On Handling Data Minimization in Workflows

Saliha Irem Besik[1] Johann-Christoph Freytag[2]

Abstract: With the arrival of the European Union General Data Protection Regulation (GDPR), organizations dealing with personal data put in a great deal of effort into achieving compliance. GDPR, by promoting a Privacy by Design (PbD) approach, forces organizations to embed privacy into their technology design proactively. We argue that workflows can help to ensure PbD through design-time compliance checking. In our work, we focus on one of the significant data protection principles under GDPR which is data minimization, i.e. the personal data shall be adequate, relevant, and limited to what is necessary in relation to the purposes for which they are processed. This paper introduces our formal approach for detecting and repairing data minimization violations in workflows.

Keywords: Data Privacy; General Data Protection Regulation (GDPR); Business Reengineering

1 Introduction

The Big Data revolution coins the motto “data is the new oil” which has triggered several companies to store as much data as possible, including personal data. With the emergence of the European Union General Data Protection Regulation (GDPR), suddenly those massive data collections became serious risks because companies must take the data minimization principle, among others, into consideration when storing and processing personal data. The data minimization principle under GDPR disallows organizations to process more data than necessary.

According to the principle of Privacy by Design (PbD), companies dealing with the personal data of European citizens have to be proactive in addressing privacy. Therefore, they must ensure that their data operations are compliant with GDPR throughout the data lifecycle including the design phase. In order to support PbD, we believe that companies can strongly benefit from workflow technology by checking compliance of their workflow models with GDPR during design time.

While many researchers have been primarily focused on the control flow aspect of the workflows which is about the activities and their ordering, there are only a few approaches on the data flow perspective of the workflows. In our work, we investigate existing workflows as a means to capture how data is processed for what purpose at the design level. We use

[1] Humboldt-Universität zu Berlin, Department of Computer Science, Unter den Linden 6, 10099 Berlin, Germany besiksal@informatik.hu-berlin.de

[2] Humboldt-Universität zu Berlin, Department of Computer Science, Unter den Linden 6, 10099 Berlin, Germany freytag@informatik.hu-berlin.de

Business Process Model and Notation (BPMN) as it is a defacto standard for business process modeling to represent workflows; however, our proposed approach can be applied to any workflow model in any other process modeling language as well.

In this article, we present our formal approach to detect and repair privacy violations regarding the data minimization principle in BPMN-based workflows.

The rest of the article is structured as follows: The next section briefly introduces the data minimization principle under GDPR. Section 3 provides our formal foundations regarding BPMN-based workflows. In Section 4, we present our approach for detecting data minimization anomalies in workflows. We explain our repair strategies regarding these anomalies in Section 5, and analyze our methodology in Section 6. Section 7 discusses related work, before Section 8 concludes this paper and points to future direction.

2 Overview of Data Minimization Principle under GDPR

In the following, we briefly introduce fundamental concepts under GDPR with respect to the data minimization principle to provide a clear understanding of our approach.

The General Data Protection Regulation (GDPR) became an enforceable law starting May 25, 2018, affecting all organizations processing personal data of EU citizens. In a nutshell, GDPR establishes principles on how to process personal data (Article 5). We do not describe all of those principles in this paper, but rather concentrate on one of the fundamental principles, namely the data minimization principle.

Data Minimization Principle: “Personal data shall be adequate, relevant, and limited to what is necessary in relation to the purposes for which they are processed.” [GDPR, Article 5 §1(c)]

In this statement, we encounter two vital concepts which are *personal data* and *purpose*. GDPR has a broad interpretation of *personal data* as it includes any information that relates to a natural person, even if it does not uniquely identify that person (e.g. search terms submitted to a search engine). Since data privacy is always the subject of personal data, throughout the paper when we use the term "data", we refer to personal data.

Purpose specifies the reason for which personal data is collected, used, or disclosed. There might be different purposes both privacy-related ones defined in the GDPR (public interest, public health, etc.) and domain-specific ones (authentication, screening, treatment, etc.). GDPR mandates that personal data shall be collected for specified, explicit, and legitimate purposes, and not further processed in a manner that is incompatible with those purposes (GDPR, Article 5 §1(b)).

3 Foundation

In this section, we introduce some key concepts and formal definitions regarding workflows which we shall use in the remainder of this paper.

In our research, we adopt BPMN 2.0 as a notation for modeling workflows as it is a well-established standard which is developed and maintained by the Object Management Group (OMG). We use BPMN-based workflows as a means to ensure data privacy during design time. Therefore, we focus on the data flow perspective of workflows. Based on the definitions given by [DDO07] and [TVdAS09], we provide a formal definition for *data-aware workflow*. A data-aware workflow is defined as a subset of BPMN elements, as defined in Definition 1.

