatically generated from the STS tool and it contains all the SecBPMN2 models of our case study. The file can be downloaded from:  
<https://github.com/QRamadan/MoPrivFair-ConflictsDetection/blob/master/Telemedicine.exp>
- To view and modify the models in this file you need to import it into the STS:
- In the STS tool, do *File* → *Import* → *project* → *next* → *select the "telemedicine.exp" file from your desktop.*
- The projects contains two SecBPMN2 models namely, "SoSyM19-PaperExample.bpmnl" and "Full\_annotatedModel.bpmnl".
- The project contains a catalog of domain-independent anti-patterns.
- To run a conflict analysis open the SecBPMN2 model you want to analyze from the following directory:  
*/SoSym19-telemedicine/Models/SecBPMN2/Business process/*
- In the STS tool, do *Analysis* → *Check Security Policy.*
- The analysis will take a few minutes. Once the analysis is finished press *OK.*
- The analysis result will be shown as textual messages in a new window. The messages will be of two types "Error" or "Warning". The former represents conflict while the later represents potential conflict.
- To view the anti-patterns find the following directory:  
*/SoSyM19-telemedicine/Models/SecBPMN2/Conflicts\_Patterns*



## Appendix B

# SecBPMN2 to UMLsec: A Walk Through Artifacts and Tool Support

This appendix presents the artifacts used in Chapter 3, which provides a Framework for Integrating BPMN- and UML-based Data-Protection Engineering. This framework is a sub-framework of the proposed MoPrivFair methodology. This appendix includes the model transformation from SecBPMN2 to UMLsec models as well as four examples models from the Air Traffic Management System case study. It explains the process of using the transformation, and the verification of the generated UMLsec models using the CARiSMA tool.

**Resources.** In the following, we list the artifacts that are used in Chapter 3. We also provide access to the implementation of our transformation tool.

- **Artifact:** Eclipse Project package (*myexample.zip*):

<https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec/blob/master/myexample.zip>

- **Artifact:** A mirror of the CARiSMA update site:

<https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec/blob/master/CARiSMA.zip>

- **Artifact:** A mirror of the Henshin update site:

[https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec/blob/master/org.eclipse.emf.henshin.sdk\\_1.5.0.zip](https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec/blob/master/org.eclipse.emf.henshin.sdk_1.5.0.zip)

- **Artifact:** Air Traffic Management Case Study SecBPMN2 models:  
<https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec/blob/master/projects.exp>
- **Artifact:** Mirrors of the STS-tool current versions: <https://figshare.com/s/b322d002077d99577949>

**Artifact contents.** The project package *myexample.zip* has the following contents:

- *src/my.example* directory:
  - *Externalservices1.bpmn*, *Externalservices2.bpmn*, *Flightplan.bpmn*, *Landing.bpmn*: Input models from the ATM case study.
  - *example1.bpmn*: small input model for testing purposes.
  - *\*.henshin files*: Henshin modules implementing the transformation.
  - *BpmnToUml.java*: Java class for executing the transformation via orchestration of Henshin modules.
  - *BpmnToUmlMetricsPrinter.java*: Java class for computing metric values for input and output models.
- *Testing directory*: Additional test input models

**Prerequisite.** We recommend using Eclipse Neon, Modeling Tools distribution, with installed Henshin and CARiSMA plug-ins. These plug-ins can be installed either using online update sites, or the mirrored update sites provided as part of the artifact. *From the CARiSMA update site, please only install the main features (BPMN2 and UML2 support).*

- **Installation from online update site:** In Eclipse, do *Help* → *Install New Software...* Use the following online update sites: <http://carisma.umlsec.de/updatesite> for CARiSMA, and <http://download.eclipse.org/modeling/emft/henshin/updates/release> for Henshin.
- **Installation from mirrors:** Download the mirrored update sites of CARiSMA and Henshin to your computer → install both of them in Eclipse, using *Help* → *Install New Software...* → *Add* → *Archive*.

---

**Performing the transformation.** To execute the transformation from SecBPMN2 to UMLsec models, please follow the following instruction.

- Import our project package to your local Eclipse workspace.
- Right-click on the main class "*src/my.example/BpmnToUml.java*" → *Run As JUnit Plug-in Test*. By default, our transformation takes as input the *example1.bpmn* file. To change the input file, first copy the name of one of the BPMN files that are provided in *myexample* → *src* → *my.example* directory. Second, find line 91 in the *BpmnToUml.java* file (**public static final String EXAMPLE = "example1.bpmn";**) and replace the file name "*example1*" with the name of the selected BPMN file. *Please note that you cannot directly view or modify these bpmn models from you Eclipse. For viewing and modifying the bpmn models please look into the last section of this Appendix viewing and modifying the SecBPMN2 models.*
- After running the *BpmnToUml.java* file, you should see the console output informing you about the generation process. The process could take a few minutes, and there might be some warnings/error messages related to the underlying plug-ins. As these do not concern us, we can ignore them. The process is finished when the following line is printed to the console: Saved result in '*example1-generated-result.uml*'. The name of the *.uml* file in this line is the name of the selected *.bpmn* file as input to our transformation followed by *-generated-result.uml*.
- The results of the transformation process are three files that will be stored to the *myexample/src/my.example* directory:
  - The first file contains the generated UMLsec model. The name of this file is the name of the selected *.bpmn* file as input to our transformation followed by *-generated-result.uml* (e.g., *example1-generated-result.uml*).
  - The second file contains the generated trace model which links the SecBPMN2 and UMLsec models. The name of this file is the name of the selected *.bpmn* file followed by *-generated-result-trace.xmi* (e.g., *example1-generated-result-trace.xmi*).
  - The third file contains the UMLsec operations that have access restrictions together with rights the should be assigned to the roles to grant them access for these operations. The name of this file is the name of the selected *.bpmn* file followed by *-generated-result-rights.txt* (e.g., *example1-generated-result-rights.txt*).

