

A Semi-Automated BPMN-based Framework for Detecting Conflicts between Security, Data-Minimization and Fairness Requirements.

Qusai Ramadan · Daniel Strüber · Mattia Salnitri · Jan Jürjens · Volker Riediger · Steffen Staab.

Received: date / Accepted: date

Abstract Requirements are inherently prone to conflicts. Security, data-minimization and fairness requirements are no exception. 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. For example, a process may require anonymous execution of a task that writes data into a secure data storage, where the identity of the writer is needed for the purpose of accountability. Moreover, con-

flicts not arise from trade-offs between requirements elicited from the stakeholders, but also from misinterpretation of elicited requirements while implementing them in business processes, leading to a non-alignment between the data subjects' requirements and their specifications. Both types of conflicts are substantial challenges for conflict detection.

To address these challenges, we propose a BPMN-based 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 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 new. For formulating our 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.

Keywords Conflicts · Requirements Engineering · Security · Data Minimization · Fairness · BPMN

This research was partially supported by: (i) The Deutscher Akademischer Austauschdienst (DAAD), (ii) the project "Engineering Responsible Information Systems" financed by the University of Koblenz-Landau.

Q. Ramadan (✉)
University of Koblenz-Landau, Koblenz, Germany
E-mail: qramadan@uni-koblenz.de

D. Strüber
Chalmers University | University of Gothenburg, Gothenburg, Sweden
E-mail: danstru@chalmers.se

M. Salnitri
Politecnico di Milano, Milano, Italy
E-mail: mattia.salnitri@polimi.it

J. Jürjens
University of Koblenz-Landau, Koblenz, Germany
Fraunhofer-Institute for Software and Systems Engineering ISST, Dortmund, Germany
E-mail: <http://jan.jurjens.de>

V. Riediger
University of Koblenz-Landau, Koblenz, Germany
E-mail: riediger@uni-koblenz.de

S. Staab
Universität Stuttgart, Stuttgart, Germany
University of Southampton, UK
E-mail: s.r.staab@soton.ac.uk

1 Introduction

Restrictions on data collection and data usage play a central role in data protection regulations and legal frameworks. The 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* [22] and *data-misuse* concerns [21].

Key data protection concepts are *data minimization* [32, 72] and *fairness* [8, 19]. 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 [56], 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 *protected data* as defined by laws or organizational policies [4, 12]. For example, *Article 22(4)* of the European General Data Protection Regulation (GDPR, [5]) prohibits decision making based on *protected data* as defined in *Article 9* of the same regulation, such as ethnicity, religion, and gender.

Apart from 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 to avoid fairness and privacy threats, respectively. Data-minimization concepts were first defined by Pfitzmann et al. [56] and later included in the ISO 15408 standard of common criteria for information technology security evaluation [36]. The two fairness concepts are formally explored by many research works in the algorithmic fairness field [8, 28].

While privacy-enhancing technologies [75] and algorithmic fairness techniques [78] respectively address specific data-minimization and fairness needs, security, privacy and fairness violations often do not come from loopholes in the applied protection technologies [33], but from conflicts between data protection interests at the business level of the target system [10, 30–33, 51]. For example, according to recent research from Google AI [31], one of the main challenges in the field of the algorithmic fairness is conflicts between data-minimization and fairness requirements.

The variety of requirements arising from security, data protection, and fairness considerations give rises to various types of conflicts. Two main sources of such conflicts are clashes between needs and misinterpretation of requirements.

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 data 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 [12].

Misinterpretation of requirements. Misinterpretation of the elicited requirements can happen intentionally or unintentionally while implementing the requirements in business processes. 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 Fairness*). To fulfill this requirement, a business analyst may specify that the gender attribute should not be accessed by the decision-making activity. However, this solution does not prevent *indirect discrimination* [21], which may arise from the availability of data that are highly correlated with gender, such as a person's first name. In fact, to prevent indirect discrimination, it might be necessary to access a protected data attribute and use it to identify correlated data.

Challenges. 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 at the elicitation phase without analyzing conflicts between them [13, 22, 37, 52]. 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 conflicting. 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, as far as our knowledge goes, no approach supports fairness requirements in the early stages of system development. Detecting conflicts between fairness and other data protection requirements in the early stages of the system development is, therefore, currently not possible. Although a textual syntax with precise semantics amenable to automatic analysis may be a solution to address the above

challenges, 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. Moreover, a graphical solution helps to manage the complexity of the models of large-scale real-world systems.

Our previous contributions. To the best of our knowledge, apart from our earlier work [62], there exist no other approaches to detect conflicts between data protection requirements in the design of a given system. In [62], we proposed an extension of the Business Process Modeling Notation (BPMN 2.0, [1]) supporting: (i) the specification of security and data-minimization requirements in BPMN models, (ii) the detection of conflicts between security and data-minimization requirements based on a catalog of domain-independent anti-patterns. We reused the security annotations from an existing security-oriented BPMN extension called SecBPMN2 [68]. To express the anti-patterns, we extended a graphical query language for BPMN 2.0 models called SecBPMN2-Q [68].

Our contribution in this paper. This paper presents an extension of our earlier conference paper [62] in three 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 addition to data minimization and security, we also address conflicts related to fairness, which represents one of the main challenges in the field of algorithmic fairness [31]. Third, we performed a more comprehensive experimental evaluation, involving a larger set of participants and contrasting subjective usefulness assessments with objective performance measurements. Specifically, our contributions in this paper are:

- a *semi-automated process* for supporting: (i) the enrichment of 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*,
- an extension to the catalog of *the domain-independent anti-patterns* in [62] to support the automatic conflict detection of among security, fairness, and data-minimization requirements, where each anti-pattern represents a conflict or potential conflict,
- 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
- a *user evaluation*, in which we studied the usefulness of our conflict detection technique in an experiment with 30 participants.

The paper is organized as it follows. Sec. 2 provides the necessary background. Sec. 3 describes our proposed framework. Sec. 4 introduces our BPMN extension. Sec. 5 presents our alignment verification approach. Sec. 6 presents our conflict detection approach. Sec. 7 presents the tool support for our approach. Sec. 8 and 9 are devoted to the validation based on a case study and a user evaluation. Sec. 10, 11, and 12 discuss limitations of our current approach, survey related work, and conclude, respectively.

2 Background

We introduce the fundamental data-minimization and fairness concepts used in our work, and a BPMN-oriented security engineering approach whose security concepts we reused.

Data-minimization concepts. Pfitzmann et al. [56] define five data minimization concepts that can be refined into privacy requirements for the target system [13, 22, 37, 52]. In the following, the five data-minimization concepts that are considered in our work are listed, each with its definition as provided in [56]: (i) *Anonymity* is the inability of an adversary to sufficiently identify a subject within a set of subjects, called the anonymity set. (ii) *Pseudonymity* 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. (iii) *Unlinkability* 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 or not. Although it is not explicitly mentioned in [56], the definition of unlinkability implies that the two or more IOIs are of comparable types, otherwise it is infeasible to make the comparison ([22], p.8). (iv) *Undetectability* is the inability of an adversary to sufficiently distinguish whether an IOI exists or not. By the definition [56], 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). (v) *Unobservability* 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).











Fairness concepts. Although there is no agreement in the literature on the definition of fairness, two main types of fairness are distinguished [8]: we have *Individual fairness* if an activity with a decision-making purpose produces the same output for every two data subjects whose data are identical excerpt for data that have been defined as protected data. We have *Group fairness* if a decision-making activity 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 data are not limited to those listed as protected by the laws and regulations. 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 data in *Article 9* of the GDPR [5], 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 data in a decision-making activity does not necessarily ensure fairness. Other data may act as *proxies* for protected data due to correlations between them, and as a consequence, may lead to *indirect discrimination* [20]. For example, in 2016, it was found that a decision-making software by Amazon.com, Inc., excluded minority neighborhoods with the African American community in the United States from being able to participate in a delivery-free service, although the software did not explicitly use *ethnicity* for making the decisions [35]. The decision on whether a piece of data is correlated with a protected characteristic or not depends on correlation metrics and a threshold value that can be specified by domain experts. On the other hand, discrimination on the ground of protected data 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* [74].

BPMN-based security engineering. Modeling security requirements during the design phase of the business processes models is a promising research direction in the field of security engineering [43]. The key idea is to extend graphical business process modeling languages such as BPMN [1] to support the modeling and analysis of security requirements.

Despite the availability of various BPMN-based security extensions [43], support for data-minimization and fairness requirements in business process models is sparse. For data-minimization requirements, we are only aware of the work proposed in [64], which considers one data-minimization requirement, namely anonymity. However, fairness and further data-minimization requirements such as unlinkability and undetectability were not addressed yet. Readers interested in a comprehensive overview of the concepts that have been considered by previous BPMN security extensions are referred to Maines et al.'s survey in [43].

To capture conflicts between security, data-minimization, and fairness requirements, a unified framework for modeling these types of requirements is needed. Compared to other approaches, we found that the security concepts from SecBPMN2 [68] offer the following advantages: (i) in contrast to the research work in [11, 14, 39, 48, 64, 66, 77] which support only a restricted set of security aspects, SecBPMN2 enriches BPMN 2.0 modeling language with 10 security-specific concepts. In what follows, we list the 10 security concepts that are supported by SecBPMN2, each with its corresponding annotation and definition as provided in [68]: *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 keep 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*  and *Separation of duties*  requires that the same person or different persons should be responsible for the completion of two related tasks, respectively. *Confidentiality*  and *Integrity*  indicate that only authorized users are allowed to read or modify data from a given activity, message flow, or data object, respectively. *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.

