Creating Privacy Policies From Data-Flow Diagrams

Jens Leicht, Marvin Wagner, and Maritta Heisel

paluno - The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany {jens.leicht, marvin.wagner, maritta.heisel}@uni-due.de

Abstract. Privacy policies are often used to fulfill the requirement of transparency of data protection legislation like the General Data Protection Regulation of the European Union. The privacy policies are used to describe how the data subject's data are handled by the data controller. Domain and legal experts mostly create these policies manually. We propose a tool-supported method to improve the creation of accurate privacy policies based on information from the development phase of a system. During privacy and security threat analyses information about system behavior is collected in form of data-flow diagrams. These diagrams describe which data flows from where to where within the system and to which external actors.
Based on this data-flow information we can create the basic structure of a privacy policy, already containing the data-flows. The extracted information is one of the most important parts of a privacy policy, providing transparency when data is transferred to external parties.

Keywords: Privacy Policy · Data-Flow Diagram · LINDDUN · Privacy Impact Analysis · Model-Based.

1 Introduction

Privacy policies are an important tool for service providers to comply with data protection legislation, like the General Data Protection Regulation of the European Union (GDPR) [5]. The creation of privacy policies is a time consuming and complex process, which we support by reusing information from the software development process.

Data-flow diagrams (DFDs) contain important information about the transfer of data in the context of a piece of software or a complete system. Information regarding internal data processing is also present in DFDs. This information covers some of the details that data controllers need to provide to data subjects via privacy policies. We present an automated method, including tool-support, which enables data controllers to retrieve relevant privacy policy information from their DFDs.

DFDs are often created in the context of privacy requirements engineering or privacy impact analyses, as required by the GDPR. They are also relevant in security threat modeling approaches, meaning that they are an important model

for privacy- or security critical software. For example, the LINDDUN method [4] makes use of DFDs to identify privacy threats. Since DFDs are created during the analysis we can use them with a minimal overhead to extract privacy relevant information from them, supporting policy authors in the creation of privacy policies. Updating the privacy policies based on updated DFDs can also improve the accuracy of the privacy policy when the system design changes.

Our method to derive parts of privacy policies from DFDs uses the Prolog-Layered Privacy Language (P-LPL) for the created privacy policies. In previous work we used P-LPL to perform automated GDPR-compliance checks on policies, providing feedback to the policy authors [12]. These compliance-checks are part of the Privacy Policy Compliance Guidance (PriPoCoG) framework. This framework also includes an access control methodology called P2BAC [11] which supports the enforcement of P-LPL privacy policies. By using P-LPL, we ensure compatibility with the PriPoCoG-framework.

The paper is structured as follows. We start with relevant background information in Section 2. Our concept is presented in Section 3, followed by a look at our DFD tool in Section 4. Next, we discuss related work in Section 5. Finally, we close this paper with a conclusion and a look at future work in Section 6.

2 Background

In this section we provide short introductions to the terminology used by the General Data Protection Regulation of the European Union (GDPR), the LINDDUN privacy threat modeling approach, data-flow diagrams, privacy policies, and the Eclipse Modeling Framework.

2.1 GDPR Terminology

The General Data Protection Regulation (GDPR) [5] introduces some terminology in the context of data handling and privacy policies, which we use throughout the paper to distinguish different roles and actors.

Data Subject is the person whose data are processed by the service provider (data controller). This is the person that we need to inform about any data handling, which is mostly done using privacy policies.

Data Controller is the person or legal entity in charge of controlling the data handling. This can be the service provider itself or a person/entity operating in the name of the service provider. The data controller specifies the privacy policy to inform its data subjects.

Data Processor (Data Recipient) is an external entity that processes some data on behalf of the data controller. Since the data processors receive the data from the controller, we also call these data recipients in the context of privacy policies.

Purpose is an explanation describing the reason for which data are processed. Privacy policies contain purposes explaining to the data subjects why their data are being handled.

2.2 LINDDUN

LINDDUN is a privacy threat modeling approach. The name is derived from the different types of privacy threats considered in the threat analysis: Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness (and Unintervenability), Non-compliance. The LINDDUN approach provides three methods for performing privacy analyses. In this paper we reference the LINDDUN Pro method, which uses data-flow diagrams to systematically analyze a system regarding privacy.

The LINDDUN Pro method begins with the definition of DFDs based on a high-level system description. It provides privacy threat trees and catalogues, which are used to identify threats in the DFDs. After identifying possible misuse scenarios and assessing and prioritizing identified risks, the method elicits privacy requirements and selects appropriate privacy enhancing technologies (see [4] for further details on the LINDDUN approach).

2.3 Data-Flow Diagrams

Data-flow diagrams (DFDs) are diagrams using five components that describe which process or external actor has access to which data by modeling data-flows within and out of the system. DFDs were introduced by DeMarco [3]. The following elements can be used in DFDs:

Process represents a system-internal process that handles some data. It is visualized by a circle (cf. *Data Bundling* in Fig. 1).

Actor represents an entity that may send or receive data. Actors are visualized by rectangles (cf. *Data Processor* and *Internal Data Miner* in Fig. 1).

Storage is used to represent internal data bases, file systems, or files that store some data they receive from processes or actors. They are visualized differently depending on the implementation of the DFD editor. We use the notation introduced by DeMarco [3], two horizontal lines surrounding the label of the storage (cf. *Database* in Fig. 1).

Data-Flow is a directed line between two diagram elements (process, actor, or storage), which is either annotated with a label or a list of data that flow along this connection (cf. *DF1* to *DF5* in Fig. 1). We use labels and provide tables with mappings from labels to their corresponding lists of data (cf. right-hand side of Fig. 2). The origin of a data-flow is called source, and the destination of a data-flow is called sink.