**Performing the verification.** In this step, we use CARiSMA checks to verify the generated UML models against UMLsec security policies. For this, please follow the following instructions.

- Right-click on the *myexample* project in the *Project Explorer view* → *New* → *Other* → *CARiSMA* → *Analysis* → *Next* → in the dialog select the file that is generated from the last step (e.g., *example1-generated-result.uml*) and then click *finish*.
- From the *analysis editor* window click on the *add checks to list* icon → select the check that you want to perform (e.g., *secure links UMLsec check* and *secure dependency UMLsec check*) then click *RUN*.
- For RABAC policy, you have to select both *RABACsec: Create transformation input* and *RABACsec: Use transformation input checks* checks. The former allows you to select the role that you want to verify his accessibility to the system operations, while the second return the set of operations that the selected role have access to them.
- The verification results are provided in the *Analysis Results view*. One can *right-click* on the result and select *create a report for the selected analysis*. The report will be stored to the project root directory. **Please note** that all the checks should pass except for RABAC which depends on rights that the user would like to assign to each role during performing RABACsec: Use transformation input checks.
- More details about the execution of abac checks will provided below. Other information can be found in the user manual of CARiSMA. After installing CARiSMA, the manual is available under: *Help* → *Help Contents* → *CARiSMA*.

**More details information about performing CARiSMA checks.** In what follows, we walk through the three checks supported by our transformation output: «*secure links*» (for deployment diagrams), «*secure dependency*» (for class diagrams), and RABAC (for class diagrams).

1. «*secure links*» UMLsec check.

**Purpose:** To check whether the communication links between physical nodes are secure or not with respect to the adversary type and the data communicated across them. To perform this check on the generated UML file from the earlier steps (e.g., *example1-generated-result.uml*), please mind the following instructions:

- From the *analysis editor* window click on the *add checks to list* icon → select the *secure links UMLsec check* → click *OK* and then click *RUN*.

## 2. «*secure dependency*» UMLsec check.

*Purpose:* To check whether the dependencies between objects or subsystems respect the security requirements on the data communicated across them. To perform this check on the generated UML file from the earlier steps (e.g., `example1-generated-result.uml`), please follow the following instructions:

- From the *analysis editor* window click on the *add checks to list* icon → select the *secure dependency UMLsec check* → click *OK* and then click *RUN*.

## 3. RABAC (Role Attribute-based Access Control).

**Purpose:** To check the access rights of each role and the access constraints assigned to specific operations based on predefined attributes. UMLsec implements the RABAC access control model via the policy «*abac*», which uses two tags called *{role}* and *{right}* to assign roles to subjects and rights to roles, respectively. Operations in need of an access restriction can be annotated with the «*abacRequire*» stereotype along with its associated *{right}* tag.

*Prerequisite:* In our transformation, we can automatically assign *roles* to subjects and the *rights* that are needed for restricting the access for some operation. Since our transformation does not allow to automatically specify all the required information for executing the RABAC check, some information still needs to be inserted manually: One needs to manually assign rights to roles, based on the following instructions:

- Right click on the generated UML file from the last step (e.g., `example1-generated-result.uml`) → *open with* → *UML Model Editor*.
- Click on `platform:/resource/myexample/example1-generated-result.uml` → *open the model* → *select the RABAC class*.
- In the *properties* view, assign a value to the "*right*" property. The value should have the following format: **{(role\_name,right)}**. For instance, `{(Airplane,Modify_Flight plan)}`, means that the Airplane has the right to modify a flight plan data. Multiple rights can be given to the same role as follows: `{(Airplane,Modify_Flight planRead_Flight plan)}`. This means that the Airplane has the right to both read and modify the flight plan.
- One can also get benefits from the generated `example1-generated-result-rights.txt` file that contains all the operations that have an access restriction together with the rights that should be assigned to the system roles to grant them an access to these operations. In what follows, we will assume that `{(Airplane,Modify_Flight planRead_Flight plan)}` is the specified as a value for the "*right*" property. *Please a void whitespaces in your entries, otherwise a matching cannot be found and the check will fail.*
- Save the changes you made on the models. Perform RABAC checks: To perform this check on the generated UML file (i.e., Trans-

formed\_serialized\_profile.uml) from the last steps, please mind the following instructions:

- In the *analysis editor* window, select both the *RABACsec: Create transformation input* and *RABACsec: Use transformation input checks* and click *OK*.
- In the checks list, you will see the selected RABAC checks. To perform the checks, first unselect the selected checks, only select the *RABACsec: Create transformation input* and click *RUN*.
- Save the configuration file "*rabac\_configuration.xml*" into *myexample/src/my.example* directory.
- In the pop-up window "*RABACsec transformation input*", first click on the droplist of the users and select "*subject*".
- From the droplist of the roles, we can select a role for which we want to view the accessible operations. For example, select *Airplane*.
- The *Active* checkbox allows you to identify whether the selected role is active or not. Please tick the checkbox.
- Put the cursor inside the *Value* textbox and press *enter* to save the changes. This textbox can be used to specify some attributes for the selected role. However, this not part of our work. Therefore, please leave the textbox empty.
- Click *Save* and close the window.

**Hint:** Please press *enter* in the textbox to save the changes, ignoring the *Save* button. This because of a bug in the implementation of the check.

- In the *analysis editor* window, uncheck the last performed check (*RABACsec: Create transformation input*), only select *RABACsec: Use transformation input checks* check and click *Run*.
- Press *OK* in the pop-up window and select the generated configuration file from the last step (i.e., "*rabac\_configuration.xml*") and click *Open*.
- The result of the verification is provided in the *Analysis Results view*.

To generate the report text file for the generated checks, you can *right-click* on the result and select *create a report for the selected analysis*. The report will be stored to the *myexample/src/my.example* directory. In our example, the output result for the RABAC check will show that the selected *Airplane* role has access to *Notify local authority* operation.