Reusing these concepts allows us to study interactions between a comprehensive set of security, data-minimization, and fairness requirements, enabling a powerful approach to conflict detection. (ii) While other works [53, 76] use textual stereotypes to enrich business process models with security requirements (e.g., «confidentiality»), SecBPMN2, as illustrated in the previous paragraph, represents security requirements using graphical annotations [68]. Compared to textual annotations, graphical ones can reduce the cognitive complexity for understanding the resulting business process models [50], and as a result, contribute to usability.

In addition, (iii) SecBPMN2 provides a security query language for specifying queries that can be matched against a given SecBPMN2 model, called SecBPMN2-Q [68]. 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 the system to be. Second, specifying domain-independent conflicts between security,

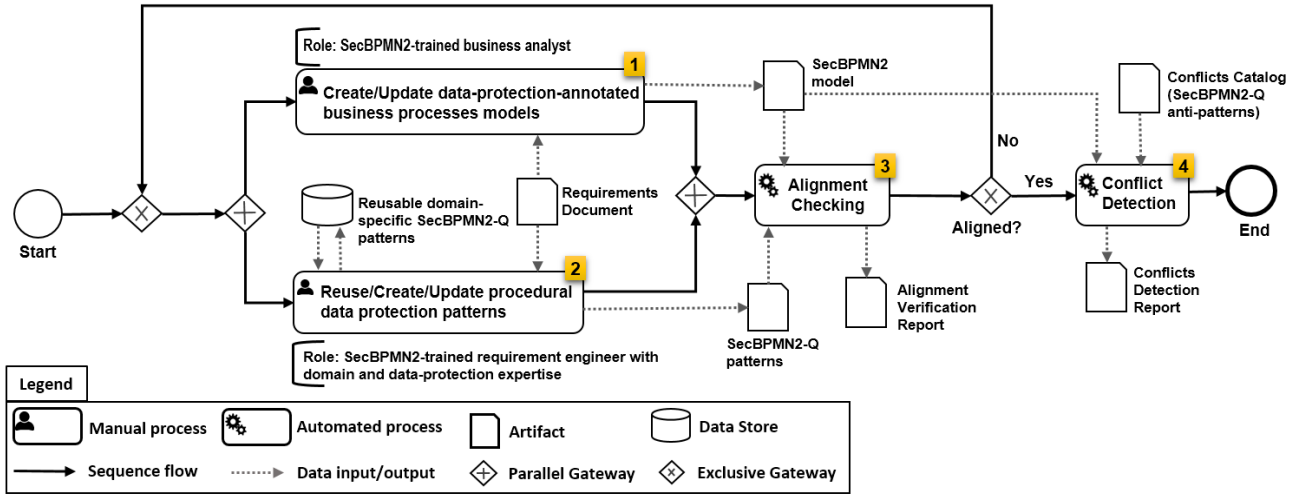


Fig. 1: The semi-automated conflict detection BPMN-based process proposed in this paper.

data-minimization, and fairness requirements as anti-patterns that can be used later for uncovering conflicts in SecBPMN2 models. To this end, in what follows, the *data-protection* concept, otherwise it is mentioned, is used to refer to all security, data-minimization, and fairness concepts that are considered in this paper.

For query evaluation, SecBPMN2-Q uses an artificial intelligence system based on disjunctive logic programming, DLV [42]. In particular, it is based on planner functionality called DLV- \mathcal{K} [25]. The planner uses a knowledge base and defines possible plans to be executed in order to achieve given objectives. In the case of SecBPMN2, the knowledge base consists of the business process(es) analyzed, while the objectives are the 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 pattern is matched. This solution allows overcoming performance issues due to the possible very complex design of business processes.

3 Process

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 [62]. As mentioned in Sec. 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 [68]. These deviations may result in unexpected conflicts between the requirements and, as a consequence,

lead to security or privacy vulnerabilities. This is not a new conclusion but a fundamental expectation. For example, Kim et al. [38] define requirements conflicts in software engineering 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 Fig. 1, that 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 Sec. 1).

Roles and assumptions about their skills. The process proposed in this paper 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 requirements that aim to prevent unauthorized access and modification for system resources. Moreover, since business projects within the same domain can share many procedural business practices [27], 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 trained on using our proposed extension of SecBPMN2. Still, to ensure the correct use of our approach, some additional background about the data protection concepts that are used in our approach on top of their training 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 [34]. 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. 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. The conflicts catalog proposed in this paper extends our conflicts catalog proposed in [62] with anti-patterns for expressing conflicts between fairness and data-minimization requirements. Details of our catalog are described in Sec. 6.

Outputs. As shown in Fig. 1, 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.

Our process consists of four phases. The numbers in Fig. 1 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 of our extension to the SecBPMN2 language is provided in Sec. 4.

Phase 2. In this phase, requirements engineers: first, identify the set of domain-specific patterns to be reused for alignment checking for the business process model in question. This step can be performed by mapping data protection requirements extracted from the *requirements document* to existing domain-specific patterns. Second, design patterns for those data protection 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 [68]. 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 Fig. 1, 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 Fig. 1, *phase 1* and *phase 2* can be executed in parallel. Details description of our extension to the SecBPMN2-Q is given in Sec. 5.

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 analysts to

manually check whether the requirements are correctly specified in the business process models or not [71]. To support an automatic check, we extended the query engine of SecBPMN2-Q [68], 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.

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 Fig. 1 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. This process should be repeated until the alignment is ensured; after that *phase 4* can start. More details on alignment checking are given in Sec. 5.

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, due to the lack of experts and since the business process models frequently are composed of many tasks, a manual detection for conflicts between data protection requirements in the business process models is an 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 model 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 model 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 Fig. 1, 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 Sec. 6.

4 Modeling Security-, Data-Minimization- and Fairness-aware BPMN Models

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 new, while the security-specific elements are reused from SecBPMN2, an existing security-oriented BPMN extension. In the following, we first present a running example and then a complete description of our extension.

Running Example. Fig. 2 represents a business process in the context of healthcare management. A patient uses a tele-medicine 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* and *lanes* such as “Tele-medicine Device” and “System Portal”, respectively. Communication between pools is represented by *message flows*; the content of such communications is represented using *message*. For example, “Tele-medicine 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 “Tele-medicine 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 a data objects and targets a task, then the task reads the data object. If the *data association* targets the data objects then the task writes the data object. If the data object is connected to the same task with two *data associations* with different orientations, 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 this paper): (i) *start event*, which represents the initial point of a business process; (ii) *end event* which represents 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.