Definition 1 (Data-Aware Workflow) A data-aware workflow model is represented as 6-tuples $\mathcal{W} = (C, \mathcal{D}, \mathcal{F}, Read, Write, Delete)$ where:

- C is a set of components which can be partitioned into disjoint sets of tasks $\mathcal{T}$, events $\mathcal{E}$, and gateways $\mathcal{G}$,
- $\mathcal{D}$ is a set of data components which consists of set of data objects (DO) and set of data stores (DS); that is D = DO $\cup$ DS,
- $\mathcal{T}^{\mathcal{D}} \subseteq \mathcal{T}$ is a set of data tasks which read, write, or delete data items from data components $\mathcal{D}$,
- $e^{\mathcal{S}} \in \mathcal{E}$ is a start event, $e^{\mathcal{E}} \in \mathcal{E}$ is an end event,
- $\mathcal{G}$ can be partitioned into disjoint sets of exclusive (XOR) gateways $\mathcal{G}^{\mathcal{E}}$ and parallel (AND) gateways $\mathcal{G}^{\mathcal{P}}$,
- $\mathcal{F} \subseteq C \times C$ is the control flow relation, i.e. a set of sequence flows connecting components, where $\mathcal{F}_1$ is the source component and $\mathcal{F}_2$ is the target component of the sequence flow $\mathcal{F}$,
- $Read : \mathcal{T}^{\mathcal{D}} \rightarrow \{(atrn, pr)\}$ is the reading data annotation function which maps data tasks to the set of pairs $(atrn, pr)$, where $atrn$ is an attribute name of personal data which are read by the corresponding data task and pr stands for the purpose of reading,
- $Write : \mathcal{T}^{\mathcal{D}} \rightarrow \{(atrn)\}$ is the writing data annotation function which maps data tasks to the set $(atrn)$, where $atrn$ is an attribute name of personal data which are written by the corresponding data task,
- $Delete : \mathcal{T}^{\mathcal{D}} \rightarrow \{(atrn)\}$ is the deleting data annotation function which maps data tasks to the set $(atrn)$, where $atrn$ is an attribute name of personal data which are deleted by the corresponding data task. $\square$

A data-aware workflow $\mathcal{W}$ is a directed graph with components C as nodes and sequence flows $\mathcal{F}$ as vertices. Components C consist of the pairwise disjoint sets of tasks $\mathcal{T}$, events $\mathcal{E}$, and gateways $\mathcal{G}$. We define a set of data task $\mathcal{T}^{\mathcal{D}}$ as a subset of tasks $\mathcal{T}$ which may read, write, or delete a particular data element from data components $\mathcal{D}$. Data is handled in data-aware workflows via data store and data object elements. In some BPMN-based workflows, data objects are used as both physical (e.g. a paper invoice) and digital data storage units, and they store data only during the execution of workflows. Data store, on the other hand, contains data that persist beyond the duration of a workflow instance. However, in our study, we assume that there are no physical objects carrying personal data and all data components provide persistent storage. Thus, we do not consider data objects and data stores differently.

For each data task $t \in \mathcal{T}^{\mathcal{D}}$, we provide functions $Read$, $Write$, and $Delete$. $Read(t)$ function returns set of tuples as $(atrn, pr)$ which means data task t reads data element with the name $atrn$, for the purpose of pr. For instance, $Read(t_1) = (name, marketing)$ means that data task t_1 reads $name$ for the purpose of $marketing$. $Read(t)$ function might return an empty set in case data task t does not read any data element. Whenever a data task reads a data item, the purpose for this access must be stated (GDPR, Article 5 §1(b)).

$Write$ function returns a set of attribute names which are written by the corresponding data task and $Delete$ function returns a set of attribute names which are deleted by the corresponding data task. Similar to the $Read$ function, $Write$ and $Delete$ functions might return empty sets. For write and delete operations, we do not include purpose information. Therefore, only $Read$ functions include purpose information in addition to the attribute name.

It is important to highlight that $atrn$ in the annotation functions is considered as an attribute name from a given database schema. During design time, we know which attributes are used by which data task but we cannot determine actual *attribute values*.

Figure 1 shows the core BPMN elements which we use in our data-aware workflow definition.

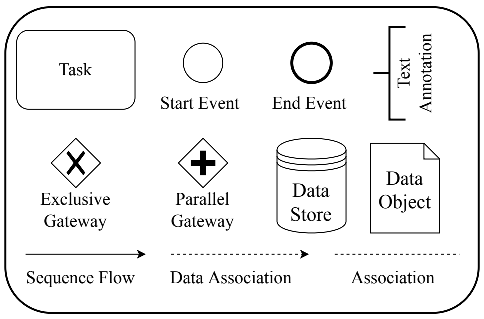


Fig. 1: Core BPMN elements.

We assume that a data-aware workflow $\mathcal{W}$ is assumed to be structurally sound [We12], which means that it meets the following criteria:

- It begins with exactly one start event $e^{\mathcal{S}} \in \mathcal{E}$, which is the only component without any incoming edges.
- It terminates with exactly one end event $e^{\mathcal{E}} \in \mathcal{E}$, which is the only component without any outgoing edges.
- Each of its components $c \in C$ lies on a path from $e^{\mathcal{S}}$ initial node to the $e^{\mathcal{E}}$ final node.

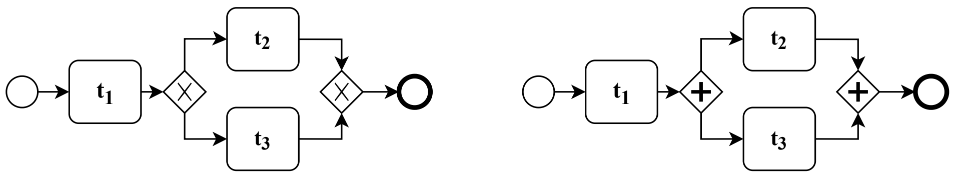
Soundness is the most widely used correctness notion for workflows. It is similar to the normalization concept in database theory. Further information regarding the soundness property and why it is beneficial can be found in [Du13].