---

**Viewing and Modifying the SecBPMN2 Models.** The BPMN models that are provided in `my.example` directory represents a SecBPMN2 models for our Air Traffic Management System case study. Please note that you can not open or modify these models directly from your Eclipse. These models are designed by using the Socio-Technical-System (STS) tool. To view and modify these BPMN models please follow the instructions:

- Install the STS tools from (<http://www.sts-tool.eu/>) or from the mirrored files of the current STS-version provided as part of the artifacts (available from <https://figshare.com/s/b322d002077d99577949>).
- Download the *projects.exp* file from (<https://github.com/QRamadan/MoPrivFair-SecBPMN2UMLsec/blob/master/projects.exp>) to your desktop. This file is automatically generated from the STS tool and it contains all the SecBPMN2 models of our case study.
- To view and modify the models in this file you need to import it into the STS:
  - In the STS tool, do *File* → *Import* → *project* → *next* → select the "projects.exp" file from your desktop.
- In case you want to insert a new designed SecBPMN2 model or a modified version of the provided SecBPMN2 models into our transformations approach, please follow the following instructions:
  - First, from the workspace of the STS tool please find the SecBPMN2 model file that you want to transform.
  - Second, simply copy and paste the file into the *my.example* directory.
  - Third, since our approach take an input a (.bpmn) files, please right click on the inserted *file* → *rename* → change the file extension to bpmn.

**Viewing the generated UMLsec model.** To create a Papyrus UML diagram initialized with the contents of a given .uml model file (e.g., `example1-generated-result.uml`), please follow the below instructions:

- create a new Papyrus model by right-clicking the UML model (e.g., `example1-generated-result.uml`) → *New* → *Other* → *select Papyrus Model* → *Select UML* → save the model to the root directory (i.e., `myexample`) → *Finish*.
- From the *Repair* Stereotypes and some profiles have changed pop-up windows press *OK*. After that, the *.di* and *.notation* files are created. By following the previous instructions, the required files to work with the UML model in Papyrus can be initialized.

- Now make sure that you are working on the Papyrus modeling perspective → *open the created Papyrus model* → *create view* → *select the root element of our UML model* → *select the desired type of UML diagrams* that you want to create → *enter a name to the diagram*. **Please note that our .uml file only contains elements related to the Deployment and Class diagrams.**
- After opening the created diagram you will see nothing but a blank window. This because in general .uml file has no information regarding any diagram. All you have is the model elements, but no diagrams. Therefore, you have to create the diagrams manually by dragging and dropping the model elements to the desired diagram view (e.g., class ). To do so, make sure that you are working in Papyrus modeling perspective view and the *Model Explorer* view is visible. Since one can see that dragging and dropping the UML elements to the diagram view manually is a time and efforts consuming, we can suggest another way to perform this task.
- After creating the desired UML diagram, *select Diagram from the menu bar of your Eclipse* → *Filters* → *Synchronized with Model*. Then all the UML classifiers elements will be automatically inserted to your diagram. Personally speaking, this suggestion is useless since you need to remove all the classifiers that are not related to your diagram. For example, if you want to create a class diagram, by following the last suggestion not only the classes will be inserted into your class diagram view but also the UML nodes and artifacts which are part of our deployment diagram. Moreover, you still need to drag and drop the missing details such as the operations, dependencies, and associations manually from the *Model explorer* view. **For further information, please see** <https://www.eclipse.org/forums/index.php/t/1071157/>.

**Computing the metric values for input and output models.** The Eclipse project package *myexample.zip* has a *BpmnToUmlMetricsPrinter.java* class for computing the metric values for input and output models (i.e., SecBPMN2 and UMLsec models). To run the metrics printer, please follow the following instructions:

- First, you need to transform all example BPMN models, namely "*example1.bpmn*", "*Flightplan.bpmn*", "*Landing.bpmn*", "*Externalservices1.bpmn*", and "*Externalservices2.bpmn*" to UMLsec models.
- Right-click on the *BpmnToUmlMetricsPrinter.java* class → *Run As* → *Java Application*. There might be some warnings/error messages related to the underlying plug-ins.
- The result is a console output shows the models and their metrics values.



## Appendix C

# Curriculum Vitae

### Personal Data

Name: Qusai Ramadan  
Date and place of birth: 09/07/1986, Al-Ramtha, Jordan  
Address: Am alten Schützenplatz 1, 56072 Koblenz  
Contact: qramadan@uni-koblenz.de  
+49 261 287-2795

### Working Experience

10/2015 - till present: Researcher at Software Engineering Research Group,  
(Prof. Dr. Jan Jürjens),  
Koblenz-Landau University, Germany. (Full Time)  
11/2011 - 08/2013: Lecturer at Preparatory Year Deanship,  
Najran University, Saudi Arabia. (Full Time)  
12/2010 - 06/2010 and  
06/2008 - 11/2008 Teacher assistance at Computer Science Department,  
Jordan University of Science and Technology,  
Jordan. (Part Time)

**Grants**

- A research grant from the German Academic Exchange Service (DAAD) for the time period between 09/2015 and 09/2019.

**Education**

- M.Sc. in Information Technology (2009-2010), Information Technology Dept., Utara University, Malaysia, Rating (Excellent).

- B.Sc. in Computer Science (2005-2008), Computer Science Dept., Al Al-Bayt University, Jordan, Rating (Very Good).