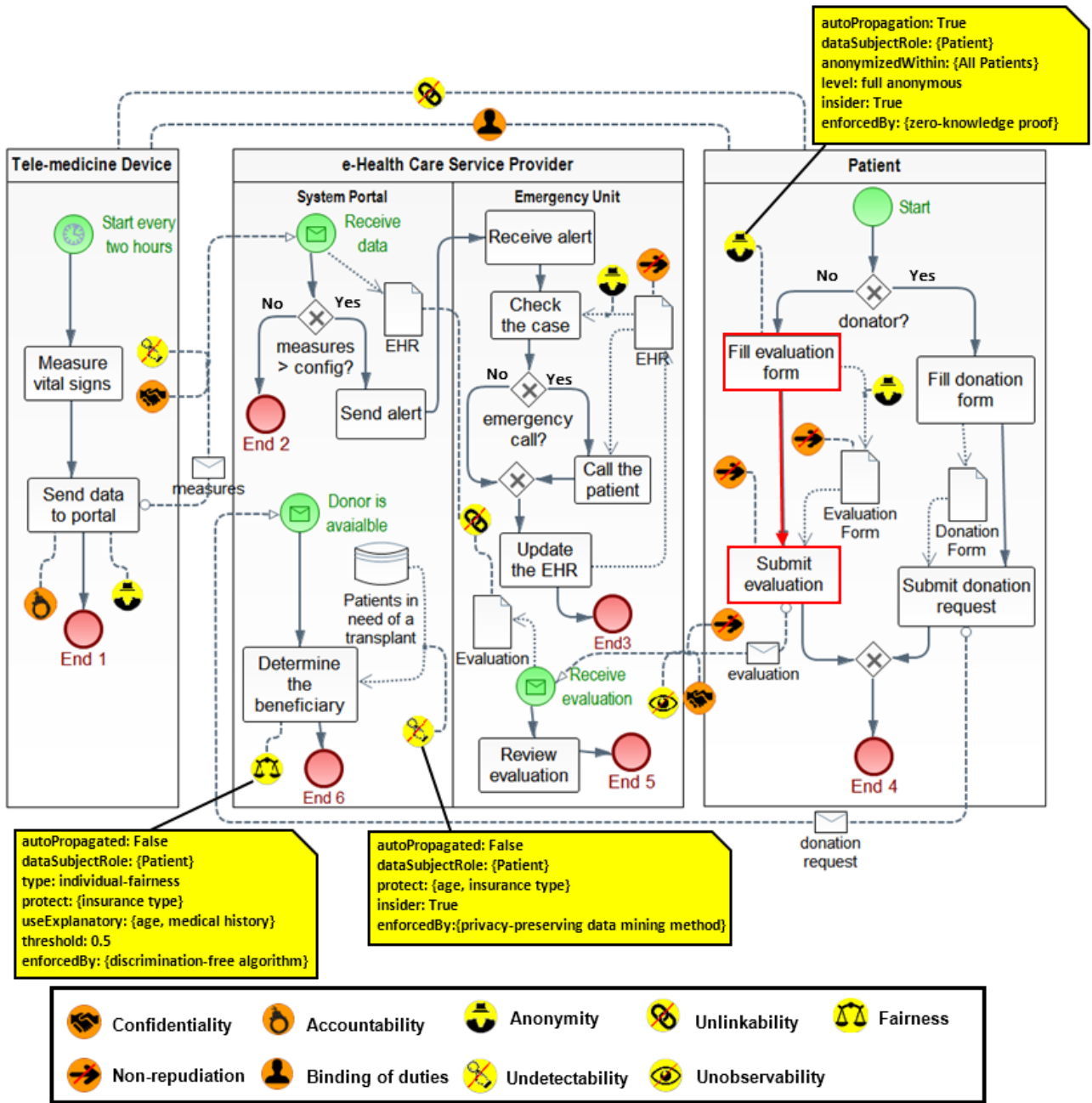


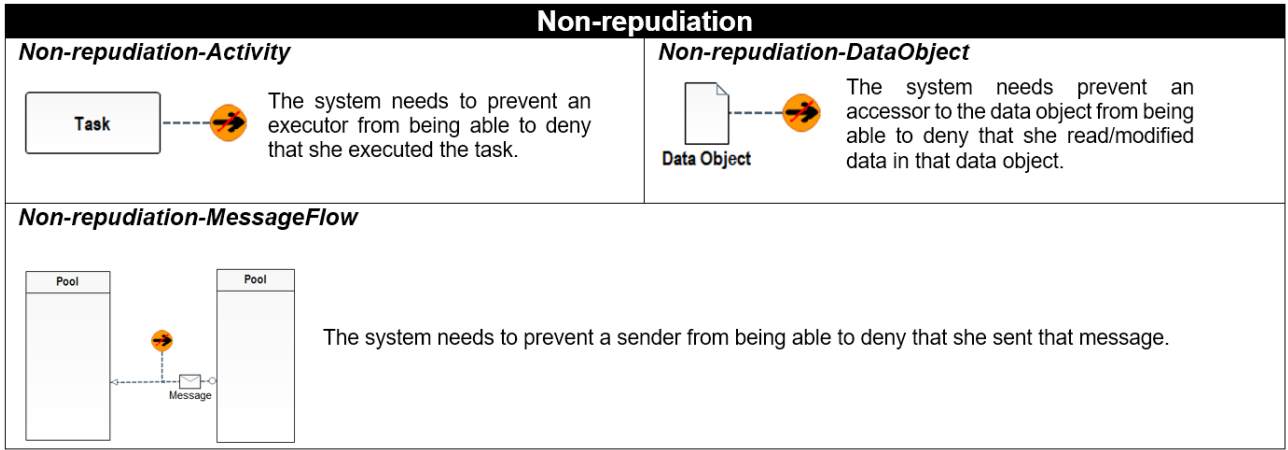
Fig. 2: Running example: Specifying security, data-minimization, and fairness requirements in a healthcare business process.

Gateways specify deviations of the execution sequence, they split the processes into two or more branches and allow to execute one branch based on a specified condition. For example, the gateway “measures \geq 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. SecBPMN2 supports 10 security-specific annotations. However, for the scope of this paper, we use only 4 of them, described below.

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 to tasks and expresses the need for monitoring the execution of the tasks. In Fig. 2, the *accountability* annotation that is associated to

Fig. 3: All variants of SecBPMN's *Non-repudiation* annotation.

the “Send data to portal” task meaning that the task’s executor must be monitored. In general, implementing such monitors will require a monitoring component that intercepts the calls to the tasks or store the logs of the execution of the task, 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. It 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 can be defined in one or more variants, depending on which element the annotation is connected. For example, *Non-repudiation* (➡) has three variants. The variants’ definitions are shown in Fig. 3.

Our new fairness and data-minimization concepts, discussed below, are represented with yellow annotations, expressing that these concepts are more directly related to privacy than security.

Data-minimization and fairness annotations. To allow annotating BPMN models with data-minimization and fairness requirements, we extended the *artifact* class from BPMN with five concepts, namely *anonymity* (👤), *undetectability* (🔍), *unlinkability* (🔗), *unobservability* (👁), and *fairness* (⚖). The first four concepts are data-minimization-specific. The meta-model of our BPMN extension with data-minimization and fairness concepts is shown in Fig. 4. Gray parts in the meta-model represent part of SecBPMN2 elements. White parts are new elements.

Since an additional concept described by Pfitzmann 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 Sec. 2, 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 al-

lows 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., full anonymous vs pseudonymous). Using one annotation to represent related concepts is recommended to reduce graphical complexity [43]. For the same purpose, we introduced a specific annotation for *unobservability*, although unobservability, by definition, can be achieved by preserving both anonymity and undetectability.

We designed the graphical syntax of these annotations by following Moody’s guidelines for increasing the usability of modeling languages [50]. 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 annotations, 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 business process 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 to prevent some annotations from being linked with wrong BPMN elements. Details description of the linkage constraints and the rationale beyond them are provided in Appendix (A).

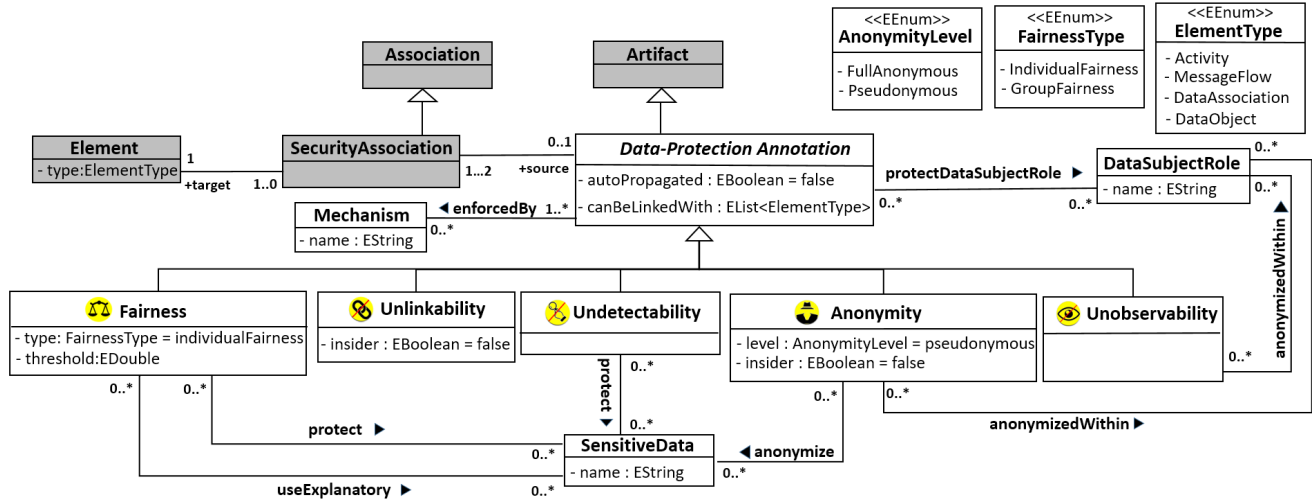


Fig. 4: Meta-model of our BPMN extension. Attributes show their default values.

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* and *explanatory* data, which are described in Sec. 2. 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 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 annotations proposed in this paper 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 Fig. 5. In the following, we describe the definitions and the attributes that can be used to shape the exact semantic of the annotation.

Anonymity, as shown in Fig. 5, comes in four variants. (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 Fig. 4, the following attributes can be used to shape the exact semantic of the anonymity annotation: the reference attribute *protectDataSubjectRole* 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., full 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.

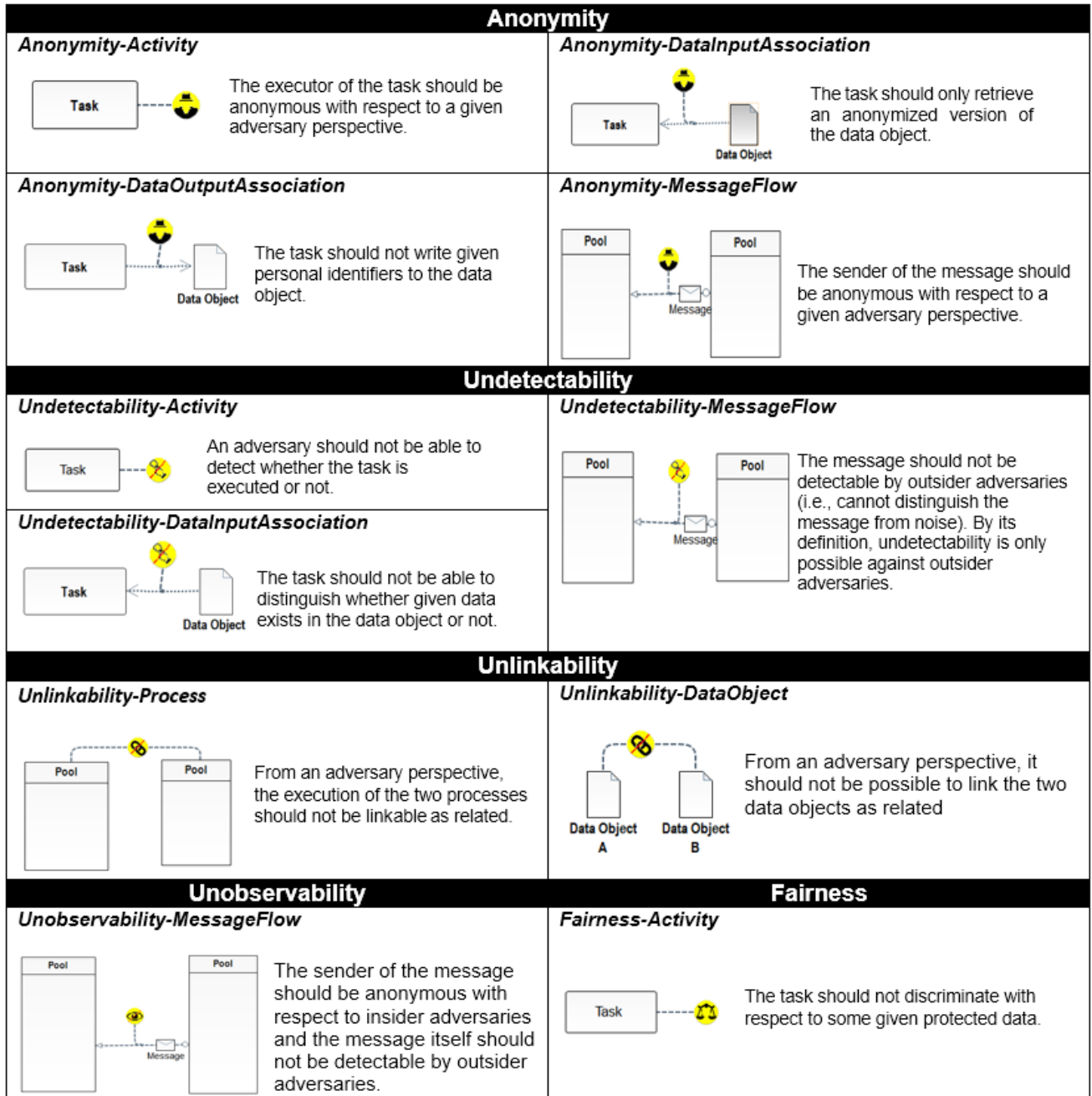


Fig. 5: All variants of our data-minimization and fairness annotations.