In order to assess the compliance of a data-aware workflow, we need to understand its execution semantics. A data-aware workflow might have different executions because of gateway components. Exclusive gateway, for instance, allows choosing one out of several branches according to the transition condition linked to the gateway. Hence, each branch might result in a different execution. We define a data flow trace that represents all possible executions of data tasks of a given a data-aware workflow, in Definition 2. In our definition, we have abstracted the definition of execution trace given by [RW12], by disregarding all the components except data tasks.

Definition 2 (Data Flow Trace) Let $\mathcal{W}$ be a data-aware workflow model, $\mathcal{T}^{\mathcal{D}}$ be the set of data tasks of $\mathcal{W}$, and $\Phi_{\mathcal{W}}$ be the set of all possible completed data flow executions on $\mathcal{W}$. Then, a data flow trace $\phi \in \Phi_{\mathcal{W}}$ is given by $\phi = <t_1, \dots, t_k>$, all $t_i \in \mathcal{T}^{\mathcal{D}}$ where the order of $t_i \in \phi$ reflects the temporal order in which data tasks t_i related to the execution of activities or the evaluation of transition conditions occurred on $\mathcal{W}$. □

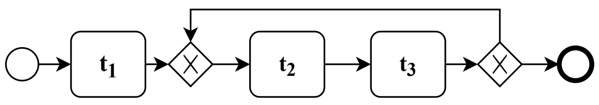
A data flow trace represents a completed execution. That is, it contains a list of data tasks to be executed from the start event (e^S) to the end event (e^E). We can determine the possible data flow traces during design time.

Figure 2 illustrates different branching scenarios and resulting traces. In this figure, we omit the data components for simplification purpose; however, we assume that all tasks are data tasks. Figure 2(a) contains alternative data tasks (either t_2 or t_3) to be executed after an exclusive gateway. Second exclusive gateway is a merge gateway which waits for one of the branches to be completed before triggering the end event. Figure 2(b) contains a parallel gateway which is used to show concurrent execution of the data tasks. Both t_2 and t_3 are executed but the order of the execution cannot be determined during design time. Figure 2(c) illustrates a loop scenario which means the tasks within the loop (t_2 and t_3) can be executed several times until the transition condition attached to the exclusive gateway met. A data-aware workflow is assumed to be structurally sound; therefore, there must not be any



(a) ϕ_1 = <t_1, t_2>, ϕ_2 = <t_1, t_3>

(b) ϕ_1 = <t_1, t_2, t_3>, ϕ_2 = <t_1, t_3, t_2>



(c) ϕ_1 = <t_1, t_2, t_3>, ϕ_2 = <t_1, t_2, t_3, t_2, t_3>, ...

Fig. 2: Data flow traces for different scenarios: (a) alternative branching with XOR gateways, (b) concurrent execution with AND gateways, (c) loops.

deadlock and the loop must be ended after a bounded number of iterations. In Figure 2(c), ϕ_1 contains one iteration and ϕ_2 have two iterations.

Figure 3 illustrates a sample BPMN-based workflow for blood screening. In this data-aware workflow, all tasks are data tasks as they access the central data store HospitalDB. The workflow starts with a *start event* patient arrives and continues with the data tasks. We attach data annotations to each data task. For instance, after patient arrives, the first task is *perform physical examination*. It writes *anamnesis* to the data store, which is shown as $W\{anamnesis\}$. The data task *conduct research* reads $blood-type$ for the purpose of $statistics$, reads $blood-sugar$ for the purpose of $screening$, and writes $result$. Its data annotation is shown as $R\{(blood-type, statistics), (blood-sugar, screening)\}, W\{result\}$.

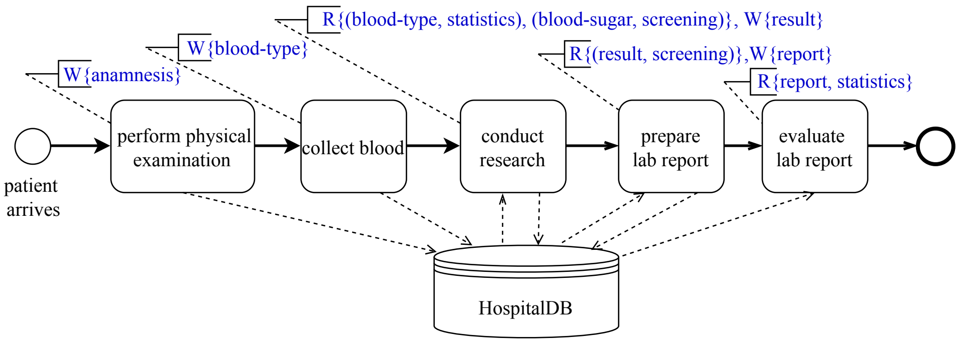


Fig. 3: Sample Data-aware Workflow.

There is only one data flow trace for this data-aware workflow since there are no gateways which create alternative paths. Let t_1 as being the first data task (perform physical examination), t_2 as the second data task (collect blood), and so on. Then, the data flow trace becomes ϕ = <t_1, t_2, t_3, t_4, t_5>.

4 Detecting Data Minimization Anomalies

In this section, we present how to determine whether a given data-aware workflow is free from data minimization problems.

According to the data minimization principle under GDPR, personal data must comply with three conditions: being adequate, relevant, and limited to what is necessary. We define the data minimization anomalies as *missing data*, *unrelated data*, and *redundant data* by deciding the opposite of each condition listed above. Thus, we express the data minimization principle in other words, as follows: Data processing should be free from missing data, unrelated data, and redundant data in relation to the purposes for which they are processed.