# Bibliography

- [1] Business Process Model and Notation (BPMN) Version 2.0. Number formal/2011-01-03. Object Management Group (OMG), January 2011. URL <https://www.omg.org/spec/BPMN/2.0/PDF> (accessed: 18/12/2019).
- [2] Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data., 2016.
- [3] OMG® Unified Modeling Language® (OMG UML®) Version 2.5.1. Number formal/2017-12-05. Object Management Group (OMG), December 2017. URL <https://www.omg.org/spec/UML/2.5.1/PDF> (accessed: 18/12/2019).
- [4] Amir Shayan Ahmadian, Sven Peldszus, Qusai Ramadan, and Jan Jürjens. Model-Based Privacy and Security Analysis with CARiSMA. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 989–993. ACM, 2017.
- [5] Amir Shayan Ahmadian, Daniel Strüber, Volker Riediger, and Jan Jürjens. Model-based privacy analysis in industrial ecosystems. In *European Conference on Modelling Foundations and Applications*, pages 215–231. Springer, 2017.
- [6] Amir Shayan Ahmadian, Daniel Strüber, Volker Riediger, and Jan Jürjens. Supporting privacy impact assessment by model-based privacy analysis. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1467–1474. ACM, 2018.
- [7] Aws Albarghouthi and Samuel Vinitzky. Fairness-aware Programming. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 211–219. ACM, 2019.
- [8] Aws Albarghouthi, Loris D’Antoni, Samuel Drews, and Aditya V. Nori. FairSquare: Probabilistic Verification of Program Fairness. *Proceedings of the ACM on Programming Languages*, (Object-Oriented Programming, Systems, Languages & Applications), 2017.

- [9] Maysoon Aldekhail, Azzedine Chikh, and Djamel Ziani. Software Requirements Conflict Identification: Review and Recommendations. *International Journal of Advanced Computer Science and Applications*, 7(10):326–335, 2016.
- [10] Duaa Alkubaisy. A framework managing conflicts between security and privacy requirements. In *International Conference on Research Challenges in Information Science*, pages 427–432. IEEE, 2017.
- [11] Sascha Alpers, Roman Pilipchuk, Andreas Oberweis, and Ralf Reussner. The Current State of the Holistic Privacy and Security Modelling Approach in Business Process and Software Architecture Modelling. In *International Conference on Information Systems Security and Privacy*, pages 109–124. Springer, 2018.
- [12] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: advanced concepts and tools for in-place EMF model transformations. In *International Conference on Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2010.
- [13] Wihem Arzac, Luca Compagna, Giancarlo Pellegrino, and Serena Elisa Ponta. Security validation of business processes via model-checking. In *International Symposium on Engineering Secure Software and Systems*, pages 29–42. Springer, 2011.
- [14] Solon Barocas and Andrew D. Selbst. Big data’s disparate impact. *Cal. L. Rev.*, 104:671, 2016. URL <http://www.cs.yale.edu/homes/jf/BarocasSelbst.pdf> (accessed: 18/12/2019).
- [15] Andreas Bauer, Jan Jürjens, and Yijun Yu. Run-time security traceability for evolving systems. *The Computer Journal*, 54(1):58–87, 2010.
- [16] Kristian Beckers, Stephan Faßbender, Maritta Heisel, and Rene Meis. A Problem-based Approach For Computer-Aided Privacy Threat Identification. In *Annual Privacy Forum*, pages 1–16. Springer, 2012.
- [17] Achim D. Brucker, Isabelle Hang, Gero Lückemeyer, and Raj Ruparel. SecureBPMN: Modeling and enforcing access control requirements in business processes. In *ACM Symposium on Access Control Models and Technologies*, pages 123–126. ACM, 2012.
- [18] Yuriy Brun and Alexandra Meliou. Software Fairness. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 754–759. ACM, 2018.

- [19] Hugo Bruneliere, Jordi Cabot, Frédéric Jouault, and Frédéric Madiot. MoDisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 173–174. ACM, 2010.
- [20] Jens Bürger, Stefan Gärtner, Thomas Ruhroth, Johannes Zweihoff, Jan Jürjens, and Kurt Schneider. Restoring security of long-living systems by co-evolution. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pages 153–158. IEEE, 2015.
- [21] Jens Bürger, Daniel Strüber, Stefan Gärtner, Thomas Ruhroth, Jan Jürjens, and Kurt Schneider. A framework for semi-automated co-evolution of security knowledge and system models. *Journal of Systems and Software*, 139:142–160, 2018.
- [22] Toon Calders and Sicco Verwer. Three naive Bayes Approaches For Discrimination-Free Classification. *Data Mining and Knowledge Discovery*, 21(2):277–292, 2010.
- [23] Gary Charness, Uri Gneezy, and Michael A. Kuhn. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization*, 81(1):1–8, 2012.
- [24] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
- [25] Yulia Cherdantseva and Jeremy Hilton. A reference model of information assurance & security. In *2013 International Conference on Availability, Reliability and Security*, pages 546–555. IEEE, 2013.
- [26] Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.
- [27] Amit Datta. *Fairness and Privacy Violations in Black-Box Personalization Systems: Detection and Defenses*. PhD thesis, Carnegie Mellon University, 2018.
- [28] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated Experiments on Ad Privacy Settings. *Proceedings on privacy enhancing technologies*, 2015(1):92–112, 2015.
- [29] Anupam Datta, Matt Fredrikson, Gihyuk Ko, Piotr Mardziel, and Shayak Sen. Proxy Non-Discrimination In Data-Driven Systems. *arXiv preprint arXiv:1707.08120*, 2017. URL <https://arxiv.org/abs/1707.08120> (accessed: 18/12/2019).

- [30] Anupam Datta, Matthew Fredrikson, Gihyuk Ko, Piotr Mardziel, and Shayak Sen. Use Privacy in Data-Driven Systems: Theory and Experiments with Machine Learnt Programs. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS'17*, pages 1193–1210. ACM, 2017.
- [31] Narayan Debnath, Carlos Alejandro Martinez, Fabio Zorzan, Daniel Riesco, and German Montejano. Transformation of business process models BPMN 2.0 into components of the Java business platform. In *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, pages 1035–1040. IEEE, 2012.
- [32] Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, and Wouter Joosen. A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32, 2011.
- [33] Dorothy E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, 1976.
- [34] Vasiliki Diamantopoulou, Nikolaos Argyropoulos, Christos Kalloniatis, and Stefanos Gritzalis. Supporting The Design Of Privacy-Aware Business Processes via Privacy Process Patterns. In *International Conference on Research Challenges in Information Science*, pages 187–198. IEEE, 2017.
- [35] Remco Dijkman and Pieter Van Gorp. BPMN 2.0 execution semantics formalized as graph rewrite rules. In *International Workshop on Business Process Modeling Notation*, pages 16–30. Springer, 2010.
- [36] Cynthia Dwork and Christina Ilvento. Fairness Under Composition. *arXiv preprint arXiv:1806.06122*, 2018. URL <https://arxiv.org/abs/1806.06122> (accessed: 18/12/2019).
- [37] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness Through Awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226. ACM, 2012.
- [38] Steve Easterbrook. Resolving requirements conflicts with computer-supported negotiation. *Requirements engineering: social and technical issues*, 1:41–65, 1994.
- [39] Alexander Egyed and Paul Grunbacher. Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability can Help. *IEEE software*, 21(6):50–58, 2004.
- [40] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete knowledge. In *Proc. of Computational Logic*, volume 1861, pages 807–821. 2000.