The example model Fig. 2 shows three anonymity annotations associated to different BPMN elements. Consider, for example, the one associated to 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 to use the “zero-knowledge proof” [49] mechanism to enforce anonymous authentication. Since the requirement is propagated, the same requirement applies to the “Submit evaluation” task.

Unlinkability, as shown in Fig. 5, comes in two variants. (i) *Unlinkability-Process* can be linked with two pools/lanes to specify that an adversary of the given type shall not be able to link 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 [56]. (ii) *Unlinkability-DataObject* can be linked with two data objects to specify that, from the given adversary perspective, it should not be possible to link the two data objects as related. Specifically, if linked to two data objects, this annotation specifies that the

two data objects should do not share data that 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 Fig. 4, the reference attribute *protectDataSubjectRole* can be used to specify the data subjects who should be protected from the linkability threats in term 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 Fig. 2 includes two unlinkability annotations. Consider, for example, 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)” [29] and the “k-anonymity” [73] mechanisms to enforce this requirement. We did not show these specifications in Fig. 2 to keep the model readable.

Undetectability, as shown in Fig. 5, comes in three variants. (i) *Undetectability-Activity* specifies that an adversary should not be able to detect whether an activity is executed. (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. (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 Fig. 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 to specify the predefined list of sensitive data that a conclusion about their existence in the data object in question should not be possible.

By definition ([56], 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 [56] defines the undetectability on 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 Fig. 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 using a “privacy-preserving data-mining method” (e.g., [47]) as a mechanism to enforce this requirement.

Unobservability 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 Fig. 4, the following reference attributes can be used to specify the exact semantic of the unobservability annotation: the attribute *protectDataSubjectRole* specifies all roles who 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 outsider adversaries should not be able to 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 using the “DC-networks” [18] and the “Dummy traffic” [63] as mechanisms to enforce this unobservability requirement.

Fairness, as shown in Fig. 5, can only be applied to activities, leading to precisely one variant called *Fairness*. This variant specifies that the activity should not discriminate between the data subjects based on a specific list of their sensitive data, called protected data. Based on the meta-model in Fig. 4, the exact semantic of this annotation can be shaped by the *protectDataSubjectRole*, *Protect* and *useExplanatory* reference attributes and the *type* attribute, as it follows.

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

Deciding when two pieces of data are correlated so 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 [74]. For example, the term $\text{Pr}(\text{insurance type} = \text{Private} \mid \text{job} = \text{Engineer})$ represents the probability that a person is privately-insured given that the person is working as an engineer. In contrast, the term $\text{Entropy}(\text{insurance type} = \text{Private} \mid \text{job} = \text{Engineer})$ 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 metric carefully.

The example model in Fig. 2, specifically the “System Portal” pool at the bottom, shows one fairness annotation associated with the “Determine the beneficiary” task. The attributes’ specifications of this annotation are shown in a yellow box linked with the fairness annotation in the model. 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., [16]) to enforce fairness and “0.5” as the threshold for when to consider a piece of data as a proxy for the “insurance type”.

5 Alignment Checking

In this section, we propose an extension to the SecBPMN2-Q 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:

- **Req1:** A telemedicine-device should execute the “Send data to portal” task anonymously by using pseudonyms.
- **Req2:** A telemedicine-device should be accountable when it performs the “Send data to portal” task.

If a business analyst unintentionally uses the anonymity level *full anonymous* instead of *pseudonymous* for specifying Req1 in Fig. 2, a conflict with Req2 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 Req2. Conflicts resulting from 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 that 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 mis-alignments.

Modeling SecBPMN2-Q patterns. SecBPMN2-Q supports custom queries enriched with security requirements that can be matched to SecBPMN2 models [68] 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.

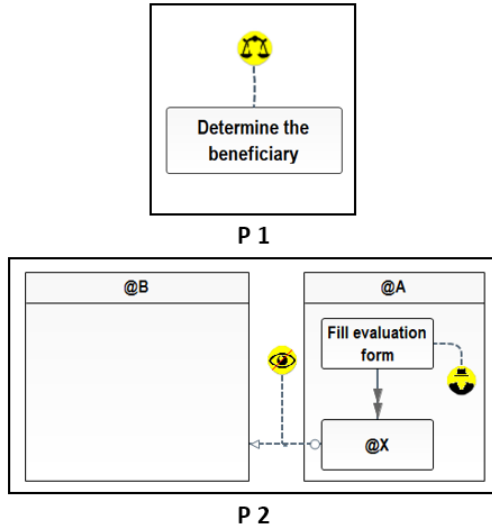


Fig. 6: Requirements specified as SecBPMN2-Q patterns.

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 Fig. 6.

Pattern P1 in Fig. 6 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]. List. 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 BPMN model in question.

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 [27]. For example, in Fig. 2, the procedural process that starts by “Fill donation form” task and ends by “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 Fig. 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.

Listing 1: Properties of *Fairness* in P1

```
autoPropagated=False,
dataSubjectRole={Patient},
type=individual fairness,
protect={insurance type},
useExplanatory={age, medical history},
threshold=0.5,
enforcedBy={discrimination-free algorithm}.
```

Listing 2: Properties of *Anonymity* in P2

```
autoPropagated=True,
dataSubjectRole={Patient},
anonymizedWithin={All Patients},
level=fully anonymous,
insider=True,
enforcedBy={zero-knowledge proof}.
```

Listing 3: Properties of *Unobservability* in P2

```
autoPropagated=False,
dataSubjectRole={Patient},
anonymizedWithin={All Patients},
enforcedBy={Dummy traffic, DC-network}.
```

Considering pattern P2 in Fig. 6, 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 using 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 in Fig. 6 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. List. 2 and List. 3 show how the properties of the anonymity and the unobservability annotations are specified in pattern P2, respectively.

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 transmission of the messages. 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.

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 Fig. 6, together with the specifications of their properties, can be found in the SecBPMN2 model in Fig. 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 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 (Sec. 6) are easier to interpret by users. However, performing the alignment check can have broader positive implications for the overall development process. 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.

6 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 pro-

pose 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. Consequently, our tool shows conflicts as *errors* and potential conflicts as *warnings* to the user.

6.1 Automated conflict detection using anti-patterns

The reason for choosing SecBPMN2-Q for automating the conflict detection process is its expressiveness for encoding conflict anti-patterns. Specifically, SecBPMN2-Q has a relation called *walk* which, as mentioned earlier, 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 (47 anti-patterns).
- (B) potential conflicts between data-minimization and security requirements (93 anti-patterns).
- (C) conflicts between data-minimization and fairness requirements (2 *constrained* anti-patterns).
- (D) potential conflicts between data-minimization and fairness requirements (1 *constrained* anti-patterns).

If an anti-pattern has a match in the business process model, a conflict or potential conflict is reported. 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 the available data protection mechanisms is not a feasible because: (i) the number of the available data protection mechanisms grows continuously [75], and (ii) 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 [15]. 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

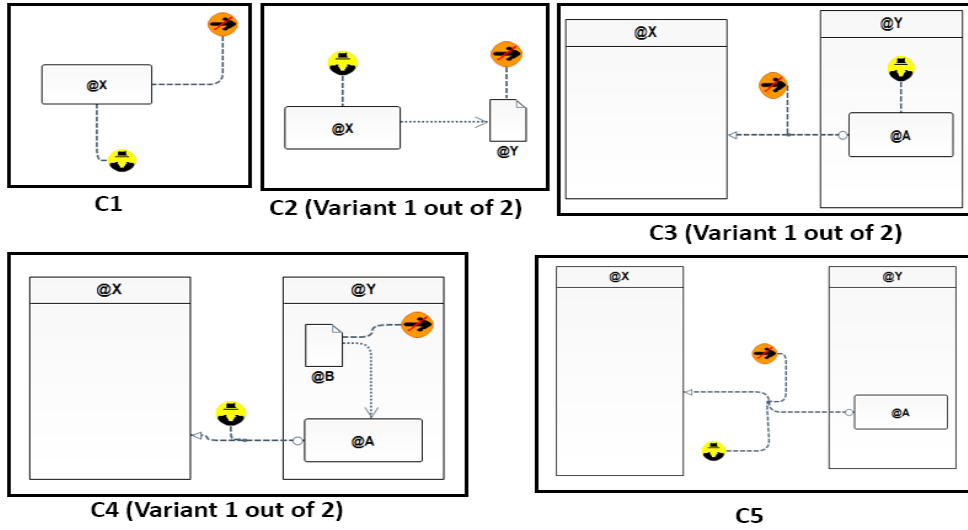


Fig. 7: Conflicts C1–C5 between *non-repudiation* and *anonymity* as anti-patterns.

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 Sec. 6.2, while a more detailed account is provided in Appendix (B).