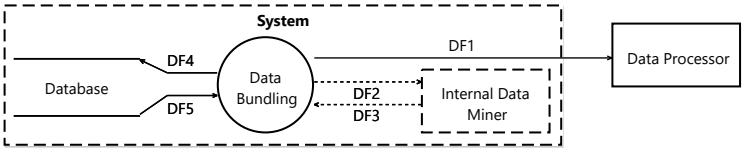


Fig. 1. Exemplary data-flow diagram.

Trust Boundary is a segment of a data-flow diagram surrounded with a dashed line (cf. *System* in Fig. 1). This segment can have different meanings, which should be defined when creating the DFD. We primarily use trust boundaries to differentiate internal processes, which are directly controlled by the data controller, from external actors.

All the above-described elements, except for the trust boundary, can optionally be marked as out-of-scope. This means that the elements are not considered in the analysis but might still be helpful in understanding the data-flows. Out-of-scope elements use a dashed outline/line (cf. *Internal Data Miner*, *DF2*, and *DF3* in Fig. 1). We also do not consider these elements when translating a DFD into a privacy policy.

2.4 Privacy Policies

To comply with the transparency requirements of the GDPR many data controllers make use of privacy policies.

Textual Policies A commonly used form of privacy policies is the textual form. Data controllers verbally describe how they process the data subject's data and explain the reasons behind the processing (purposes). Excerpts from the Amazon.de privacy policy can be found in Listings 1, 2, and 3 in Section 3.2. We use a more formal and computer processable form of privacy policies as described below.

P-LPL Policies The Prolog-Layered Privacy Language (P-LPL) [12] is a derivative of the Layered Privacy Language (LPL) [6] and provides a computer processable form for expressing GDPR-compliant privacy policies. P-LPL is part of the Privacy Policy Compliance Guidance (PriPoCoG)-framework [12], which provides GDPR-compliance checks for P-LPL policies, as well as access control based on these policies using P2BAC [11].

P-LPL policies use a hierarchical structure to arrange processing purposes, which we make use of as described in Section 3.5. All elements of a P-LPL policy have titles and descriptions that are used for the policy representation towards the data subjects.

3 Methodology

In this section we present the general concept and provide a detailed explanation of the generation of privacy policy information from data-flow diagrams.

3.1 Concept

Creating privacy policies from DFDs is achieved by performing the following steps:

1. **Creation of DFDs of the system behavior using our tool (cf. Section 4).** This can be done as part of privacy or security threat analyses, e.g., using LINDDUN [4] or STRIDE [9], or independently of any threat modeling technique. During privacy impact analyses (PIAs), DFDs also play an important role. The DFDs created in this step also help document the system behavior. If the DFDs have shared components, for example, recurring actors, these shared components need to be imported from the main model (cf. Section 4) to prevent duplicate elements representing the same entity. During this step it is important to not introduce ambiguities in the DFDs, e.g., the term *address* can have many different interpretations of what data are included in an address. Travis Breaux's and Mitra Bokaei-Hosseini's Ontology of Personal Information[1] (OPI) shows that these ambiguities can occur for many terms used in data flows and privacy policies. The solution is to name each data element as precisely as possible. Alternatively, a mapping table could be introduced revealing what data is contained in a specific term. Ambiguities in the DFDs result in the same ambiguities in the privacy policy.
2. **Importing combined model into privacy policy editor.** All DFDs created within a project using our tool share a single model. Hence, they can be visualized in a single Privacy Data-Flow Diagram representing all data-flows of the system. That diagram can be imported into our privacy policy editor (cf. Section 3.6).
3. **Creation of intermediate privacy policy.** The editor extracts all available information from the diagram, as explained in Section 3.5 below. Since the extracted information is not sufficient to create a complete privacy policy, we call it an intermediate privacy policy.
4. **Manually completing the privacy policy.** The policy author can now manually edit the intermediate policy, entering further details. The final output is a P-LPL policy that can be checked for GDPR-compliance using PriPoCoG [12].

Deriving the main components of a privacy policy from DFDs modeling the system behavior has the following benefits: *1.* The complexity of the task of creating a privacy policy is reduced. *2.* The resulting policy can be checked for GDPR-compliance using the PriPoCoG-framework. *3.* The resulting policy accurately describes the actual system behavior. The chance of discrepancies between policy and system behavior is reduced. *4.* If the system is adapted and the DFDs are updated according to the new system behavior, the policy can be updated, too. Thus, further improving the accuracy of the policy.

In the security engineering context (e.g., STRIDE [9]) DFDs become more complex compared to DFDs from the privacy engineering context (e.g., LINDDUN). This increased complexity leads to more detailed, and therefore more complex, privacy policies. However, the use of a purpose hierarchy, as described in Section 3.5 (Step 4) below, increases the comprehensibility of such detailed privacy policies. The top-level purposes give a general overview over the data

[1] `https://opi.cs.cmu.edu/show/address`

processed by a data controller, whereas the sub-purposes can provide more details to the interested reader.

In the future, layered DFDs could be used to create a more abstract top-level DFD that is more suitable for the privacy policy creation. The details needed in the security context could then be placed in lower-level DFDs, refining the processes presented in the top-level DFD.

In the following we introduce a running example which we use as a guide through the process of extracting privacy policy-relevant information from DFDs.

3.2 Running Example

As a running example we use two sections of the *Amazon.de* [1] privacy policy and create a DFD from the described behavior. In general, the DFDs for our approach will not be created based on existing privacy policies, but instead from actual system behavior. We just make use of an existing privacy policy, as we have no insights into the actual system behavior of the *Amazon.de* systems. In our running example we focus on the delivery of products, as well as the processing of payments. All other mentioned processing purposes will not be considered in this paper. Listing 1 shows how *Amazon.de* describes the purposes for which they process personal data. In Listing 2 we can see in which cases the personal data of the data subject are transferred to third-party service providers. Amazon also receives updated personal data from some third-party service providers, e.g., a corrected delivery address from delivery partners (cf. Listing 3).