- [41] Mohamed El-Attar. From misuse cases to mal-activity diagrams: bridging the gap between functional security analysis and design. *Software & Systems Modeling*, 13(1):173–190, 2014.
- [42] Golnaz Elahi and Eric Yu. A Goal Oriented Approach for Modeling and Analyzing Security Trade-offs. In *International Conference on Conceptual Modeling*, pages 375–390. Springer, 2007.
- [43] Marcelo Fantinato, Maria Beatriz Felgar de Toledo, Lucinéia Heloisa Thom, Itana Maria de Souza Gimenes, Roberto dos Santos Rocha, and Diego Zuquim Guimarães Garcia. A survey On Reuse In The Business Process Management Domain. *International Journal of Business Process Integration and Management*, 6(1):52–76, 2012.
- [44] Michael Felderer, Basel Katt, Philipp Kalb, Jan Jürjens, Martín Ochoa, Federica Paci, Thein Than Tun, Koen Yskout, Riccardo Scandariato, Frank Piessens, et al. Evolution of security engineering artifacts: a state of the art survey. *International Journal of Secure Software Engineering (IJSSE)*, 5(4):48–98, 2014.
- [45] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 259–268. ACM, 2015.
- [46] David Ferraiolo, Janet Cugini, and D. Richard Kuhn. Role-based Access Control (RBAC): Features and Motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.
- [47] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.
- [48] Ariel Fuxman, Marco Pistore, John Mylopoulos, and Paolo Traverso. Model checking early requirements specifications in tropos. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pages 174–181. IEEE, 2001.
- [49] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing Software for Discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 498–510. ACM, 2017.
- [50] Daniel Ganji, Haralambos Mouratidis, Saeed Malekshahi Gheytsassi, and Miltos Petridis. Conflicts Between Security and Privacy Measures in Software Requirements Engineering. In *International Conference on Global Security, Safety, and Sustainability*, pages 323–334. Springer, 2015.

- [51] Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee, and Siv Hilde Houmb. An aspect-oriented methodology for designing secure applications. *Information & Software Technology*, 51(5):846–864, 2009.
- [52] Maya Gupta, Andrew Cotter, Mahdi Milani Fard, and Serena Wang. Proxy Fairness. *arXiv preprint arXiv:1806.11212*, 2018. URL <https://arxiv.org/abs/1806.11212> (accessed: 18/12/2019).
- [53] Seda Gürses, Carmela Troncoso, and Claudia Diaz. Engineering privacy by design. *Computers, Privacy & Data Protection*, 14(3), 2011. URL <https://software.imdea.org/~carmela.troncoso/papers/Gurses-CPDP11.pdf> (accessed: 18/12/2019).
- [54] Marit Hansen, Meiko Jensen, and Martin Rost. Protection goals for privacy engineering. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 159–166. IEEE, 2015.
- [55] Denis Hatebur, Maritta Heisel, Jan Jürjens, and Holger Schmidt. Systematic development of UMLsec design models based on security requirements. In *International Conference on Fundamental Approaches to Software Engineering*, pages 232–246. Springer, 2011.
- [56] Constance L. Heitmeyer, Myla Archer, Elizabeth I. Leonard, and John McLean. Applying formal methods to a certifiably secure software system. *IEEE Trans. Software Eng.*, 34(1):82–98, 2008.
- [57] Jennifer Horkoff and Eric Yu. Finding solutions in goal models: an interactive backward reasoning approach. In *International Conference on Conceptual Modeling*, pages 59–75. Springer, 2010.
- [58] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. Eliciting Security Requirements and Tracing them to Design: an integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering*, 15(1):63–93, 2010.
- [59] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge university press, 2004.
- [60] ISO and IEC. Common Criteria For Information Technology Security Evaluation - Part 2 Security Functional Components. 2017. URL <https://www.commoncriteriaportal.org/cc/> (accessed: 16/12/2019).
- [61] Jan Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In *IFIP International Information Security Conference*, pages 93–107. Springer, 2001.



- [62] Jan Jürjens. *Secure systems development with UML*. Springer Science & Business Media, 2005.
- [63] Jan Jürjens and Pasha Shabalin. Tools for Secure Systems Development with UML. *International Journal on Software Tools for Technology Transfer*, 9(5-6):527–544, 2007.
- [64] Jan Jürjens and Guido Wimmel. Formally testing fail-safety of electronic purse protocols. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 408–411. IEEE, 2001.
- [65] Christos Kalloniatis, Evangelia Kavakli, and Stefanos Gritzalis. Addressing privacy requirements in system design: the PriS method. *Requirements Engineering*, 13(3):241–255, 2008.
- [66] Basel Katt, Matthias Gander, Ruth Breu, and Michael Felderer. Enhancing model driven security through pattern refinement techniques. In *International Symposium on Formal Methods for Components and Objects*, pages 169–183. Springer, 2011.
- [67] Alan Kennedy, Kennedy Carter, William Frank, and Domain Architects. MDA Guide Version 1.0. Technical report, 2003.
- [68] Lano Kevin, Clark David, and Androustopoulos Kelly. Safety and Security Analysis of Object-Oriented Models. In *International Conference on Computer Safety, Reliability, and Security*, pages 82–93. Springer, 2002.
- [69] Minseong Kim, Sooyong Park, Vijayan Sugumaran, and Hwasil Yang. Managing Requirements Conflicts In Software Product Lines: A Goal and Scenario Based Approach. *Data & Knowledge Engineering*, 61(3):417–432, 2007.
- [70] Ralf Kollmann and Martin Gogolla. Capturing dynamic program behaviour with UML collaboration diagrams. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, pages 58–67. IEEE, 2001.
- [71] Dexter Kozen. Results on the propositional  $\mu$ -calculus. *Theoretical computer science*, 27(3):333–354, 1983.
- [72] Abdelouahed Kriouile, Najiba Addamssiri, Taoufiq Gadi, and Youssef Balouki. Getting the static model of PIM from the CIM. In *Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in*, pages 168–173. IEEE, 2014.
- [73] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