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. For example, consider the *accountability* and *anonymity* annotations linked with the “Send data to portal” task in the “Telemedicine Device” pool at the left-hand side of Fig. 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.

Second, requirements related to different, dependent assets may be conflicting. For example, in Fig. 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, as shown in Fig. 5, imposes that an executor to the “Fill evaluation form” task should be fully anonymous against insider adversaries; the latter, as shown in Fig. 3, 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 Fig. 7 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*. All anonymity annotations in Fig. 7 are specified with the following attributes: $\{level=full\}$ *anonymity*, $\{insider=true\}$.

Consider, for example, conflicts C1 and C5 in Fig. 7. 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 Fig. 2: 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.

In contrast, C5 does not occur in the example model, since the model does not have an *anonymity*- and *non-repudiation*-annotated message flow. C2, C3, and C4 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. 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. 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, C4 does not occur in the example model, since the model does not have

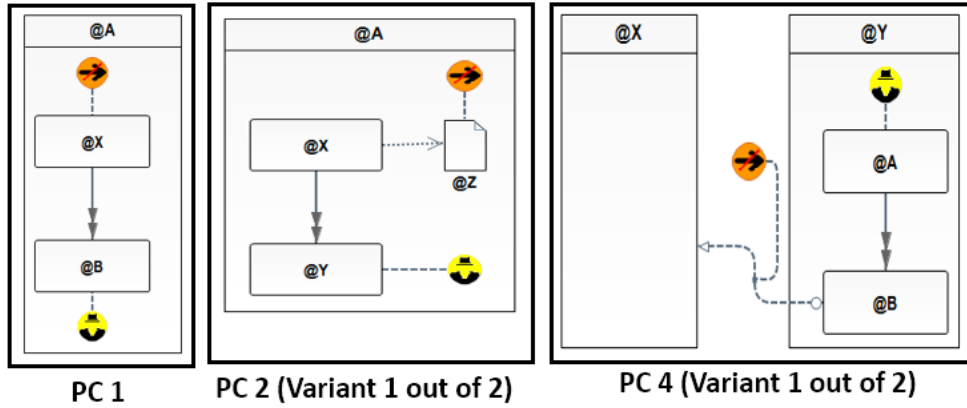


Fig. 8: Potential conflicts between *non-repudiation* and *anonymity* as anti-patterns.

a task that reads or writes data to/from a *non-repudiation*-annotated data object and at the same time sends messages over an *anonymity*-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, Fig. 2 includes a control dependency path between the *anonymity*-annotated “Fill evaluation form” task and the *non-repudiation*-annotated “Submit evaluation” task. Such situations not necessarily give rise to an actual conflict. For instance, imagine a flow 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. Such potential conflicts should be reported and discussed during the design of the business process models.

In Fig. 8 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, in Fig. 8 only 3 out of 8 overall potential conflicts for the considered pair of requirements are shown. Two additional cases called PC3 and PC5 are formed in analogy to C3 and C5 in Fig. 7; three additional variants arise from duality like in the discussion of the conflicts. Again, all anonymity annotations are specified with: $\{anonymity\ level=full\ anonymous, insider=true\}$.

The potential conflict PC1 in Fig. 8 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 Fig.8 with the example model in Fig. 2, four warnings will be reported, due to the following reasons: (i) there is a control dependency path between the *anonymity*-annotated “Fill evaluation form” task and the *non-repudiation*-annotated “Submit evaluation” task, thus violating PC1. (ii) PC2 is violated twice: first, there is a control path between the *anonymity*-annotated “Submit evaluation” task and the “Fill evaluation form” task, which writes data to the *non-repudiation*-annotated data object. Second, there is 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. (iii) there is a control 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 Fig. 9.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 fair-

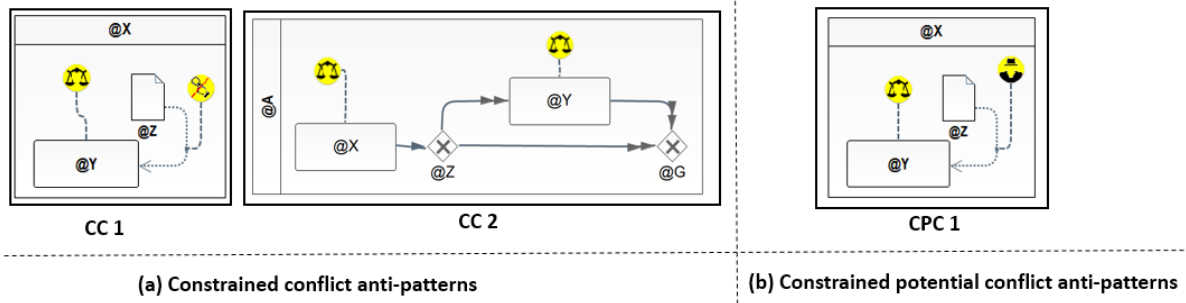


Fig. 9: Conflicts and potential conflicts between *fairness*, *undetectability*, and *anonymity* as constrained anti-patterns.

ness 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 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 Sec. 4, define the protected data 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 data 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 data [8]. This is important to avoid influencing the outputs of an *individual-fairness*-annotated task by protected data.

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, a 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 *undetect*, a conflict is reported if:

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

The *fair.getExplanatory()* and *fair.getProtected()* retrieve the set of data defined by *fair* as explanatory and protected data, respectively, while *und.getProtected()* retrieves the set of data specified as protected by *undetect*.

Matching the anti-pattern CC1 to the example model in Fig. 2 yields one conflict to be reported: There is a match for CC1, as the "System Portal" pool has an *undetectability*-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 Fig. 9 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 the first task intersects with the protected data 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 out-

put 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 data 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*, 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 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:

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

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

From matching the constrained conflict CC2 to the example model in Fig. 2, no conflicts are reported, since CC2 has no matches in the example model.

D. 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. Fig. 9 shows the constrained po-

tential conflict CPC1. The anonymity annotation in CPC1 is specified with the following attribute: $\{insider=true\}$. For the fairness annotation, no attribute values are specified.

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 this situation represents a conflict or not 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 [31]: (i) detect proxies for protected data in the case of individual-fairness is required or (ii) 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:

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

The $anon.getProtected()$ 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 data that are defined by the matched fairness annotation in m as explanatory and protected data, respectively.

From matching the constrained potential conflict CPC1 to the example model in Fig. 2, no conflicts are identified, as the anti-pattern has no matches in the example model.

6.2 Catalog of a domain-independent conflicts

As mentioned in Sec. 2, SecBPMN2 supports 10 different security requirements. Similar to our 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 happen. For each identified situation, we specified an anti-pattern using our extension of SecBPMN2-Q. To assure the correctness and completeness of our technique, each possible combination of requirements and their resulting conflicts were discussed by at least two authors. 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 Sec. 9).

Table 1: Overview of conflict + potential conflict anti-patterns per pair of requirements.

Requirement pair	<i>Accountability</i>	<i>Authenticity</i>	<i>Audibility</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>
Anonymity	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*
Unlinkability	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
Unobservability	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
Fairness	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

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. We now give an overview of the resulting catalog of anti-patterns. A more detailed account is provided in Appendix (B). Table 1 shows all pairs of requirements for which we identified a (potential) conflict. Each cell 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 Fig. 7 and Fig. 8. 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, similarly to *Non-repudiation*, 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: *Unlinkability* is linked to two pools and indicates that the two process executions should 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 ex-

ample, *zero-knowledge protocols* [49]. 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 on the abstraction level of process models. Therefore, as shown in Table 1, 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* [33]. 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. These kind of interactions can not be considered as conflicts or potential conflicts, and thus, they are omitted from Table 1.

Conversely, conflicts may not only occur between different categories of requirements (e.g., security requirements vs data-minimization requirements). Table 1 indicates that a particular *Anonymity* annotation might conflict with other *Anonymity* or *Unobservability* annotations. For example, requiring full anonymous access 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 this paper 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.

	Description
(Error) D_Anonym.NonRep(1)	An Anonymity (Anonymous,Insider)-annotated task sends messages over a Non-reputation-annotated message flow
(Error_A_Anonym.Account)	A task annotated with Anonymity (anonymous, insider) and Accountability
(Warning_E_Unobserv.NonRep)(1)	A path between a task that reads data from a Non-reputation-annotated data object and a task that sends messages
(Error_E_Unobserv.NonRep)(1)	A task reads data from a Non-reputation-annotated data object and sends messages over an Unobservability-annota

Fig. 10: *Analysis results* view from our tool.

In specific situations, a *Fairness* requirement can have conflicts with other fairness requirement or (potential) conflicts with data-minimization requirements. As shown in Table 1, *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 Fig. 9 and discussed in the previous section. The two intersections in Table 1 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 Fig. 9 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 Sec. 6.1.

7 Tool Support

We developed a prototypical implementation of our work on top of STS-tool [2], the supporting tool for SecBPMN2 [68]. STS-tool is a security requirement software tool that allows creating diagrams using SecBPMN2 language, it supports automated analyses and enforcement of security constraints¹. Our implementation supports the contributions of this work: first, modeling of security, data-minimization and fairness requirements in BPMN models, using a suitable model editor; second, modeling of security, data-minimization and fairness requirements as patterns and anti-patterns; third, automatic alignment verification between data protection requirements specified as patterns and their specifications in the annotated BPMN models; fourth, automatic conflict detection between data protection requirements in annotated BPMN models with data protection requirements, based on our catalog of domain-independent anti-patterns.

The examples shown in Figs. 2 and 6–9 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 output of the conflict detection is a set of textual messages that describe the detected conflicts. Fig. 10 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 Fig. 2 is the result of selecting the conflict message that describes the PC1 anti-pattern in Fig. 8. Our implementation is available online at <http://www.sts-tool.eu/downloads/secbpmn-dm>. Support for the constrained anti-patterns in Fig. 9 is not yet implemented in the current version of our tool and it is part of future work.