> ‘‘ Purchase and delivery of products and services. We use your personal information to take and handle orders, deliver products and services, process payments, and communicate with you about orders, products and services, and promotional offers .’’

Listing 1. Excerpt from the *Amazon.de* privacy policy (purposes) [1].

> ‘‘ Third party service providers: We employ other companies and individuals to perform functions on our behalf. Examples include fulfilling orders for products or services, delivering packages, [...], processing payments [...].’’

Listing 2. Excerpt from the *Amazon.de* privacy policy (data processors) [1].

> ‘‘ updated delivery and address information from our carriers or other third parties, which we use to correct our records and deliver your next purchase or communication more easily’’

Listing 3. Excerpt from the *Amazon.de* privacy policy (data from other sources) [1].

Since the *Amazon.de* privacy policy is not very detailed in stating which data are transferred to which third-party service provider, we assume that the following data are transferred for the purposes *product delivery* and *payment processing*:

– **product delivery**
  - Name
  - Address
  - Phone Number
  - E-Mail Address

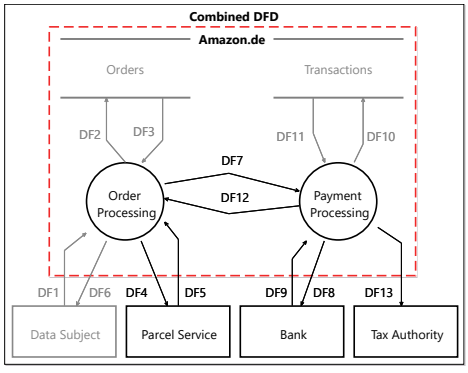
– **payment processing**
  - Amount due
  - Name
  - Address
  - Bank Account Number

A data-flow diagram for this scenario is shown in Fig. 2. This diagram differs from a normal DFD because it was created using our DFD-tool and already contains annotations, which we explain in further detail in Sections 3.4 and 4.3 below. Without annotations the actor *Data Subject* and its related data-flows would not be greyed-out. The storages *Orders* and *Transactions* are greyed-out because the diagram shown is a *PrivacyDataFlowDiagram* (cf. Section 4.3).

The data subject places an order by providing all necessary information: Order(the items ordered), Name, Address, Phone Number, Mail Address, and Bank Account Number (*DF1*). Order details are processed by *Order Processing* and stored in *Orders* (*DF2* + *DF3*).

The *Payment Processing* process receives Order, Name, Address, and Bank Account Number from *Order Processing* (*DF7*) and forwards the Amount due (extracted from Order), Name, Address, and Bank Account Number to the *Bank* (*DF8*). Transaction details of the bank transfer are returned to the *Payment Processing* process and used to decide whether the product delivery should be triggered (*DF9*). All transaction-related information (Order, Name, Address, Bank Account Number, and Transaction Details are stored in *Transactions* (*DF10* + *DF11*). Transaction Details, Name, and Address will be forwarded to the *Tax Authorities* if required (*DF13*).

The Transaction Confirmation (based on the success of a transaction) is internally (i.e., inside the Amazon.de trust boundary) forwarded to the process *Order Processing* (*DF12*) and from there stored in *Orders* (*DF2* + *DF3*). *Order Processing* forwards Name, Address, Phone Number, and E-Mail Address



DF and Data
DF1 | Order, Name, Address, Phone Number, E-Mail Address, Bank Account Number
DF2 | Order, Name, Address, Phone Number, E-Mail Address, Delivery Status, Transaction Confirmation
DF3 | Order, Name, Address, Phone Number, E-Mail Address, Delivery Status, Transaction Confirmation
DF4 | Name, Address, Phone Number, E-Mail Address
DF5 | Delivery Status, Address, Name
DF6 | Order History, Payment Status, Delivery Status
DF7 | Order, Name, Address, Bank Account Number
DF8 | Amount, Name, Address, Bank Account Number
DF9 | Transaction Details
DF10 | Order, Name, Address, Bank Account Number, Transaction Details
DF11 | Order, Name, Address, Bank Account Number, Transaction Details
DF12 | Order, Transaction Confirmation
DF13 | Name, Address, Transaction Details

Fig. 2. left: Combined data-flow diagram for our running example, data subject and storages are greyed-out based on annotations (cf. Section 3.4); right: Data-flows and their corresponding data.

to the *Parcel Service* for product delivery (*DF4*). The *Parcel Service* returns the current delivery status, as well as a potentially updated address and name (cf. Listing 3) to the *Order Processing* process (*DF5*), which is then stored in *Orders* (*DF2* + *DF3*). The *Data Subject*, receives an Order History (based on the past orders stored in *Orders*), a Payment Status (based on the *Transaction Confirmation*), and the Delivery Status (*DF6*).

3.3 Validation Conditions

To reduce the number of errors that can happen during the process of defining DFDs as well as extracting privacy policy information from such diagrams, we define validation conditions. These conditions can be checked manually, especially when drawing DFDs by hand. However, we also implement these validation conditions in our DFD-tool, as described in Section 4.4, so that they can be checked by the tool when creating the DFDs.

We identified the following 14 validation conditions. Some of these conditions are relevant for data-flow diagrams in general, and some are introduced to be able to transform data-flow diagrams into privacy policies. Conditions 1 to 10 apply to data-flow diagrams in general. Conditions 11 to 14 are specific to Privacy Data-Flow Diagrams.

General Data-Flow Diagrams:

1. Each data-flow needs a source and a sink.
2. Each data-flow needs at least one assigned data-element.
3. Processes need to be located inside a trust boundary.
4. Storages need to be located inside a trust boundary.
5. Source and sink of a data-flow must not both be of type storage.
6. Source and sink of a data-flow must not both be of type actor.
7. Source and sink of a data-flow need to be different.
8. An actor must be source or sink of at least one data-flow.
9. An element cannot be inside more than one trust boundary.
10. Elements cannot be inside and outside a trust boundary at the same time.