The anomalies are categorized into three types, as the following:

Missing Data Issue might occur when some data element needs to be accessed, i.e. read or deleted, but either it has not been written before or it has been deleted without having been written again. We classify it as a potential issue because we do not know whether the data item has already been written by another workflow beforehand. Missing data anomaly might also occur due to technical failures (e.g. database crash). Since such failures cannot be anticipated during the design phase, we do not take them into consideration.

Unrelated Data Issue might occur when some data element needs to be read, but not required for the stated purpose.

Redundant Data Issue occurs when some data element is written but it is not accessed, i.e. read or deleted, by any subsequent element before the workflow execution is completed. Similar to the missing data issue, we consider it as a potential issue as the data item can be read by another workflow afterward.

In order to understand and to justify why and when missing data anomalies and redundant data anomalies might occur, we derive a state diagram which outlines a data life cycle of any data item. The state diagram, shown in Figure 4, illustrates which transitions lead to undesired behavior, namely missing data and redundant data issues. Transitions from one state to another are fired by either data operations (Read, Write, and Delete) or EOW (when the end state is reached). The state diagram has the initial state as *init*. Each data item is considered to be at *init* state initially, which means attribute name exists on the database schema, yet no value is written. A data item reaches $value\_exist$ state when a value is written by a data task, and $value\_read$ state represents when the value is read. Finally, the state diagram has three final (accepting) states as $REDUNDANT$, $MISSING$, and $SUCCESS$.

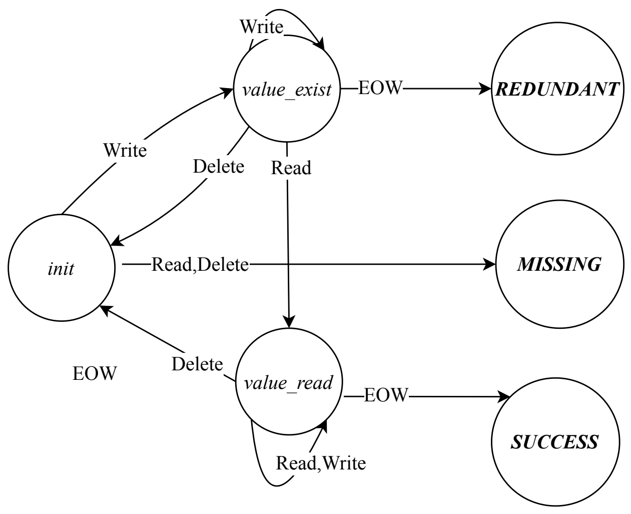


Fig. 4: State Diagram for Data Attributes.

In order to detect the unrelated data anomalies, one must also know which data items are required to accomplish the purpose given. We give a formal definition for a privacy policy in Definition 3.

Definition 3 (Privacy Policy) A privacy policy Pol consists of the set of rules which are represented as tuples p = (pr, {attrn}) where:

- pr is the reason for which data is collected, used, or disclosed,
- {atrn} is a set of attribute names which are required to be processed for the purpose given. □

For any policy rule $p \in Pol$, $getPurpose(p)$ returns its purpose and $getAtrn(p)$ returns the set of attribute names stated in p.

With the help of the state diagram and privacy policy, we design Algorithm 1 for detecting anomalies. The algorithm takes a data flow trace ϕ_I and privacy policy Pol as inputs. It returns as a result, whether or not the given data flow trace is compliant with the data minimization principle, that is, it is free from anomalies (missing data, unrelated data, and redundant data).

Within the algorithm, we also generate a list named $report$ to return details. A $report$ includes tuples as $(atrn, category, dt - id)$, where $atrn$ is an attribute name, $category \in \{MISSING, UNRELATED, REDUNDANT, SUCCESS\}$ is the state of the attribute, and $dt - id$ is the identifier of the respective data task. $SUCCESS$ is defined as the status of attributes without anomalies.

Algorithm 1: Detecting Data Minimization Anomalies

```
def DETECT-ANOMALIES($\phi$, $Pol$):
    /* Initialize a list report as (attribute-name, category, data-task-id) */
    initialize list $report = \oslash$
    category $\in \{MISSING, UNRELATED, REDUNDANT, SUCCESS\}$
    /* Initialize a temporary list temp of (attribute-name, status, data-task-id) */
    initialize list $temp = \oslash$
    status $\in \{init, value\_exist, value\_read\}$
    compliance = true
    /* compliance is a boolean flag which turns into false when we encounter an anomaly */
    size $\leftarrow \phi.size$
    for $i \leftarrow 0$ to $size - 1$ do
        $t \leftarrow \phi[i]$
        if $Write(t).notEmpty$ then
            foreach $atrn \in Write(t)$ do
                // atrn as attribute name
                if $\nexists\, atrn \in temp$ then
                    $temp.add(atrn, value\_exist, t.id)$
        if $Read(t).notEmpty$ then
            foreach $(atrn, purpose) \in Read(t)$ do
                if $\nexists p \in Pol : atrn \in p.getAttr() \wedge purpose == p.getPurpose()$ then
                    $report.add(atrn, UNRELATED, t.id)$
                    $compliant = false$
                else if $\nexists\, atrn \in temp$ then
                    $report.add(atrn, MISSING, t.id)$
                    $compliant = false$
                else if $temp.get(atrn.status == value\_exist$ then
                    $temp.update(atrn, value\_read, t.id)$
                else if $temp.get(atrn).status == init$ then
                    $report.add(atrn, MISSING, t.id)$
                    $compliant = false$
        if $Delete(t).notEmpty$ then
            foreach $atrn \in Delete(t)$ do
                if $\nexists\, atrn \in temp$ then
                    $report.add(atrn, MISSING, t.id)$
                    $compliant = false$
                else if $temp(atrn).status == init$ then
                    $report.add(atrn, MISSING, t.id)$
                    $compliant = false$
                else
                    $temp.update(atrn, init, t.id)$
    foreach $l \in temp$ do
        // EOW is reached
        switch $l.status$ do
            case $value\_exist$
                $report.add(l.attrn, REDUNDANT, l.task.id)$
                $compliant = false$
            otherwise
                $report.add(l.attrn, SUCCESS, l.task.id)$
    return $(compliant, report)$
```

We define three levels of compliance, as follows:

Fully Compliant represents a data-aware workflow, the data flow traces of which are all compliant with the data minimization principle.