8 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, a pediatric Italian hospital. The case study was part of the *VisiOn* research project [3]. The main objective of VisiOn consisted of increasing the citizens' awareness on privacy. The final outcome of the project was a platform that can be used by public administrations and companies to design their systems, using privacy as a first-class requirement. The teleconsultations case study described a situation where a patient EHR can be transferred from the OPBG system to specialists in another hospital for a 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 paper, we extended this scenario to cover situations where data minimization and fairness play an important role in protecting the users' data. To this end, we modeled a process featuring an over distance healthcare service, an excerpt being shown in Fig. 2. Using our approach, as explained in Sect. 4, we were able to enrich the model with data minimization and fairness requirements that represent data-protection preferences for patients.

¹ <http://www.sts-tool.eu>

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 Sect. 6. Applied to the model, our conflict detection technique precisely 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 at the tool level. Our tool detects 21 potential conflicts as expected. The model with all security, data-minimization, and fairness requirements is publicly available at <https://github.com/QRamadan/SoSyM-conflictsDetection>.

9 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 our automated conflict detection?

We studied RQ1 and RQ2 in a user experiment with 30 new participants, a significantly larger sample than in our earlier preliminary study, which was based on 7 participants [61] and did not address RQ1, as it was based on a subject assessment only.

9.1 Set-up

Our overall sample consists of 30 participants, specifically, 18 master students, 1 bachelor student, 8 researchers, and 2 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 software development approach, and are therefore suitable as stand-ins for practitioners [67]. 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 authors (convenience sampling). We also asked all participants to self-assess their expertise in the relevant background fields BPMN,

Table 2: Experience levels of participants.

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

security, and privacy on a 5-point likert scale. For each background field, Table 2 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 an 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. The questionnaire consisted of three tasks in total. Tasks 1 and 2 were warm-up tasks for making the participants familiar with the data protection annotations from our approach. Task 3, elaborated below, focused on conflict detection. The tasks were based on the running example of this paper as introduced in Sec. 4. We showed excerpts from the contained BPMN model and security, data-minimization, and fairness requirements. Depending on the availability of a laptop, participants filled out the questionnaire on their computers or on a paper printout.

Task 3 and Task 4 represent the experimental tasks. We used a variant of the within-subject experimental design [17], 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. Afterward, 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. We provide all experimental materials and raw data online at <https://github.com/QRamadan/SoSyM-conflictsDetection>.

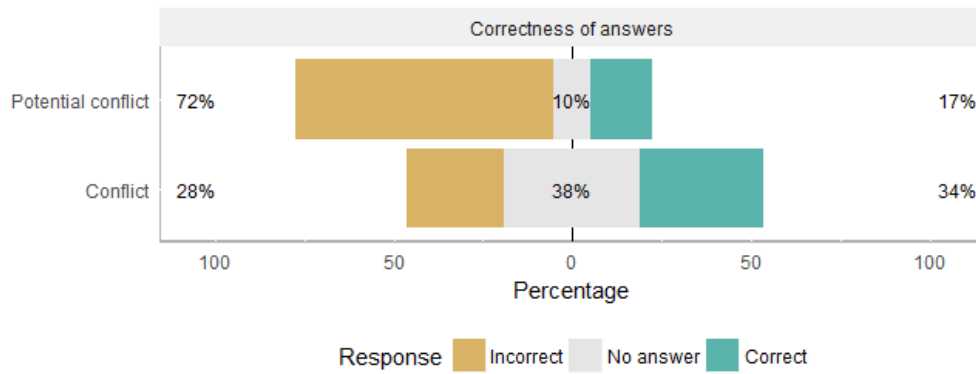


Fig. 11: Results for RQ1: Error-proneness of manual conflict detection

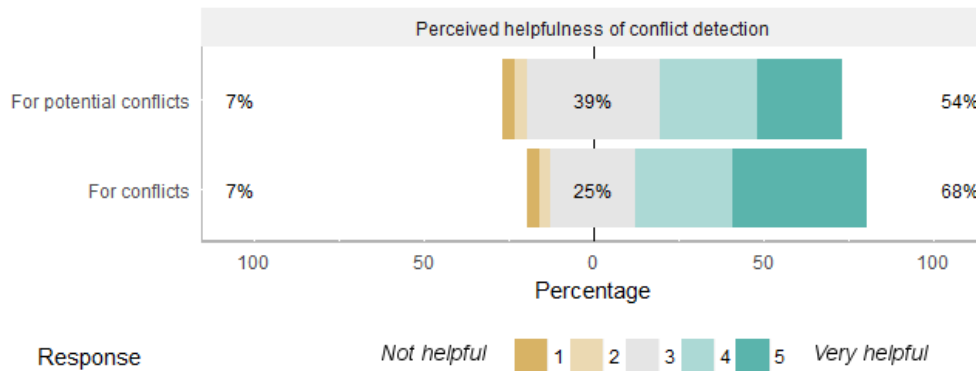


Fig. 12: Results for RQ2: Perceived helpfulness of automated conflict detection

9.2 Results

Regarding RQ1, Figure 11 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 12 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 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 annotations. Provided with the cheatsheet, a third of all participants were able to specify correct answers during manual conflict detection, indicating its general understandability. Still, we view this comment as 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). Our experimental design is subject to a number of threats to validity, as discussed below.

9.3 Threats to Validity.

The three main types of validity threats to our study are external, internal, and construct validity.

External validity. Our experimental design is subject to various threats to external validity: the obtained results may not generalize to other cases. 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. A more comprehensive study with a greater variety of case study is left to future work. Second, we relied to a large extent on students as participants in the study. We provided the rationale for this decision in Sec. 9.1.

Internal validity. A threat to internal validity is experimenter expectation. 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.

Construct validity. The validity of our experiments may be disputed due to constructing validity. When studying the practical applicability of our approach, we only considered the output of the conflict detection technique, rather the full process, including the assumption that the 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 our earlier work [71].

10 Related Work

Having given an overview of existing BPMN security-oriented extensions and their limitations in Sec. 2, we now provide a comprehensive discussion of other related work in the field of security engineering.

10.1 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. [33] define six privacy and security goals for supporting the privacy needs of users. The authors considered a subset of the data-minimization concepts in [56], namely *anonymity* and *unlinkability*, and discuss their relationships. However, conflicts are discussed on 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. [30] 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. [30] envision a realization based on the SecureTropos framework as future work.

10.2 Model-based conflict detection approaches

We are not the first to use models for detecting conflicts between data-protection properties. Paja et al. [54] 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 a 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 [68]. Second, a technique that supports the transformation from STS models to SecBPMN2 models is available [69].

Elahi et al. [26] 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 conflict analysis between security, data-minimization and fairness-related 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. [65] propose an approach that automatically, based on a 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 better overall satisfaction of non-functional requirements. For the same purpose, Pasquale et al. [55] 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 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, these works do not support conflict analysis between data protection 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 the relationships between them, without considering the target system's behavior.

10.3 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. [24] 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. [45] 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 [44]. 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. [57] propose a framework 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 [24, 44, 45, 57] do not support conflict detection between data protection requirements. Instead, they aim to detect conflicts between non-functional requirements such as security, usability, efficiency, and maintainability. Surprisingly, although the proposed matrix in [45] considers the privacy and 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.

10.4 Data-minimization-aware approaches

Various works in security requirements engineering aim to specify privacy requirements using the data minimization concepts proposed in [56]. In Deng et al.'s LINDDUN framework [22], 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. [37] 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. [52] 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. [13] 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. Ahmadian et al. [6] support a privacy analysis for system design models, based on the four privacy key elements of purpose, retention, visibility, and granularity.

Diamantopoulou et al. [23] 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 the 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 different 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 security, data-minimization, and fairness requirements and then to perform conflict detection.

10.5 Fairness-aware approaches

Fairness is currently dealt with as an algorithmic problem at the implementations 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 data or not. Approaches of this type can be further classified into while-box (e.g., [8]) and black-box (e.g., [20,31,74]) approaches. The trade-offs between fairness and privacy needs are recently considered as a challenge in the field of algorithmic fairness [20]. However, so far there is 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 is first has been highlighted by our work in [59,60].

11 Limitations

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 between the authors of this paper, and from the participants of an earlier experiment [62]. 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 the most significant direction for follow-up work.

This paper does not include a systematic performance evaluation 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 [70]. To still offer first insights into the scalability of our technique, we performed a preliminary assessment based on our running example², showing the results in Table 3. Models 1 and 2 are

Table 3: 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

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 our technique seems adequate for practical use on models of the considered size.

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.

12 Conclusions and Future Work

We proposed a 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 BPMN models, based on existing security annotations from the SecBPMN2 modeling language [68] 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 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.

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

² Available at <https://github.com/QRamadan/SoSyM-conflictsDetection>

study, respectively. 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 to rely on algebraic graph transformations in the following way: First, 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. Second, we aim to apply a formally based conflict detection technique [40,41], which can discover all possible conflicts arising for a set of rules.

Second, we aim to extend our approach to support the resolution of conflicts. Although a fully automated process would be appreciated, we believe that the resolution of actual conflicts (e.g., between two requirements related to different views of system users) is a sensitive issue that requires human intervention, a further challenging task that involves reasoning on the privacy impact of different solution strategies [7,46]. Once that all conflicts are resolved, the system design typically needs to be aligned with the specified data protection requirements, a challenge that can benefit from the use of model transformation technology [61].

Third, the maintenance of data protection patterns in our framework largely relies on manual tasks. This is due to a lack of 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 such as [69], and will allow avoiding badly modeled and duplicated patterns in the pattern repository.

Fourth, similar to the work in [58], our approach allows specifying enforcement technologies for data-protection annotations. In the future, to reduce the number of reported potential conflicts, we plan to refine our technique to consider the specified enforcement technologies during conflict detection. This requires as input a database with up-to-date information about enforcement technologies and their abilities to preserve data-protection requirements.