Privacy Data-Flow Diagrams:

11. Each data-element needs to be referenced by at least one data-flow.
12. Each data-flow diagram needs at least one process.
13. Each data-flow diagram needs at most one data subject.
14. When the combined data-flow diagram contains more than one trust boundary, only one of the trust boundaries can be considered as the data controller.

Condition 11 ensures that the created privacy policy does not contain unnecessary information about data that is not actually flowing anywhere. Adding such unnecessary information would clutter the resulting privacy policy, reducing transparency towards the data subject. Condition 12 ensures that information from the data-flow diagram can be combined into a purpose inside the privacy policy (cf. Section 3.5). A process is also required for a data-flow analysis to be

sensible, as data-flows without a process (directly between actors) will not be of interest for the party conducting the analysis. Such a data-flow would only be relevant for a data-flow analysis conducted by the actors themselves. Condition 13 assures that the data subject used in the diagrams represents a single data subject, which is congruent with a privacy policy that represents all necessary information about the data of the single data subject reading the policy. Finally, condition 14 assures that trust boundaries, used for example for groups of external actors, are not considered to be part of the system. This means that the combined Privacy DFD has at most one trust boundary representing the data controller's system. Any additional trust boundary will be considered external.

3.4 Annotated Data-Flow Diagram

In addition to creating the data-flow diagrams using our tool, privacy policy authors will need to annotate the data-flow diagrams. The overhead, however, is very small as they only need to mark the one actor representing the data subject, as well as the trust boundary that represents the data controller. It is also possible that the DFDs contain no actor representing the data subject. In this case only the trust boundary of the data controller needs to be annotated in the DFDs.

Optionally, authors can add descriptions for each element in the diagram. These descriptions are used to create user-friendly representations of the elements inside the privacy policy. For the basic privacy policy elements: purpose, data, and data recipient the descriptions are used directly to describe these elements. Data-flows do not have a direct representation in the privacy policy. Hence, we combine all data-flow descriptions in the description of the purpose, which is part of the policy.

Further information that may be required for a GDPR-compliant privacy policy needs to be entered manually using our work-in-progress privacy policy editor, which we describe in more detail in Section 3.6 below. Missing information includes, for example, the rights of the data subject or information about data controllers or data protection officers. An alternative way of entering some of the additional information could be to further annotate the diagrams. However, we expect a better usability when entering the data using the policy editor, instead of further annotating the diagrams.

3.5 Intermediate Policy

The combined data-flow diagram (cf. Step 2 in Section 3.1 and Section 4.3) is exported from our DFD-tool as an XML-file. This file is then imported into our privacy policy editor (cf. Section 3.6) which extracts all available information from the XML-file and creates the corresponding privacy policy elements as described below.

Table 1 provides an overview of the processes shown in Fig. 2, their related data-flows, and the data included, as well as the actors involved in these data-flows. The intermediate policy (listed in Table 2) is created based on the

Table 1. Summary of the elements of the DFD shown in Fig. 2.

Process	Data-Flows	Data	Actors
Order Processing	DF1-DF7, DF12	Order, Name, Address, Phone Number, E-Mail Address, Bank Account Number, Delivery Status, Order History, Payment Status, Transaction Confirmation	Parcel Service
Payment Processing	DF7-DF13	Order, Name, Address, Bank Account Number, Amount, Transaction Details, Transaction Confirmation	Bank, Tax Authority

information shown in Table 1 using the following procedure:

1. For each data-element used in the combined DFD, a corresponding element is created in the policy editor. If source or sink of all data-flows containing a data-element are marked as out-of-scope, the data-element will not be translated into the privacy policy. The same applies if all relevant data-flows themselves are marked out-of-scope. The name of the data-element as well as the optional description annotation are used to pre-fill the editor with additional information about the element.
2. For each actor that is either sink or source of a data-flow from or to a process inside the data controller (annotated trust boundary, cf. Section 3.4), a data recipient element is created in the policy editor. Actors that are marked as out-of-scope and the data subject are not translated into the policy.
3. A purpose element is created for each pair of *process* and *actor* that are connected via a data-flow. Again, data-flows marked out-of-scope and to or from the data subject are not considered here. *Processes*, that do not have any external actor as data recipient, are still translated to corresponding purposes in the policy. These purposes describe data processing by the data controller and are therefore also relevant for a privacy policy. The data-elements of all incoming and outgoing data-flows of such processes are added to the resulting purpose.
4. If a *process* is connected to multiple *actors*, an additional purpose is created, that combines all purposes created for this *process* in the previous step. The purposes created during this step of the translation are arranged in a purpose hierarchy, which, for our running example, is shown as a screenshot from our privacy policy editor in Fig. 3. For the process *Payment Processing* two purposes are created (*Bank* and *Tax Authority*) and combined in the main *Payment Processing* purpose.
5. The descriptions of the purposes, if supplied in the annotated DFDs (cf. 3.4), are created based on the description of the *process*, as well as the descriptions of the data-flows from and to the *process*. Listings 4 and 5 show the exemplary descriptions of *DF8* and *DF9*. Combined with the description of the

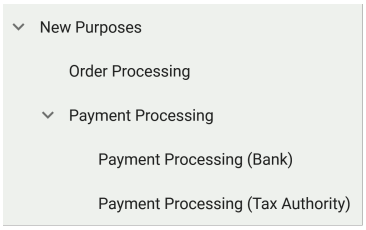


Fig. 3. Purpose hierarchy of the intermediate policy.

Payment Processing process, the resulting description of the corresponding child purpose *Payment Processing (Bank)* is shown in Listing 6.

6. For hierarchical purposes, the descriptions are a concatenation of the descriptions of the child purposes. Listing 7 shows the description of the parent purpose *Payment Processing* combining Listing 6 with the description of the second child purpose *Payment Processing (Tax Authority)*.
7. Finally, data and data recipients are assigned to the corresponding purposes. This assignment is based on all data-flows between a process and an actor. The hierarchical purposes that combine multiple purposes are assigned with the union of data-elements and data recipients of their child purposes.