Maybe Compliant represents a data-aware workflow only some of the data flow traces of which are compliant with the data minimization principle.

Non-compliant represents a data-aware workflow none of the data flow traces of which are compliant with the data minimization principle.

Algorithm 2 checks the compliance level of a data-aware workflow. It takes a set of all completed data flow traces $\Phi_{\mathcal{W}}$ and privacy policy Pol as input and determines the compliance level of the workflow $\mathcal{W}$ through checking compliance of each data flow trace $\phi \in \Phi_{\mathcal{W}}$.

Algorithm 2: Checking compliance level of a given data-aware workflow

```
/* Function for determining the compliance level of a workflow W */
def CHECK-COMPLIANCE-LEVEL($\Phi_{\mathcal{W}}$, $Pol$):
    input: $\Phi_{\mathcal{W}}$ as set of all completed data flow traces, $Pol$ as privacy policy
    initialize list $compList = \oslash$
    /* compList is a list of true, false to add compliance of each data flow trace $\phi$) */
    foreach $\phi \in \Phi_{\mathcal{W}}$ do
        compList.add (DETECT-ANOMALIES($\phi$, $Pol$))
        /* DETECT-ANOMALIES($\phi$, $Pol$) returns the true or false */
        if $\forall c \in compList : c == true$ then
            return FULLY-COMPLIANT
        else if $\exists c \in compList : c == false$ then
            return MAYBE-COMPLIANT
        else
            return NON-COMPLIANT
```

4.1 Applying the Detection Algorithms to Our Running Example

We apply the detection algorithms to our running example shown in Figure 3. As it is explained in Section 3, there is only one data flow trace $\phi = <t_1, t_2, t_3, t_4, t_5>$ for the example workflow.

Let policy rules be $p_1, p_2 \in Pol$ such that:

- $p_1 = (screening, \{blood - type, blood - sugar, result\})$
- $p_2 = (statistics, \{report\})$

When we run the Detect-Anomalies function in Algorithm 1 with data flow trace ϕ and the privacy policy Pol as inputs, it returns *false* which means that the given data flow trace ϕ is not compliant with the data minimization principle.

Detect-Anomalies function generates the list *report* as it is illustrated in Table 1.

Tab. 1: Report

	atrn	category	dt-id
report [0]	anamnesis	*REDUNDANT*	t_1
report [1]	blood-type	*REDUNDANT*	t_2
report [2]	blood-type	*UNRELATED*	t_3
report [3]	blood-sugar	*MISSING*	t_3
report [4]	result	*SUCCESS*	t_4
report [5]	report	*SUCCESS*	t_4
report [6]	report	*SUCCESS*	t_5

Three of the data tasks (t_1, t_2, and t_3) contain anomalies and we use *report* to warn modelers regarding these anomalies. Warnings as shown in Figure 5 help to comprehend potential problems. For instance, in the first data task (t_1), redundant data issue might occur for the attribute *anamnesis*. Modeler can either ignore this warning or repair the issue. Repair the issue option will be covered in next section.

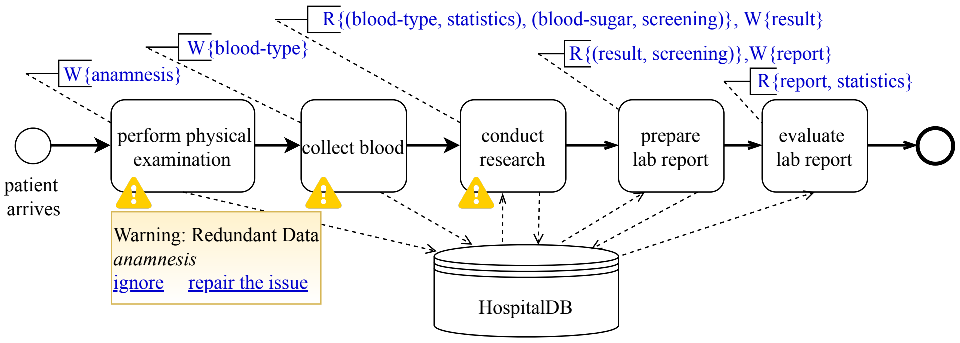


Fig. 5: Detecting anomalies and showing warnings accordingly.

Since there is only one data flow trace (ϕ), the set of all completed data flow traces $\Phi_{\mathcal{W}}$ includes only this data flow trace as well. Algorithm 2 with input $\Phi_{\mathcal{W}}$ returns $NON-COMPLIANT$ as there is no compliant data flow trace of the given data-aware workflow.

5 Repairing Data Minimization Anomalies

This section provides our pattern-based approach to repair the data minimization anomalies introduced in Section 4.