Acknowledgements We wish to thank Paolo Giorgini and the STS tool development team in the University of Trento for providing us with access to the source code of the STS tool. We also thank the participants of our experiment study. We wish to thank the anonymous referees, Julian Flake, and Ahd Al-Salman for their comments that helped us to improve the manuscript.

References

1. BPMN 2.0. <http://www.omg.org/spec/BPMN/2.0/>
2. STS. <http://www.sts-tool.eu/downloads/secbpmn-dm/>
3. VisiOn. <http://www.visioneuproject.eu/>
4. General Act on Equal Treatment (2009)
5. 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)
6. Ahmadian, A.S., Strüder, D., Riediger, V., Jürjens, J.: Model-based privacy analysis in industrial ecosystems. In: European Conference on Modelling Foundations and Applications, pp. 215–231. Springer (2017)
7. Ahmadian, A.S., Strüder, D., Riediger, V., Jürjens, J.: Supporting privacy impact assessment by model-based privacy analysis. In: ACM Symposium on Applied Computing. ACM (2018, to appear) (2018)
8. Albarghouthi, A., D’Antoni, L., Drews, S., Nori, A.: Fairness As a Program Property. arXiv preprint arXiv:1610.06067 (2016)
9. Aldekhail, M., Chikh, A., Ziani, D.: Software Requirements Conflict Identification: Review and Recommendations. International Journal of Advanced Computer Science and Applications 7(10), 326–335 (2016)
10. Alkubaisy, D.: A framework managing conflicts between security and privacy requirements. In: International Conference on Research Challenges in Information Science, pp. 427–432. IEEE (2017)
11. Arsac, W., Compagna, L., Pellegrino, G., Ponta, S.E.: Security Validation of Business Processes via Model-Checking. ESSoS 6542, 29–42 (2011)
12. Barocas, S., Selbst, A.D.: Big data’s disparate impact. Cal. L. Rev. 104, 671 (2016)
13. Beckers, K., Faßbender, S., Heisel, M., Meis, R.: A Problem-based Approach For Computer-Aided Privacy Threat Identification. In: Annual Privacy Forum, pp. 1–16. Springer (2012)
14. Brucker, A.D., Hang, I., Lückemeyer, G., Ruparel, R.: SecureBPMN: Modeling and enforcing access control requirements in business processes. In: ACM Symposium on Access Control Models and Technologies, pp. 123–126. ACM (2012)
15. Bürger, J., Strüder, D., Gärtner, S., Ruhroth, T., Jürjens, J., Schneider, K.: A framework for semi-automated co-evolution of security knowledge and system models. Journal of Systems and Software 139, 142–160 (2018)
16. Calders, T., Verwer, S.: Three naive Bayes Approaches For Discrimination-Free Classification. Data Mining and Knowledge Discovery 21(2), 277–292 (2010)
17. Charness, G., Gneezy, U., Kuhn, M.A.: Experimental methods: Between-subject and within-subject design. Journal of Economic Behavior & Organization 81(1), 1–8 (2012)
18. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM 28(10), 1030–1044 (1985)
19. Datta, A.: Fairness and Privacy Violations in Black-Box Personalization Systems: Detection and Defenses. Ph.D. thesis, Carnegie Mellon University (2018)
20. Datta, A., Fredrikson, M., Ko, G., Mardziel, P., Sen, S.: Proxy Non-Discrimination In Data-Driven Systems. arXiv preprint arXiv:1707.08120 (2017)
21. Datta, A., Fredrikson, M., Ko, G., Mardziel, P., Sen, S.: 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, pp. 1193–1210. ACM (2017)
22. Deng, M., Wuyts, K., Scandariato, R., Preneel, B., Joosen, W.: A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. Requirements Engineering 16(1), 3–32 (2011)
23. Diamantopoulou, V., Argyropoulos, N., Kalloniatis, C., Gritzalis, S.: Supporting The Design Of Privacy-Aware Business Processes via Privacy Process Patterns. In: International Conference on Research Challenges in Information Science, pp. 187–198. IEEE (2017)

24. Egyed, A., Grunbacher, P.: Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability can Help. *IEEE software* **21**(6), 50–58 (2004)
25. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Planning under incomplete knowledge. In: *Proc. of Computational Logic*, vol. 1861, pp. 807–821 (2000)
26. Elahi, G., Yu, E.: A Goal Oriented Approach for Modeling and Analyzing Security Trade-offs. In: *International Conference on Conceptual Modeling*, pp. 375–390. Springer (2007)
27. Fantinato, M., Toledo, M.B.F.d., Thom, L.H., Gimenes, I.M.d.S., Rocha, R.d.S., Garcia, D.Z.G.: A survey On Reuse In The Business Process Management Domain. *International Journal of Business Process Integration and Management* **6**(1), 52–76 (2012)
28. Feldman, M., Friedler, S.A., Moeller, J., Scheidegger, C., Venkatasubramanian, S.: Certifying and Removing Disparate Impact. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 259–268. ACM (2015)
29. Ferraiolo, D., Cugini, J., Kuhn, D.R.: Role-based Access Control (RBAC): Features and Motivations. In: *Proceedings of 11th annual computer security application conference*, pp. 241–48 (1995)
30. Ganji, D., Mouratidis, H., Gheytaei, S.M., Petridis, M.: Conflicts Between Security and Privacy Measures in Software Requirements Engineering. In: *International Conference on Global Security, Safety, and Sustainability*, pp. 323–334. Springer (2015)
31. Gupta, M., Cotter, A., Fard, M.M., Wang, S.: Proxy Fairness. *arXiv preprint arXiv:1806.11212* (2018)
32. Gürses, S., Troncoso, C., Diaz, C.: Engineering privacy by design. *Computers, Privacy & Data Protection* **14**(3) (2011)
33. Hansen, M., Jensen, M., Rost, M.: Protection goals for privacy engineering. In: *Security and Privacy Workshops (SPW)*, 2015 IEEE, pp. 159–166. IEEE (2015)
34. Houmb, S.H., Islam, S., Knauss, E., Jürjens, J., Schneider, K.: Eliciting Security Requirements and Tracing them to Design: an integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering* **15**(1), 63–93 (2010)
35. Ingold, D., Soper, S.: Amazon Doesn't Consider the Race of Its Customers. Should It? <https://www.bloomberg.com/graphics/2016-amazon-same-day/> (2016)
36. ISO, IEC: Common Criteria For Information Technology Security Evaluation - Part 2 Security Functional Components. In: *ISO/IEC 15408, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC)* (2012)
37. Kalloniatis, C., Kavakli, E., Gritzalis, S.: Addressing privacy requirements in system design: the PriS method. *Requirements Engineering* **13**(3), 241–255 (2008)
38. Kim, M., Park, S., Sugumaran, V., Yang, H.: Managing Requirements Conflicts In Software Product Lines: A Goal and Scenario Based Approach. *Data & Knowledge Engineering* **61**(3), 417–432 (2007)
39. Labda, W., Mehandjiev, N., Sampaio, P.: Modeling of privacy-aware business processes in BPMN to protect personal data. In: *ACM Symposium on Applied Computing*, pp. 1399–1405. ACM (2014)
40. Lambers, L., Born, K., Kosiol, J., Strüder, D., Taentzer, G.: Granularity of conflicts and dependencies in graph transformation systems: A two-dimensional approach. *J. Log. Algebr. Meth. Program.* **103**, 105–129 (2019)
41. Lambers, L., Strüder, D., Taentzer, G., Born, K., Huebert, J.: Multi-Granular Conflict and Dependency Analysis in Software Engineering based on Graph Transformation. In: *International Conference on Software Engineering*, pp. 716–727. IEEE/ACM (2018)
42. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic* **7**(3), 499–562 (2006)
43. Maines, C.L., Llewellyn-Jones, D., Tang, S., Zhou, B.: 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*, pp. 1756–1763. IEEE (2015)
44. Mairiza, D., Zowghi, D.: An Ontological Framework to Manage the Relative Conflicts between Security and Usability Requirements. In: *Managing Requirements Knowledge (MARK)*, 2010 Third International Workshop on, pp. 1–6. IEEE (2010)
45. Mairiza, D., Zowghi, D., Nurmiliani, N.: Towards a catalogue of conflicts among non-functional requirements. In: *International Conference on Evaluation of Novel Approaches to Software Engineering*. SciTePress (2010)
46. Meis, R., Heisel, M.: Systematic Identification Of Information Flows From Requirements To Support Privacy Impact Assessments. In: *International Joint Conference on Software Technologies*, vol. 2, pp. 1–10. IEEE (2015)
47. Mendes, R., Vilela, J.P.: Privacy-Preserving Data Mining: Methods, Metrics, and Applications. *IEEE Access* **5**, 10,562–10,582 (2017)
48. Menzel, M., Thomas, I., Meinel, C.: Security requirements specification in service-oriented business process management. In: *International Conference on Availability, Reliability and Security*, pp. 41–48. IEEE (2009)
49. Mohr, A.: A survey of zero-knowledge proofs with applications to cryptography. *Southern Illinois University, Carbondale* pp. 1–12 (2007)
50. Moody, D.: 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)
51. Morton, A., Sasse, M.A.: Privacy is a process, not a PET: A theory for effective privacy practice. In: *Proceedings of the 2012 workshop on New security paradigms*, pp. 87–104. ACM (2012)
52. Mouratidis, H., Kalloniatis, C., Islam, S., Huget, M.P., Gritzalis, S.: Aligning Security and Privacy to Support the Development of Secure Information Systems. *J. UCS* **18**(12), 1608–1627 (2012)
53. Mülle, J., von Stackelberg, S., Böhm, K.: A security language for BPMN process models. KIT, Fakultät für Informatik (2011)
54. Paja, E., Dalpiaz, F., Giorgini, P.: Managing Security Requirements Conflicts in Socio-Technical Systems. In: *International Conference on Conceptual Modeling*, pp. 270–283. Springer (2013)
55. Pasquale, L., Spoletini, P., Salehie, M., Cavallaro, L., Nuseibeh, B.: Automating trade-off analysis of security requirements. *Requirements Engineering* **21**(4), 481–504 (2016)
56. Pfizmann, A., Hansen, M.: A Terminology For Talking About Privacy by Data Minimization: Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity Management. TU Dresden and ULD Kiel, Tech. Rep (2011)
57. Poort, E.R., de With, P.: Resolving Requirement Conflicts through Non-Functional Decomposition. In: *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pp. 145–154. IEEE (2004)
58. Pullonen, P., Matulevičius, R., Bogdanov, D.: PE-BPMN: privacy-enhanced business process model and notation. In: *International Conference on Business Process Management*, pp. 40–56. Springer (2017)
59. Ramadan, Q., Ahmadian, A.S., Jürjens, J., Staab, S., Strüder, D.: Explaining Algorithmic Decisions with respect to Fairness. In: *SE/SWM 2019: Multikonferenz Software Engineering und Management, Special Track on Explainable Software*, Stuttgart, Germany (2019). To appear
60. Ramadan, Q., Ahmadian, A.S., Strüder, D., Jürjens, J., Staab, S.: Model-based discrimination analysis: a position paper. In: *Proceedings of the International Workshop FairWare@ICSE 2018, Gothenburg, Sweden* (2018)