This translation procedure is supported by our DFD-tool as well as our privacy policy editor. However, it can also be applied to any DFD and any form of privacy policy, e.g., when manually translating DFDs into a policy.

"Data are transferred to our bank to process the payment for your order."

Listing 4. Exemplary description of DF8.

"The bank gives us access to transaction details after a payment has been processed."

Listing 5. Exemplary description of DF9.

"We process your data in order to carry out financial transactions for the payment of your orders. Data are transferred to our bank to process the payment for your order. The bank gives us access to transaction details after a payment has been processed."

Listing 6. Exemplary description of the child purpose *Payment Processing (Bank)*.

"We process your data in order to carry out financial transactions for the payment of your orders. Data are transferred to our bank to process the payment for your order. The bank gives us access to transaction details after a payment has been processed. We forward your data to the tax authority as required by law."

Listing 7. Exemplary description of the parent purpose *Payment Processing*.

Table 2. Policy elements created from the DFD shown in Fig. 2 (cf. Table 1).

Purpose	**Data**	**Data Recipients**	**Sub-purposes**
Order Processing	Order, Name, Address, Phone Number, E-Mail Address, Bank Account Number, Delivery Status, Order History, Payment Status, Transaction Confirmation	Parcel Service	-
Payment Processing	Order, Name, Address, Bank Account Number, Amount, Transaction Details, Transaction Confirmation	Bank, Tax Authority	Payment Processing (Bank), Payment Processing (Tax Authority)
Payment Processing (Bank)	Amount, Name, Address, Bank Account Number, Transaction Details	Bank	-
Payment Processing (Tax Authority)	Name, Address Transaction Details	Tax Authority	-

Table 2 shows for each purpose created what data are used for this purpose and who the data recipients are. Additionally, for the purpose *Payment Processing* its sub-purposes *Payment Processing (Bank)* and *Payment Processing (Tax Authority)* are shown. The parent purpose contains all data-elements, as well as all data recipients of its children. We explain how this information is used in the privacy policy editor in Section 3.6 below.

3.6 Privacy Policy Editor

The intermediate policy discussed in the previous section can be used in our privacy policy editor to create a GDPR-compliant privacy policy. The policy author can load the intermediate policy to fill parts of the privacy policy with the information gathered from the DFDs. An excerpt from the main page of our currently work-in-progress policy editor is shown in Fig. 4. The editor highlights the data, data recipients, and purpose tiles in red, because these contain some of the information from the intermediate policy, but there is still information missing for a complete privacy policy.

The overall policy additionally requires the following information:

- essential policy information (e.g., the language used or a reference to a textual privacy policy),
- information about the data controllers,
- information about data protection officers,
- a list of rights that the controller grants the data subjects,
- and information regarding the competent supervisory authority.

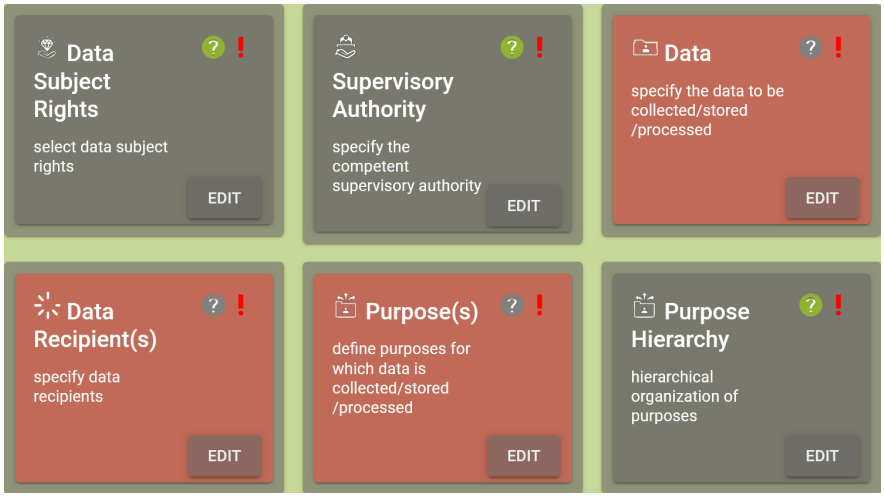


Fig. 4. Excerpt from the main page of the PriPoCoG privacy policy editor pre-filled with information from the DFD shown in Fig. 2.

The data-elements imported from the DFDs are missing the following information: data type, sensitivity level (e.g., explicit, sensitive, or non-sensitive). The data recipients are missing information regarding their classification as either a person, legal entity, or public authority. Regarding the purposes, authors need to decide whether the data subject must accept the purpose, or whether this is optional. Purposes additionally lack information regarding data retention and the legal bases on which a purpose is based.

Although the above-mentioned information needs to be added manually, most of the policy is filled with the information from the DFDs. The purposes and their corresponding data and data recipients are the largest part of the privacy policy. The proportion between the pre-filled elements and the data to be added manually depends on the size of the system under consideration.

Once the policy author enters the missing information manually the tiles will turn green to show that these parts of the privacy policy are complete. The grey tiles indicate that no information has yet been entered in these categories. When sufficient information is entered, the policy can be checked for GDPR-compliance. We explain how we perform compliance-checks on the privacy policy in [12].

4 Tool Support

To help users of our approach create DFDs, we provide a graphical editor. It is based on the *Eclipse Modeling Framework (EMF)*[2] and *Sirius*[3]. Sirius builds on

[2] `https://www.eclipse.org/modeling/emf/`
[3] `https://www.eclipse.org/sirius/`

EMF and the *Acceleo Query Language (AQL)*. AQL is a specification language similar to the *Object Constraint Language (OCL)*[4]. The elements of an EMF metamodel can be filtered, created, deleted, and manipulated with AQL. In the following we describe the main elements of the metamodel, as well as the different graphical representations and the implementation of the validation conditions.