- [74] Sabine Kuske, Martin Gogolla, Ralf Kollmann, and Hans-Jörg Kreowski. An integrated semantics for UML class, object and state diagrams based on graph transformation. In *International Conference on Integrated Formal Methods*, pages 11–28. Springer, 2002.
- [75] Wadha Labda, Nikolay Mehandjiev, and Pedro Sampaio. Modeling of privacy-aware business processes in BPMN to protect personal data. In *ACM Symposium on Applied Computing*, pages 1399–1405. ACM, 2014.
- [76] Leen Lambers, Daniel Strüber, Gabriele Taentzer, Kristopher Born, and Jevgenij Huebert. Multi-Granular Conflict and Dependency Analysis in Software Engineering based on Graph Transformation. In *International Conference on Software Engineering*. IEEE/ACM, 2018. to appear.
- [77] Leen Lambers, Kristopher Born, Jens Kosiol, Daniel Strüber, and Gabriele Taentzer. Granularity of conflicts and dependencies in graph transformation systems: A two-dimensional approach. *Journal of logical and algebraic methods in programming*, 103:105–129, 2019.
- [78] Kevin Lano, David Clark, and Kelly Androutsopoulos. Safety and security analysis of object-oriented models. In *Computer Safety, Reliability and Security, 21st International Conference, SAFECOMP 2002, Catania, Italy, September 10-13, 2002, Proceedings*, pages 82–93, 2002.
- [79] Yves Ledru, Jean-Luc Richier, Akram Idani, and Mohamed-Amine Labiadh. From KAOS to RBAC: A case study in designing access control rules from a requirements analysis. In *Conference on Network and Information Systems Security*, pages 1–8. IEEE, 2011.
- [80] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, July 2006.
- [81] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *International Conference on the Unified Modeling Language*, pages 426–441. Springer, 2002.
- [82] Curtis L. Maines, David Llewellyn-Jones, Stephen Tang, and Bo Zhou. A cyber security ontology for BPMN-security extensions. In *International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing*, pages 1756–1763. IEEE, 2015.
- [83] Dewi Mairiza and Didar Zowghi. An Ontological Framework to Manage the Relative Conflicts between Security and Usability Requirements. In *Managing Requirements Knowledge (MARK), 2010 Third International Workshop on*, pages 1–6. IEEE, 2010.

- [84] Dewi Mairiza, Didar Zowghi, and Nur Nurmuliani. Towards a catalogue of conflicts among non-functional requirements. In *International Conference on Evaluation of Novel Approaches to Software Engineering*. SciTePress, 2010.
- [85] Salvador Martínez, Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, and Jordi Cabot. Model-driven extraction and analysis of network security policies. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 52–68. Springer, 2013.
- [86] Salvador Martínez, Valerio Cosentino, and Jordi Cabot. Model-based analysis of Java EE web security configurations. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, pages 55–61. ACM, 2016.
- [87] Rene Meis and Maritta Heisel. Systematic Identification Of Information Flows From Requirements To Support Privacy Impact Assessments. In *International Joint Conference on Software Technologies*, volume 2, pages 1–10. IEEE, 2015.
- [88] Ricardo Mendes and João P. Vilela. Privacy-Preserving Data Mining: Methods, Metrics, and Applications. *IEEE Access*, 5:10562–10582, 2017.
- [89] Michael Menzel, Ivonne Thomas, and Christoph Meinel. Security requirements specification in service-oriented business process management. In *International Conference on Availability, Reliability and Security*, pages 41–48. IEEE, 2009.
- [90] Austin Mohr. A survey of zero-knowledge proofs with applications to cryptography. *Southern Illinois University, Carbondale*, pages 1–12, 2007.
- [91] Daniel Moody. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.
- [92] Anthony Morton and Angela Sasse. Privacy is a process, not a PET: A theory for effective privacy practice. In *Proceedings of the 2012 workshop on New security paradigms*, pages 87–104. ACM, 2012.
- [93] Haralambos Mouratidis. *A security oriented approach in the development of multiagent systems: applied to the management of the health and social care needs of older people in England*. PhD thesis, University of Sheffield, 2004.
- [94] Haralambos Mouratidis and Jan Jürjens. From goal-driven security requirements engineering to secure design. *International Journal of Intelligent Systems*, 25(8):813–840, 2010.
- [95] Haralambos Mouratidis, Christos Kalloniatis, Shareeful Islam, Marc-Philippe Huget, and Stefanos Gritzalis. Aligning Security and Privacy to Support the