We design repair patterns similar to *exception handlers* proposed in [Le10]. One can consider our prespecified anomalies as anticipated exceptions.

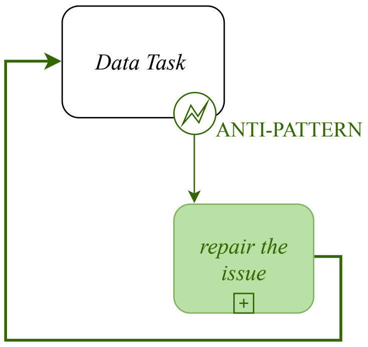


Fig. 6: Sample Repair Pattern.

Figure 6 shows a sample repair pattern. Assume a data minimization anomaly occurs within a data task. We add a boundary error event to the respective data task. A boundary error event throws an exception that can occur during the execution of the data task. When an exception is thrown, the respective data task is stopped, and instead, the repair action is taken. After the repair action, the data task is continued. The transformed parts within the figure are shown in green.

We design the repair actions as collapsed BPMN sub-processes which include a set of tasks to be executed. We do not specify the tasks within the collapsed sub-processes because different workflows might require different solutions. Instead, we only suggest high-level ideas for each anomaly category and we give the modelers responsibility to specify the exact actions to be taken.

5.1 Repair Strategy for Missing Data Issue

Missing data anomaly might occur in case of reading a data item before initializing it. During the design phase, we cannot determine the missing data problems due to database or system unavailability since they occur during execution. Hence, we do not take such issues into consideration. In order to solve the missing data issue, we suggest adding the repair pattern shown in Figure 7(a) to the concerned data task. *Data task* tries to read the value of a data attribute. If the value does not exist according to Detect-Anomalies function, $MISSING\_DATA$ is thrown and the repair action is taken. After handling missing data, the respective data task is continued to execute.

We give the modeler responsibility to specify the exact actions to be taken within handle missing data. First, they should check whether the data item is written by another workflow. If it is the case, they can simply ignore the warning. Otherwise, they should make sure that the value of the missing data item is written within the repair action. They can assume that a data task can simply write an attribute value just before the respective data task. However, in the real world, it is not always the case. Some data values can be generated only after some additional procedures (e.g. blood value after a lab experiment). Or some data values are written only after reaching the data subject (e.g. age of a patient). Since it is not possible to find a generic solution for all different scenarios, we again give modelers the responsibility to decide when and how to write the data.

5.2 Repair Strategy for Unrelated Data Issue

Unrelated data anomaly occurs when a data task reads some data element which is not required for the stated purpose. We suggest modelers add the repair pattern shown in Figure 7(b) to the concerned data task. One strategy to solve this problem is to delete the data task or to delete the unrelated data item. However, deleting components of the workflow triggers information loss and we do not intend to lose any information. Moreover, data might be maintained for other purposes within the workflow.

One alternative can be restricting access to the unrelated data instead of deleting it. Restriction of processing can be applied by different access control mechanisms (e.g. role-based access control model). In our research, the restriction should be related to the purpose information; therefore, one can apply a purpose-based access control model like in [Kr19] which means each query reading personal data must be associated with at least one purpose and access is only granted when data is required according to a given privacy policy.

5.3 Repair Strategy for Redundant Data Issue

Redundant data anomaly occurs when a data task writes some data element that is not accessed, i.e. read or delete, by any subsequent element. One strategy to solve this problem is to delete the respective data task with redundant data; however, similar to the unrelated data we do not want to lose information on the actual workflow. Instead, we suggest adding the repair pattern shown in Figure 7(c) to the concerned data task.

We give the modelers responsibility for specifying the exact actions to be taken to handle redundant data. First, they should decide whether the data item is used by another workflow afterward. If it is the case, they can simply ignore the warning. Otherwise, they can delete the redundant data item. Alternatively, they can consider making the data inaccessible instead of deleting it permanently. We do not explain the mechanisms to make data inaccessible since it is beyond our scope in this article.

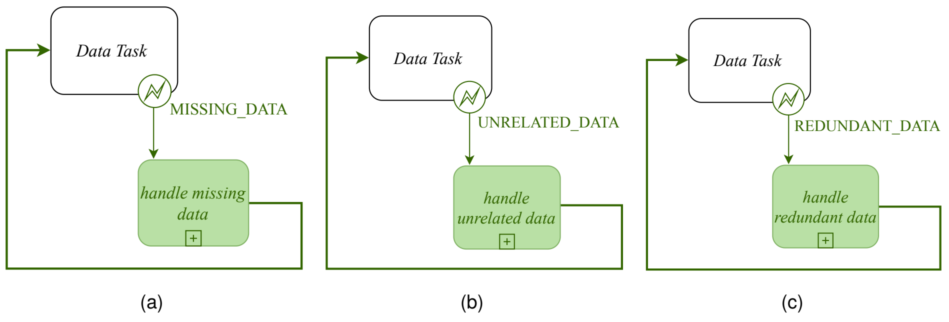


Fig. 7: Repair Strategies: (a) Pattern for Missing Data Issue, (b) Pattern for Unrelated Data Issue, (c) Pattern for Redundant Data Issue.

5.4 Overall Approach

We summarize our overall approach, as follows:

- Determine whether or not the given data-aware workflow is compliant via CHECK-COMPLIANCE-LEVEL function.
- For each data flow trace that is not fully compliant, categorize the anomalies through *report* list generated in DETECT-ANOMALIES function.
- Per category create a repair pattern as shown in Figure 7.
- Connect each data task to a corresponding repair pattern according to the type of its anomaly.