4.1 Metamodel

Using EMF we defined the metamodel shown in Fig. 5, which defines all elements of a data-flow diagram. To reduce the complexity of the metamodel for this paper, we removed two abstract classes that are used to introduce some shared attributes. Each element of the model has a name and a description (not shown in Fig. 5 to reduce complexity).

Our main element is *DataFlowModel*. It contains all other elements. Furthermore, we have the elements *TrustBoundary*, *Actor*, *Storage*, *Data*, *DataFlow*, *Process*, and *DataFlowDiagram*, which are contained in the element *DataFlowModel*. The elements: *Actor*, *Process*, *Storage*, and *DataFlow* have a boolean attribute *OutOfScope*(not shown in Fig. 5 to reduce complexity), because some elements may not be relevant or out-of-scope for a data-flow analysis.

In addition to all standard elements of a DFD, we added a boolean attribute to the *Actor* element: *dataSubject*. This attribute is used to exclude actors that

[4] `https://www.omg.org/spec/OCL/`

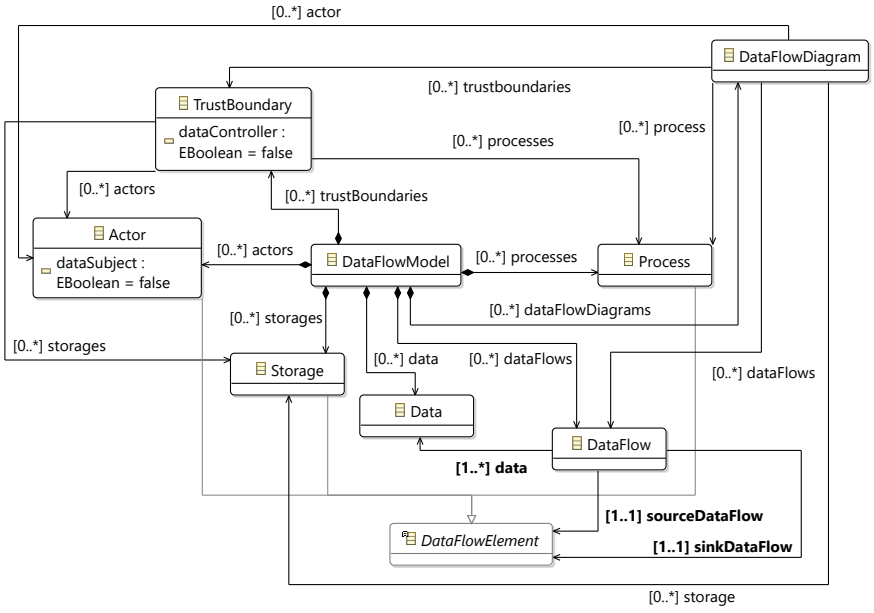


Fig. 5. EMF-metamodel describing all elements of a data-flow diagram and their relations.

represent the data subject from further processing of the diagram. The data subject is not a data recipient in the context of a privacy policy; hence, we exclude it from our translation process. Additionally, we added a boolean attribute *dataController* to the *TrustBoundary*. We use this attribute to highlight the trust boundary of the data controller. For multiple DFDs, we decide that *Actor*, *Storage*, *DataFlow*, and *Process* can be assigned to the element *DataFlowDiagram*. Thus, we can obtain a better overview by showing smaller DFDs that together form the whole model (cf. Fig. 7 vs. Fig. 2).

Similarly, we assign elements to the *TrustBoundary*, because in this way we can define which elements are inside the trust boundary. A *DataFlow* can take place between two *DataFlowElement*s which can be *Storage*, *Process*, or *Actor* and mandatorily needs a *sourceDataFlow* and a *sinkDataFlow*. Additionally, *DataFlow* must be assigned at least one or more elements of *Data*. These restrictions, requiring at least one element are highlighted in **bold** in Fig. 5.

The metamodel is part of the editor's back-end and is not visible to the user of the tool.

4.2 Model Instance

An instance of the metamodel is a data-flow model. It contains instantiated metamodel elements from a concrete data-flow model. The model instance can be created and modified via graphical representations, described below. Additionally, the editor allows storing the results of the modeling process persistently.

The tree view of the model instance from our running example is shown in Fig. 6. It contains all elements of the model. The tree view of the model instance is not comprehensible for the user. Therefore, we provide a graphical editor.

A data-flow model can contain multiple DFDs, which can have shared elements. These shared elements occur only once in the data-flow model, but are referenced by each DFD they appear in. An example of such shared elements are

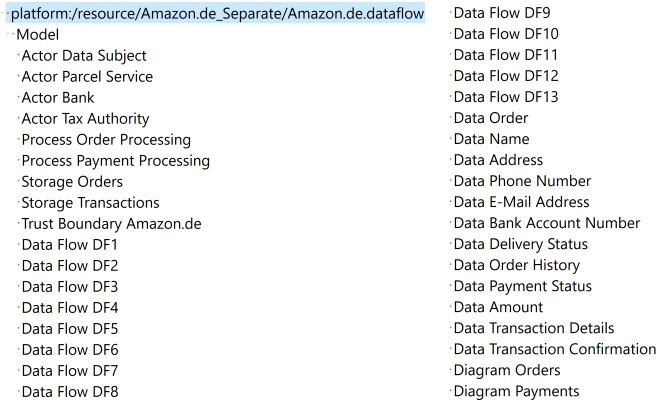


Fig. 6. Tree view of the model instance from our running example (cf. Fig. 2).

the processes and the data-flows *DF7* and *DF12* in Fig. 7, which only appear once in the combined DFD (cf. Fig. 2) and the tree view instance in Fig. 6.

The model instances are part of the backend, too. The user of the tool only interacts with the graphical representations, described in the following.