- Development of Secure Information Systems. *Journal of Universal Computer Science*, 18(12):1608–1627, 2012.
- [96] Jutta Mülle, Silvia von Stackelberg, and Klemens Böhm. *A security language for BPMN process models*. KIT, Fakultät für Informatik, 2011.
- [97] Phu H Nguyen, Koen Yskout, Thomas Heyman, Jacques Klein, Riccardo Scandariato, and Yves Le Traon. SoSPa: A system of Security design Patterns for systematically engineering secure systems. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 246–255. IEEE, 2015.
- [98] Martín Ochoa, Jan Jürjens, and Daniel Warzecha. A sound decision procedure for the compositionality of secrecy. In *International Symposium on Engineering Secure Software and Systems*, pages 97–105. Springer, 2012.
- [99] Elda Paja, Fabiano Dalpiaz, and Paolo Giorgini. Managing Security Requirements Conflicts in Socio-Technical Systems. In *International Conference on Conceptual Modeling*, pages 270–283. Springer, 2013.
- [100] Liliana Pasquale, Paola Spoletini, Mazeiar Salehie, Luca Cavallaro, and Bashar Nuseibeh. Automating trade-off analysis of security requirements. *Requirements Engineering*, 21(4):481–504, 2016.
- [101] Dino Pedreschi, Salvatore Ruggieri, and Franco Turini. Integrating Induction and Deduction for Finding Evidence of Discrimination. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law*, pages 157–166. ACM, 2009.
- [102] Sven Peldszus, Katja Tuma, Daniel Strüber, Jan Jürjens, and Riccardo Scandariato. Secure data-flow compliance checks between models and code based on automated mappings. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 23–33. IEEE, 2019.
- [103] Andreas Pfitzmann and Marit Hansen. A Terminology For Talking About Privacy by Data Minimization: Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity Management. *TU Dresden and ULD Kiel, Tech. Rep*, 2011. URL [http://www.maroki.de/pub/dphistory/2010\\_Anon\\_Terminology\\_v0.34.pdf](http://www.maroki.de/pub/dphistory/2010_Anon_Terminology_v0.34.pdf) (accessed:18/12/2019).
- [104] Amir Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 46–57. IEEE, 1977.
- [105] Eltjo R. Poort and Peter de With. Resolving Requirement Conflicts through Non-Functional Decomposition. In *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pages 145–154. IEEE, 2004.

- [106] Pille Pullonen, Raimundas Matulevičius, and Dan Bogdanov. PE-BPMN: privacy-enhanced business process model and notation. In *International Conference on Business Process Management*, pages 40–56. Springer, 2017.
- [107] Jean-Pierre Queille and Joseph Sifakis. Fairness and related properties in transition systems—a temporal logic to deal with fairness. *Acta Informatica*, 19(3):195–220, 1983.
- [108] Qusai Ramadan, Mattia Salnitri, Daniel Strüber, Jan Jürjens, and Paolo Giorgini. From Secure Business Process Modeling to Design-level Security Verification. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 123–133. IEEE, 2017.
- [109] Qusai Ramadan, Amir Shayan Ahmadian, Daniel Strüber, Jan Jürjens, and Steffen Staab. Model-based discrimination analysis: a position paper. In *Proceedings of the International Workshop FairWare@ICSE 2018, Gothenburg, Sweden, 2018*.
- [110] Qusai Ramadan, Mattia Salnitri, Daniel Strüber, Jan Jürjens, and Paolo Giorgini. Integrating BPMN-and UML-based Security Engineering via Model Transformation. *Software Engineering und Software Management*, 2018.
- [111] Qusai Ramadan, Daniel Strüber, Mattia Salnitri, Volker Riediger, and Jan Jürjens. Detecting Conflicts Between Data-Minimization and Security Requirements in Business Process Models. In *European Conference on Modelling Foundations and Applications*, pages 179–198. Springer, 2018.
- [112] Qusai Ramadan, Amir Shayan Ahmadian, Jan Jürjens, Steffen Staab, and Daniel Strüber. Explaining Algorithmic Decisions with respect to Fairness. *Software Engineering and Software Management*, 2019.
- [113] Qusai Ramadan, Marco Konersmann, Amir Shayan Ahmadian, Jan Jürjens, and Steffen Staab. Analyzing Individual Fairness based on Software Design Models. 2019. *Submitted*. A long version of the submitted paper is available online at <https://figshare.com/s/c7f3c5fb9b337595a6a4>.
- [114] Qusai Ramadan, Daniel Strüber, Mattia Salnitri, Jürjens Jan Riediger, Volker, and Steffen Staab. A Semi-Automated BPMN-based Framework for Detecting Conflicts between Security, Data-Minimization and Fairness Requirements. 2019. *To appear* in a special journal issue in SoSyM: Software and Systems Modeling.
- [115] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*, pages 10–29. Springer, 2001.

- [116] Yassine Rhazali, Youssef Hadi, and Abdelaziz Mouloudi. Transformation Approach CIM to PIM: From Business Processes Models to State Machine and Package Models. In *2015 International Conference on Open Source Software Computing (OSSCOM)*, pages 1–6. IEEE, 2015.
- [117] Yassine Rhazali, Youssef Hadi, and Abdelaziz Mouloudi. A New Methodology CIM to PIM Transformation Resulting from an Analytical Survey. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, pages 266–273, 2016.
- [118] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini. Analysis-level classes from secure business processes through model transformations. In *International Conference on Trust, Privacy and Security in Digital Business*, pages 104–114. Springer, 2007.
- [119] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini. A BPMN extension for the modeling of security requirements in business processes. *IEICE transactions on information and systems*, 90(4):745–752, 2007.
- [120] Alfonso Rodríguez, Eduardo Fernández-Medina, Juan Trujillo, and Mario Piattini. Secure business process model specification through a UML 2.0 activity diagram profile. *Decision Support Systems*, 51(3):446–465, 2011.
- [121] Salvatore Ruggieri, Dino Pedreschi, and Franco Turini. Data Mining for Discrimination Discovery. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(2):9, 2010.
- [122] Salvatore Ruggieri, Sara Hajian, Faisal Kamiran, and Xiangliang Zhang. Anti-discrimination Analysis Using Privacy Attack Strategies. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 694–710. Springer, 2014.
- [123] Mehrdad Saadatmand and Sahar Tahvili. A Fuzzy Decision Support Approach for Model-Based Tradeoff Analysis of Non-Functional Requirements. In *2015 12th International Conference on Information Technology-New Generations (ITNG)*, pages 112–121. IEEE, 2015.
- [124] Muhammad Qaiser Saleem, Jafreezal Jaafar, and Mohd Fadzil Hassan. A domain-specific language for modelling security objectives in a business process models of soa applications. *AISS*, 4(1):353–362, 2012.
- [125] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. Are students representatives of professionals in software engineering experiments? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, volume 1, pages 666–676. IEEE, 2015.