In order to increase the readability, we create repair patterns for each different category (missing data, unrelated data, and redundant data) instead of creating separate repair patterns for each anomaly.

5.5 Applying Repair Strategies to Our Running Example

We apply our repair approach to our running example shown in Figure 1. First, we determine that the given data-aware workflow is not-compliant via CHECK-COMPLIANCE-LEVEL function. We then identify the anomalies that occur in its data flow trace (ϕ). Through DETECT-ANOMALIES function, we figure out that three of its data tasks (t_1, t_2, and t_3) contain anomalies. Table 2 shows the *report* which includes only the attributes with anomalies.

In *report* list, we encounter anomalies with all three different categories; therefore, we create all three types of repair patterns shown in Figure 7. Finally, we connect each data

Tab. 2: Report including only ones with anomalies.

	atrn	category	dt-id
report [0]	anamnesis	*REDUNDANT*	t_1
report [1]	blood-type	*REDUNDANT*	t_2
report [2]	blood-type	*UNRELATED*	t_3
report [3]	blood-sugar	*MISSING*	t_3

task with an anomaly (t_1, t_2, and t_3) to a corresponding repair pattern according to the type of anomaly occur in it.

Applying repair strategies result in a transformed data-aware workflow shown in Figure 8. In this figure, we omit the data components and the data tasks without anomalies for simplification purpose. A repair pattern for redundant data issue is added to t_1 and t_2. For t_3, repair patterns for missing data issue and unrelated data issue are added.

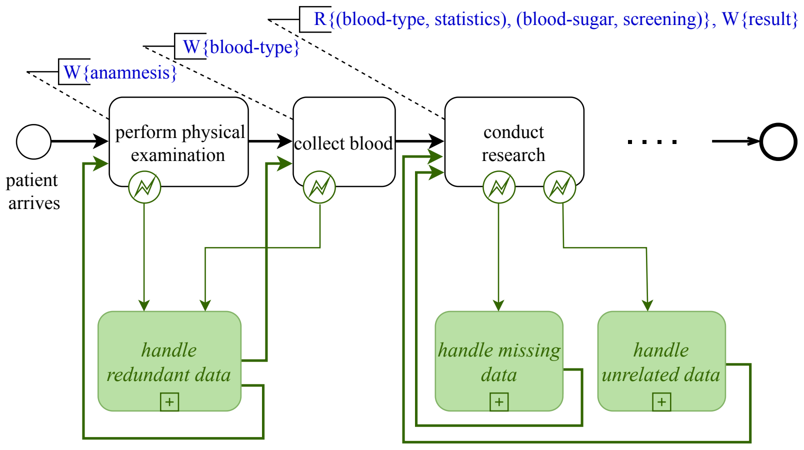


Fig. 8: Example BPMN-based workflow after applying repair strategies.

6 Discussion

In this section, we provide an analysis of our approach. First, it is prominent to highlight that in our approach we do not take inter-workflow dependencies (i.e. semantic dependencies among different workflows) into consideration. Therefore, our knowledge is limited to a given data-based workflow. However, in many real-world application scenarios workflows can be semantically interrelated. Since we do not investigate such dependencies, we do not know whether certain data elements are written or read by other workflows. For instance, we detect missing data anomaly when a data item is read before it is written. However, the data item can be written by another workflow. Hence, we do not make a certain statement, instead, we report a *potential* missing data anomaly. Similarly, redundant data issue occurs when a data item is written but not used by any subsequent component in a given data-aware workflow. We detect the issue as a *potential* anomaly as the data item might be used by another workflow at a later point in time.

Another important point to mention is that we force the modelers to add purpose information for reading operations, but not for writing operations. However, it does not mean that we encourage the organizations to write any personal data. Organizations can detect the redundant data anomalies during design time by means of our `Detect-Anomalies` function. It is their responsibility to take the required actions to handle this issue.

Finally, we create the repair patterns per category (missing data, unrelated data, and redundant data) instead of creating separate repair patterns for each of the anomalies. In this way, we aim to increase the readability of the workflow. However, we should also consider the positions of the data tasks while adding repair patterns. If the data tasks with the same type of anomalies are very far away from each other, connecting them to a single repair pattern might create more complexity. Fur such data tasks, alternatively, we can create separate repair patterns.

7 Related Work

Our research focuses on providing methods to handle the data minimization principle for workflows. There are related studies within the database and workflow management communities. In database field, [Ag02] can be considered as a pioneer of privacy-aware data management system (DBMS) design. Even though the proposal is from the pre-GDPR era, among its founding principles limited collection, limited use, and limited disclosure are very similar to the data minimization principle under GDPR. A recent proposal of a GDPR-compliant DBMS is presented in [Kr19]. [Ho14] suggests privacy design patterns to help organizations to protect privacy. These research efforts provide high-level ideas and strategies to achieve privacy protection from a database perspective which contributes to our approach to solve data minimization issues.

[Di17] provides patterns (e.g. anonymity, pseudonymity) to be applied to workflows with the goal to have “privacy-compliant” IT systems. [Ag19] presents a set of design patterns as workflows that enables organizations to tackle GDPR constraints. These works can be considered as a guide for an organization by specifying a specific sequence of steps to be taken for the respective privacy issues. Our work considerably differs as we detect problems on existing workflows and provide specific solutions for them.

[BDH18] propose their methodology to audit GDPR compliance for workflows. In their work, compliance with the data minimization principle is determined by judging whether data collected is actually used, which means it is not redundant. Their approach deals only with redundant data anomaly, not the missing data and unrelated data anomalies. They also do not provide any ideas to solve the anomalies.