4.3 Graphical Representations

We have three different graphical representations of the model instance. All three representations share the fact that trust boundaries, which are marked as data controllers, are highlighted in red. Trust boundaries that do not represent the data controller are drawn in black.

We have two graphical representations for DFDs called *DataFlowDiagram* and *PrivacyDataFlowDiagram*.

Both diagrams are almost equal. The *DataFlowDiagram* represents a selected part of the entire model. Only the elements which are assigned to the *DataFlowDiagram* are shown. This gives the user a clear overview. This representation is used for general purpose data-flow diagrams. The DFDs in Fig. 7 are created using the *DataFlowDiagram* representation.

The *PrivacyDataFlowDiagram* has two special properties compared to the DFD representation. We grey-out all storages, the data subject, and all data-flows which have storages or the data subject as source or sink. They are greyed-out because they are not relevant for the translation into a privacy policy.

Additionally, we have a main diagram called *DataFlowMainDiagram* where all elements are represented at once in a combined DFD. The represented elements are the union of all elements from the separate DFDs. This main diagram gives a complete overview of the entire model. However, the large number of elements can be overwhelming for the user. Hence, we recommend defining separate DFDs first. The *DataFlowMainDiagram* representation uses a *PrivacyDataFlowDiagram* representation for the combined model. The DFD shown in Fig. 2 is

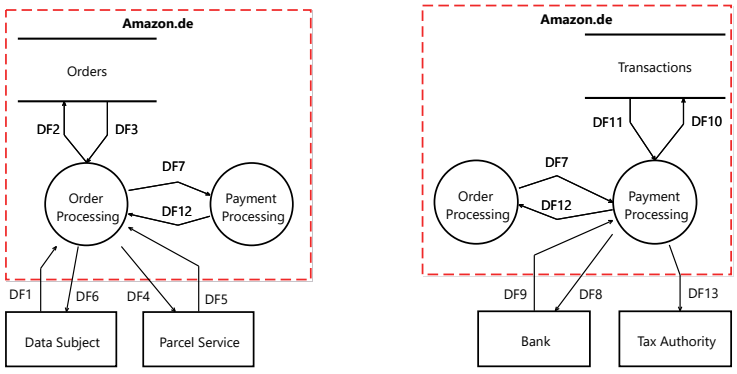


Fig. 7. left: Data-flow diagram for *Order Processing*; right: Data-flow diagram for *Payment Processing*.

the *DataFlowMainDiagram* combining the two separate DFDs for *Order Processing* and *Payment Processing* shown in Fig. 7. The combined diagram is used for exporting the DFDs to the privacy policy editor (cf. Section 3.6).

4.4 Implemented Validation Conditions (VCs)

The validation conditions of section 3.3 are implemented in the editor using AQL. They are checked automatically after a trigger of the user.

Listing 8 shows the AQL implementation of VC 14 as an example: *"When the combined data-flow diagram contains more than one trust boundary, only one of the trust boundaries can be considered as the data controller."*

```
aql: self.trustBoundaries->size()>1 implies self.
    trustBoundaries->one(t|t.dataController=true)
```

Listing 8. AQL implementation of VC 14.

The context of the VC is the element *DataFlowModel* (see Fig. 5). Therefore, *self* is of type *DataFlowModel*. The first part *self.trustBoundaries− >size()>1* is evaluated to true or false. If we have more than one trust boundary in our model, it is evaluated to true. If the first part is true, the second part after the ***implies*** *self.trustBoundaries − > one(t|t.dataController=true)* needs to be true. This is the case if the set of all trust boundaries contains exactly one trust boundary where the attribute *dataController* is set to *true*.

The other VCs are implemented similarly or covered by the constraints specified in the metamodel.

5 Related Work

Robles-González et al. propose a framework extending LINDDUN, specifically targeting identification and authentication processes [13]. We do not extend the privacy threat analysis, but instead make use of the documentation created during this process. Even extended/derived frameworks like the one of Robles-González et al. are compatible with our proposed method, as long as they continue to use data-flow diagrams for their privacy analyses.

Before LINDDUN was proposed in the privacy context, Microsoft created STRIDE in the context of security threat modeling [9]. This security threat model also uses data-flow diagrams and hence is also compatible with our method. Since STRIDE focusses on security instead of privacy, the resulting privacy policy may be lacking some data transfers, that may be identified using LINDDUN. However, using DFDs from STRIDE will still support privacy policy authors when defining a policy, as many data-flows will be covered by the STRIDE DFDs, preparing large parts of the privacy policy.

Since we support privacy policy authors in the process of creating privacy policies, by providing a tool-supported method to extract policy information from DFDs, we also want to mention the work of Hjerppe et al. [7]. They provide

a method for automatically creating LPL-policies from annotated source code. Depending on the size of a project it might be more viable to use the annotated source code to generate the privacy policy. However, in projects where a privacy impact analysis or privacy threat modeling using LINDDUN is applied, our method transforms knowledge created during these analyses into useful content for a privacy policy.

Kunz et al. also use a model-based approach in the privacy context [10]. They propose *Privacy Property Graphs* for privacy threat analyses. These privacy property graphs are created from static code analysis and are enriched data-flow diagrams. This provides the potential for an adapted version of our work presented in this paper. This adaptation could take the automatically generated privacy property graphs as input to prepare the privacy policies with less manual overhead.

Wang et al. analyzed 120 android apps concerning discrepancies between privacy policies and app behavior [14]. They identified 21 strong and 18 weak violations of the provided privacy policies. This means that 39 apps did not behave as described in their accompanying privacy policies. Using our approach and DFDs modeling the app behavior these inaccuracies in the policies could be prevented.

With the same goal Andow et al. proposed a different approach for the analysis of such discrepancies [2]. They also take the data recipient into account when verifying the behavior of the application. With this detail considered they analyzed 13796 applications and came to the conclusion that 42.4% of these applications had discrepancies between app behavior and privacy policy.