- [126] Mattia Salnitri. *Secure Business Process Engineering: a socio-technical approach*. PhD thesis, University of Trento, 2016.
- [127] Mattia Salnitri and Paolo Giorgini. Transforming Socio-Technical Security Requirements in SecBPMN Security Policies. In *iStar*, 2014.
- [128] Mattia Salnitri and Paolo Giorgini. Modeling and verification of ATM security policies with SecBPMN. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pages 588–591. IEEE, 2014.
- [129] Mattia Salnitri, Fabiano Dalpiaz, and Paolo Giorgini. Modeling and verifying security policies in business processes. In *Enterprise, Business-Process and Information Systems Modeling*, pages 200–214. Springer, 2014.
- [130] Mattia Salnitri, Elda Paja, and Paolo Giorgini. From Socio-Technical Requirements to Technical Security Design: an STS-based Framework. *DISI-University of Trento*, 2015.
- [131] Mattia Salnitri, Elda Paja, and Paolo Giorgini. Maintaining secure business processes in light of socio-technical systems' evolution. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 155–164. IEEE, 2016.
- [132] Mattia Salnitri, Mahdi Alizadeh, Daniele Giovanella, Nicola Zannone, and Paolo Giorgini. From security-by-design to the identification of security-critical deviations in process executions. In *International Conference on Advanced Information Systems Engineering*, pages 218–234. Springer, 2018.
- [133] Kurt Schneider, Eric Knauss, Siv Houmb, Shareeful Islam, and Jan Jürjens. Enhancing security requirements engineering by organizational learning. *Requirements Engineering*, 17(1):35–56, 2012.
- [134] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE software*, 20(5):42–45, 2003.
- [135] Mary Shaw. The coming-of-age of software architecture research. In *Proceedings of the 23rd international conference on Software engineering*, page 656. IEEE Computer Society, 2001.
- [136] Guttorm Sindre. Mal-activity diagrams for capturing attacks on business processes. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 355–366. Springer, 2007.
- [137] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requirements engineering*, 10(1):34–44, 2005.

- [138] Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Kwiatkowska, John McDermid, and Richard Paige. Large-scale complex IT systems. *Commun. ACM*, 55(7):71–77, 2012.
- [139] Sarah Spiekermann and Lorrie Faith Cranor. Engineering privacy. *IEEE Transactions on software engineering*, 35(1):67–82, 2009.
- [140] Steffen Staab, Sophie Stalla-Bourdillon, and Laura Carmichael. Observing and recommending from a social web with biases. *arXiv preprint arXiv:1604.07180*, 2016. URL <https://arxiv.org/abs/1604.07180> (accessed: 18/12/2019).
- [141] Harald Störrle. How are conceptual models used in industrial software development?: A descriptive survey. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 160–169. ACM, 2017.
- [142] Daniel Strüber, Kristopher Born, Kanwal Daud Gill, Raffaella Groner, Timo Kehrer, Manuel Ohrndorf, and Matthias Tichy. Henshin: A usability-focused framework for emf model transformation development. In *International Conference on Graph Transformations*, pages 125–141, 2017.
- [143] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):571–588, 2002.
- [144] Paolo Tonella and Alessandra Potrich. Reverse engineering of the interaction diagrams from C++ code. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pages 159–168. IEEE, 2003.
- [145] Florian Tramèr, Vaggelis Atlidakis, Roxana Geambasu, Daniel J. Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. Discovering Unwarranted Associations in Data-Driven Applications with the Fairtest Testing Toolkit. *CoRR*, *abs/1510.02377*, 2015.
- [146] GW Van Blarckom, John J Borking, and JG Eddy Olk. Handbook Of Privacy and Privacy-Enhancing Technologies. *Privacy Incorporated Software Agent (PISA) Consortium, The Hague*, 2003.
- [147] Axel Van Lamsweerde. *Requirements engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons, 2009.
- [148] Axel Van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE transactions on Software engineering*, 24(11):908–926, 1998.



- 
- [149] Sahil Verma and Julia Rubin. Fairness Definitions Explained. In *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, pages 1–7. IEEE, 2018.
- [150] José L. Vivas, José A. Montenegro, and Javier López. Towards a business process-driven framework for security engineering with the UML. In *International Conference on Information Security*, pages 381–395. Springer, 2003.
- [151] Christian Wolter and Andreas Schaad. Modeling of task-based authorization constraints in BPMN. *Business process management*, pages 64–79, 2007.
- [152] Yijun Yu, Jan Jürjens, and Jorg Schreck. Tools for traceability in secure software development. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 503–504. IEEE, 2008.
- [153] Tal Zarsky. The trouble with algorithmic decisions: An analytic road map to examine efficiency and fairness in automated and opaque decision making. *Science, Technology, & Human Values*, 41(1):118–132, 2016.
- [154] Lu Zhang and Xintao Wu. Anti-discrimination Learning: a causal modeling-based framework. *International Journal of Data Science and Analytics*, 4(1), 2017.
- [155] Lu Zhang, Yongkai Wu, and Xintao Wu. On Discrimination Discovery Using Causal Networks. In *International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation*, pages 83–93. Springer, 2016.
- [156] Tewfik Ziadi, Marcos Aurélio Almeida da Silva, Lom Messan Hillah, and Mikal Ziane. A fully dynamic approach to the reverse engineering of uml sequence diagrams. In *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 107–116. IEEE, 2011.
- [157] Indre Zliobaite. A survey on measuring indirect discrimination in machine learning. *arXiv preprint arXiv:1511.00148*, 2015. URL <https://arxiv.org/abs/1511.00148> (accessed: 18/12/2019).