Among all, we believe that the most related studies to our research take part in the field of data flow verification on workflows. [Co18], [Vo14], [SZNJ05] formally define data flow anomalies on workflows. They identify the anomalies regarding the correctness of data flow. However, we propose anomalies from the privacy point of view. For instance, these works consider inconsistent data anomaly which occurs when different versions of the same data exist as one of the most serious flaws. On the other hand, since we identify the anomalies from the privacy point of view, we do not consider inconsistent data as an anomaly. Moreover, they do not provide solutions to repair workflows in case of an anomaly. [Ka15] propose an approach for real-time data flow analysis which is triggered whenever a fragment is added to a workflow, instead of checking the entire model. It supports modelers by assisting in terms of data flow correctness. We check workflows after they are entirely modeled. [ADL09] describes a technique to automatically fix certain types of data anomalies that appears in the interplay between control flow and data flow. However, their data anomalies are not privacy related.

8 Conclusion and Future Work

It is important to note that the privacy-by-design concept under GDPR forces organizations to undertake a serious assessment of whether their technology design is compliant with data protection principles, including the data minimization principle. We argue that workflow technology contributes to ensuring privacy compliance during the design time. In this paper, we propose our approach to detect potential problems regarding the data minimization principle. Moreover, we present possible repair actions for BPMN-based workflows to be GDPR-compliant.

As future work, we would like to evaluate our approach against larger case studies to test its applicability. We also want to adapt our approach for a set of inter-related workflows and their dependencies to investigate the impact of data share among different workflows.

Bibliography

[ADL09] Awad, Ahmed; Decker, Gero; Lohmann, Niels: Diagnosing and repairing data anomalies in process models. In: International Conference on Business Process Management. Springer, pp. 5–16, 2009.

[Ag02] Agrawal, Rakesh; Kiernan, Jerry; Srikant, Ramakrishnan; Xu, Yirong: Hippocratic databases. In: VLDB’02: Proceedings of the 28th International Conference on Very Large Databases. Elsevier, pp. 143–154, 2002.

[Ag19] Agostinelli, Simone; Maggi, Fabrizio Maria; Marrella, Andrea; Sapio, Francesco: Achieving GDPR compliance of BPMN process models. In: International Conference on Advanced Information Systems Engineering. Springer, pp. 10–22, 2019.

[BDH18] Basin, David; Debois, Søren; Hildebrandt, Thomas: On purpose and by necessity: compliance under the GDPR. In: International Conference on Financial Cryptography and Data Security. Springer, pp. 20–37, 2018.

[Co18] Combi, Carlo; Oliboni, Barbara; Weske, Mathias; Zerbato, Francesca: Conceptual modeling of inter-dependencies between processes and data. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 110–119, 2018.

[DDO07] Dijkman, Remco M; Dumas, Marlon; Ouyang, Chun: Formal semantics and analysis of BPMN process models using Petri nets. Queensland University of Technology, Tech. Rep, 2007.

[Di17] Diamantopoulou, Vasiliki; Argyropoulos, Nikolaos; Kalloniatis, Christos; Gritzalis, Stefanos: Supporting the design of privacy-aware business processes via privacy process patterns. In: 2017 11th International Conference on Research Challenges in Information Science (RCIS). IEEE, pp. 187–198, 2017.

[Du13] Dumas, Marlon; La Rosa, Marcello; Mendling, Jan; Reijers, Hajo A: Business process management. Springer, 2013.

[Ho14] Hoepman, Jaap-Henk: Privacy design strategies. In: IFIP International Information Security Conference. Springer, pp. 446–459, 2014.

[Ka15] Kabbaj, Mohammed Issam; Bétari, Abdelkader; Bakkoury, Zohra; Rharbi, Assia: Towards an active help on detecting data flow errors in business process models. IJCSA, 12(1):16–25, 2015.

[Kr19] Kraska, Tim; Stonebraker, Michael; Brodie, Michael; Servan-Schreiber, Sacha; Weitzner, Daniel: SchengenDB: A Data Protection Database Proposal. In: Heterogeneous Data Management, Polystores, and Analytics for Healthcare, pp. 24–38. Springer, 2019.

[Le10] Lerner, Barbara Staudt; Christov, Stefan; Osterweil, Leon J; Bendraou, Reda; Kannengiesser, Udo; Wise, Alexander: Exception handling patterns for process modeling. IEEE Transactions on Software Engineering, 36(2):162–183, 2010.

[RW12] Reichert, Manfred; Weber, Barbara: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer Science & Business Media, 2012.

[SZNJ05] Sun, Sherry X; Zhao, J Leon; Nunamaker Jr, Jay F: On the theoretical foundation for data flow analysis in workflow management. AMCIS 2005 Proceedings, p. 188, 2005.

[TVdAS09] Trčka, Nikola; Van der Aalst, Wil MP; Sidorova, Natalia: Data-flow anti-patterns: Discovering data-flow errors in workflows. In: International Conference on Advanced Information Systems Engineering. Springer, pp. 425–439, 2009.

[Vo14] Von Stackelberg, Silvia; Putze, Susanne; Mülle, Jutta; Böhm, Klemens: Detecting data-flow errors in BPMN 2.0. Open Journal of Information Systems (OJIS), 1(2):1–19, 2014.

[We12] Weske, Mathias: Business process management architectures. In: Business Process Management, pp. 333–371. Springer, 2012.