GDPR-compliance in systems and privacy policies is also considered an important topic by the European Union. It, for example, funded a recent research project for the assessment of GDPR-compliance. Completed in 2021, the DEFeND[5] project supports data controllers in the planning, design, and operational phases.

6 Conclusion & Future Work

Conclusion We presented our tool-supported method for the extraction of privacy policy information from data-flow diagrams. This allows policy authors to reuse information from privacy and security threat analyses when creating privacy policies for their services.

Our method and tool improve the creation of privacy policies by automatically extracting information regarding data-flows from these diagrams and providing this information as purposes and data recipients in our privacy policy editor. DFDs from the security engineering context can be more complex compared to the ones from the LINDDUN approach. Using a purpose hierarchy, we can combine the detailed information contained in these DFDs into more general purposes, which better fit a privacy policy.

[5] `https://www.defendproject.eu/`

Although the policy author still needs to enter some information manually, a large part of the privacy policy can be pre-filled using our approach. The definition of purposes and their data and data recipients takes up a large part of the policy definition process. This time-consuming task is made easier by importing the information from the DFDs.

Additionally, extracting this information from models representing the system behavior can improve the accuracy of the privacy policies. When the policies are not created independently of the system they more closely represent the actual system behavior.

Future Work The approach we presented in this paper can be extended to support layered DFDs. This could further improve the purpose hierarchy created from the DFDs.

The integration of further privacy-related methods into the privacy policy creation process is a promising task for the future. The aim is to extract as much policy information as possible from work that has already been done during the development of a system. Thus, we reduce the overhead occurring in the privacy policy creation process.

Another goal for the future is the combination of different automated privacy policy creation approaches, like the one by Hjerppe et al. [7] with our DFD-approach. Combining different approaches could further reduce the overhead needed for the creation of privacy policies.

Besides the approach by Hjerppe et al., privacy property graphs (PPGs) by Kunz et al. [10] could also be used to extract privacy policy information. This could be achieved by adapting the methodology presented in this paper to take PPGs as input. This approach could potentially increase the amount of information extracted from the diagrams, as PPGs are enriched data-flow diagrams that contain additional information, which may be relevant for privacy policies.

Since DFDs are not part of UML[6] an extension of UML, standardizing DFDs, would be beneficial for future developments around DFDs. The UMLsec extension by Jürjens [8] is a good example for the benefits of a standardized notation.

Acknowledgement We thank Julien Lukasewycz for his useful input during the development of our approach, as well as writing this paper. We further thank the reviewers of this paper for their valuable input regarding the paper itself as well as the approach we presented.

References

1. Amazon Europe Core: Amazon.de privacy policy (2022), `https://www.amazon.de/gp/help/customer/display.html?nodeId=201909010&language=en_GB`, accessed 2023-07-02

[6] The Unified Modeling Language: `https://www.omg.org/spec/UML/`

2. Andow, B., Mahmud, S.Y., Whitaker, J., Enck, W., Reaves, B., Singh, K., Egelman, S.: Actions speak louder than words:Entity-Sensitive privacy policy and data flow analysis with PoliCheck. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 985–1002 (2020)
3. DeMarco, T.: Structure Analysis and System Specification, book section Chapter 9, pp. 255–288. Springer Berlin Heidelberg, Berlin, Heidelberg (1979). https://doi.org/10.1007/978-3-642-48354-7_9, https://doi.org/10.1007/978-3-642-48354-7_9
4. 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). https://doi.org/10.1007/s00766-010-0115-7
5. European Parliament, Council of the European Union: Regulation 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union **L119**, 1–88 (2016), http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC
6. Gerl, A.: Modelling of a privacy language and efficient policy-based de-identification. Thesis, Universität Passau (2020), https://nbn-resolving.org/urn:nbn:de:bvb:739-opus4-7674
7. Hjerppe, K., Ruohonen, J., Leppänen, V.: Extracting LPL privacy policy purposes from annotated web service source code. Software and Systems Modeling **22**(1), 331–349 (2023)
8. Jürjens, J.: UMLsec: Extending UML for secure systems development. In: UML 2002 - The Unified Modeling Language: Model Engineering, Concepts, and Tools 5th International Conference Dresden, Germany, September 30–October 4, 2002 Proceedings. pp. 412–425. Springer (2002)
9. Kohnfelder, L., Grag, P.: The threats to our products. Tech. rep., Microsoft Corporation (2009), https://nbn-resolving.org/urn:nbn:de:hbz:464-20210712-090625-4
10. Kunz, I., Weiss, K., Schneider, A., Banse, C.: Privacy Property Graph: Towards automated privacy threat modeling via static graph-based analysis. Proceedings on Privacy Enhancing Technologies **2**, 171–187 (2023)
11. Leicht, J., Heisel, M.: P2BAC: Privacy policy based access control using P-LPL. In: Mori, P., Lenzini, G., Furnell, S. (eds.) 9th International Conference on Information Systems Security and Privacy. pp. 686–697. SciTePress (2023). https://doi.org/10.5220/0011788500003405
12. Leicht, J., Heisel, M., Gerl, A.: PriPoCoG: Guiding policy authors to define GDPR-compliant privacy policies. In: Trust, Privacy and Security in Digital Business: 19th International Conference, TrustBus 2022, Vienna, Austria, August 24, 2022, Proceedings. pp. 1–16. Springer (2022)
13. Robles-González, A., Parra-Arnau, J., Forné, J.: A LINDDUN-based framework for privacy threat analysis on identification and authentication processes. Computers & Security **94**, 101755 (2020)
14. Wang, X., Qin, X., Hosseini, M.B., Slavin, R., Breaux, T.D., Niu, J.: Guileak: Tracing privacy policy claims on user input data for android applications. In: Proceedings of the 40th International Conference on Software Engineering. pp. 37–47 (2018)