61. Ramadan, Q., Salnitri, M., Strüder, D., Jürjens, J., Giorgini, P.: From secure business process modeling to design-level security verification. In: International Conference on Model Driven Engineering Languages and Systems, pp. 123–133. IEEE (2017)
62. Ramadan, Q., Strüder, D., Salnitri, M., Riediger, V., Jürjens, J.: Detecting Conflicts Between Data-Minimization and Security Requirements in Business Process Models. In: European Conference on Modelling Foundations and Applications, pp. 179–198. Springer (2018)
63. Raymond, J.F.: Traffic analysis: Protocols, attacks, design issues, and open problems. In: Designing Privacy Enhancing Technologies, pp. 10–29. Springer (2001)
64. Rodríguez, A., Fernández-Medina, E., Trujillo, J., Piattini, M.: Secure business process model specification through a UML 2.0 activity diagram profile. *Decision Support Systems* **51**(3), 446–465 (2011)
65. Saadatmand, M., Tahvili, S.: A Fuzzy Decision Support Approach for Model-Based Tradeoff Analysis of Non-Functional Requirements. In: Information Technology-New Generations (ITNG), 2015 12th International Conference on, pp. 112–121. IEEE (2015)
66. Saleem, M., Jaafar, J., Hassan, M.: A domain-specific language for modelling security objectives in a business process models of soa applications. *AISS* **4**(1), 353–362 (2012)
67. Salman, I., Misirli, A.T., Juristo, N.: Are students representatives of professionals in software engineering experiments? In: Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, vol. 1, pp. 666–676. IEEE (2015)
68. Salnitri, M., Dalpiaz, F., Giorgini, P.: Modeling and verifying security policies in business processes. In: Enterprise, Business-Process and Information Systems Modeling, pp. 200–214. Springer (2014)
69. Salnitri, M., Giorgini, P.: Transforming Socio-Technical Security Requirements in SecBPMN Security Policies. In: iStar (2014)
70. Salnitri, M., Paja, E., Giorgini, P.: Maintaining secure business processes in light of socio-technical systems' evolution. In: 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), pp. 155–164. IEEE (2016)
71. Salnitri, M., Paja, E., Giorgini, P., et al.: From Socio-Technical Requirements to Technical Security Design: an STS-based Framework. DISI-University of Trento (2015)
72. Spiekermann, S., Cranor, L.F.: Engineering privacy. *IEEE Transactions on software engineering* **35**(1), 67–82 (2009)
73. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10**(05), 571–588 (2002)
74. Tramèr, F., Atlidakis, V., Geambasu, R., Hsu, D.J., Hubaux, J.P., Humbert, M., Juels, A., Lin, H.: Discovering Unwarranted Associations in Data-Driven Applications with the Fairtest Testing Toolkit. CoRR, abs/1510.02377 (2015)
75. Van Blarckom, G., Borking, J., Olk, J.: Handbook Of Privacy and Privacy-Enhancing Technologies. Privacy Incorporated Software Agent (PISA) Consortium, The Hague (2003)
76. Vivas, J.L., Montenegro, J.A., López, J.: Towards a business process-driven framework for security engineering with the UML. In: International Conference on Information Security, pp. 381–395. Springer (2003)
77. Wolter, C., Schaad, A.: Modeling of task-based authorization constraints in BPMN. *Business process management* pp. 64–79 (2007)
78. Zliobaite, I.: A survey On Measuring Indirect Discrimination In Machine Learning. CoRR abs/1511.00148 (2015)

13 Appendix

(A) The linkage constraints of our proposed annotations

In our work, an annotation can be linked with a BPMN element if that element may act as a subject to the corresponding privacy threat of that annotation. The underlying background for our linkage constraints benefits from the research work of Deng et al.[22], 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 [22] mapped every privacy threat to all elements of the data flow diagram, while in our work we argue that some privacy threats are the consequence of other threats in the system. Thus, we mapped the privacy threats to specific elements in a restricted list of BPMN model elements to avoid the overlapping meaning of different data-minimization annotations. Generally, Table 4 shows that the BPMN *activities*, *data associations*, *data objects*, and *message flows* can be subject to one or more privacy threats.

In our work, we consider the following threats: (i) identifying data subjects who their real identity should be uncovered for privacy reasons (*Identifiability*), (ii) linking different activities or data from different sources as related (*Linkability*), (iii) leaking sensitive information about private communications or stored data even if they are encrypted (*Detectability*), (iv) 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, and (V) discriminating data subjects on the ground of *protected data* (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 4 displays the mapping between the basic elements of BPMN and the set of privacy threats *identifiability*, *linkability*, *detectability*, *observability*, *discrimination*.

Each intersection in Table 4 marked with the symbol (●) indicates a potential privacy 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 messages being transmitted over a message flow. These messages can be subject to a number of privacy 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. These threats can be mitigated by achieving the *anonymity*, *undetectability* and *unobservability* data-minimization needs, respectively.

Entries with the symbol (—) in Table 4, indicate that the BPMN element cannot be subject to the corresponding privacy threats. Table 4 shows that none of the privacy threats can be mapped to *events*, *gateways*, and *sequence flows*. The

Table 4: Mapping privacy threats to BPMN elements

Category	Element	Privacy 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	○	●	○	○	—

rationale of this is that these BPMN elements do not store, process or transmit personal information. More precisely, 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 environment of the system such as *events*. Therefore, we did not consider them as subject to privacy threats.

The symbol (○) in Table 4 indicates that the BPMN element can be subject to the corresponding privacy 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. For example, a *data object* that stores personal identifiable information can be a 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.

(B) A full textual description of our catalog of conflicts

This section provides a full textual description of our catalog of conflicts (C) and potential conflicts (PC) between security, fairness, and data minimization requirements.

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 imposing 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 data 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) an *Non-repudiation*-annotated task sends messages over an *Unobservability*-annotated message flow, or (3) a task that reads data from an *Non-repudiation*-annotated data object and sends messages over a *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 de-

fined), 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 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).

Accountability. *Accountability* 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 occurs if the business process model has: (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.
- **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).
- **C3.2 Authenticity vs Unobservability.** A conflict between an unobservability and an authenticity requirement occurs if the business process model 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).

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 imposing that it should be possible to keep track of all the actions (e.g., write, read, store) concerning the data object. Third, a message flow to specify that it should be possible to keep track of all the ac-

tions 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).
- **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. Accountability may conflict with anonymity and unobservability.

- **C5.1 Non-delegation vs Anonymity.** A conflict between these two requirements occurs if the BPMN 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-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 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.
- **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.

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 a set of related activities. Binding of duties may conflict with anonymity and unlinkability.

- **P6.1 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}.

Anonymity. *Anonymity* as specified in Sec. 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.

- **C7.1 Anonymity vs Anonymity.** A conflict between two anonymity requirements occurs if the BPMN 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 $\{level= pseudonymity\}$, 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 $\{level= pseudonymity\}$.
- **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 $\{level=pseudonymity\}$, (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 mes-

sages 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 a set 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 BPMN 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 imposing 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 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).
- **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* can be linked with: first, a data object to imposing 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 imposing 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 a *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 imposing 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 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.

- **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 Sec. 4 can be linked with a task BPMN element. This annotation can be specified differently based on: first, the data that should not be subject to discrimination. Second, the data that their effect 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 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 conflict situations as constrained conflict 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.

- **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 Fig.9.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 $undetect.getProtected()$ retrieves the set of data specified as protected by $undetect$.

- **C12.2 Fairness vs Fairness.** In our work, a conflict between two fairness requirements can be reported:
 if (the CC2 of Fig.9.a is matched in m) and

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

Here, m is a SecBPMN2 model, $fair_1.getExplanatory()$ and $fair_2.getProtected()$ retrieve the set of data that are defined as explanatory and protected data, 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 Fig.9.b is matched in m)
 and

$$(fair.getProtected() \cup fair.getExplanatory())$$

$$\cap anon.getProtected() \neq \emptyset$$

Here, m is a SecBPMN2 model, $anon.getProtected()$ retrieves the set of data that are specified as protected by the matched anonymity annotation in m . Both the $fair.getExplanatory()$ and $fair.getProtected()$ retrieve the set of data that are defined by the matched fairness annotation in m as explanatory and protected data, respectively.