A Pattern for Analysing Data Usage Requirements in Data Sharing

Accepted Manuscript

Julia Pampus[1][0000-0003-2309-6183] and Maritta Heisel[2][0000-0002-3275-2819]

[1] Fraunhofer Institute for Software and Systems Engineering ISST, Dortmund, Germany julia.pampus@isst.fraunhofer.de
[2] University of Duisburg-Essen, Duisburg, Germany
maritta.heisel@due.de

Abstract. A fundamental aspect of specifying data usage in the context of business-to-business data sharing is the definition of data usage conditions, technically represented as policies. Policies can be multifaceted and complex and are primarily used in a case dependent way. This often leads to challenges in the requirements engineering process, e.g., unclear requirements, complex representations, and/or inconsistent interpretations of policies. To address these problems, we introduce the *Data Usage Policy* pattern that defines the syntax of policies in data sharing. The pattern helps with the structured elicitation and analysis of data usage conditions and enables the creation of a consistent system design. We derive six pattern primitives and a policy grammar from scientific literature and existing requirements of four selected dataspace initiatives, that show a high level of technical maturity, and present them along with a three-step process for policy analysis. We demonstrate the applicability of our contributions with an application example.

Keywords: Policies · Data sharing · Requirements engineering · Data spaces.

1 Introduction

Data is of enormous value within businesses nowadays. However, it usually has to leave its source system to serve value creation [15]. To establish a trusted data infrastructure with shared governance and the technical capabilities for transferring and utilising data, dataspaces have emerged as a trend in various domains such as healthcare, automotive, and energy [17]. These initiatives aim for what is known as *data sovereignty*. Data sovereignty is defined as "the self-determination of [...] organizations with regard to the use of their data" [11, p.550]. As a key aspect of establishing data sovereignty, data sharing systems focus on defining data usage conditions in the form of policies, negotiating these, and enforcing them throughout data transfer and usage, employing mechanisms

for usage control and traceability. Thereby, policies represent the functional requirements that data sharing participants, acting as providers and/or consumers, pose in data usage. Within dataspaces, these requirements and their implementation are the key to a trusted environment, at the organisational and technical level. Independent from their technical representation, e.g., using the Open Data Rights Language (ODRL), policies are rules that define certain properties (usually constraints), e.g., for the geographical environment, the involved actors, time, and/or data handling. For example, a data sharing participant may state that data should only be used for a specific 'purpose' respectively context, such as research.

Current approaches elaborate on the requirements for trusted data sharing [2,8,18], the representation of data usage conditions as policies [9], and the technical approaches for enforcing these policies [16]. However, to the best of our knowledge, the research community so far has not addressed the analysis of policies during the requirements engineering process to ensure their syntactic accuracy, along with guided elicitation and conflict management. Therefore, we pose the following research questions (RQs).

- RQ1: How can policies in business-to-business data sharing be structured?
- RQ2: How can the correctness and consistency of policies be ensured in the requirements engineering process?

We follow a pattern-based approach to address these questions. Patterns facilitate identifying, analysing, and structuring requirements by capturing best practices and experiences from previous projects. Therefore, patterns are a common technique in requirements engineering, especially for the elicitation and analysis processes [12]. In this paper, we propose a composite pattern whose building blocks are grounded in know uses. Our work comprises two main contributions:

- We propose the *Data Usage Policy (DUP)* pattern, including a conceptual model for policies and a policy grammar.
- We define a three-step policy analysis (elicitation, decomposition, and conflict analysis) that specifies the usage and validation of the DUP pattern.

These contributions can be used by policy engineers or data stewards to enable a structured elicitation and analysis of data usage requirements and, thus, support the correct handling and implementation of requirements from diverse participants within business-to-business data sharing. By focusing on the context of business-to-business data sharing, we exclude end-user-centered processes and requirements.

The remainder of this paper is structured as follows: Section 2 introduces the background knowledge for this work. Next, in Section 3, we present our main contribution, the DUP pattern, whose implementation we describe and discuss in Sections 4 and 5. Afterwards, we compare our approach to related work in Section 6. Lastly, we answer the RQs and conclude our work in Section 7.

2 Fundamentals

This section describes the basic concepts of our pattern, including agendas in software development, some concrete approaches to managing conflicting requirements, and a common architecture for enforcing policies.

2.1 Agendas in Software Development

Agendas are patterns for systematically structuring a procedure for developing a software artefact [7], including the phase of requirements engineering. An agenda consists of multiple steps that are performed sequentially and have one or multiple *input* and *output* values. To make its application effective in practice, the correct implementation of the procedure is validated by verifying the output values with regard to defined conditions, namely *validation conditions*. These validation conditions can be defined for each step or for the overall process. For example, a step 'elicite requirements' could have unstructured requirements (text) as input and structured requirements (template) as output. A validation condition may define mandatory attributes of the template, ensuring that next steps that further use the created template can be executed properly. We make use of agendas to define the process of analysing policies while making use of our pattern.

2.2 Managing Conflicting Requirements

Conflict management is an important aspect of requirements engineering that enhances the development of software systems [1]. It involves identifying and resolving conflicting interests to maximise stakeholder satisfaction on all sides. Although approaches to conflict management can vary based on the context, most are grounded in a mathematical framework. This framework typically includes a selection of aspects, also referred to as decision variables [23], along with their respective weightings. For example, Mitchell et al. [13] suggest that stakeholders can be prioritised based on their power, legitimacy, and urgency. By evaluating these attributes, we can assign priority values to each factor in the conflict resolution process. Conflicts are then systematically resolved by considering both the weights of the stakeholders involved and the number of stakeholders affected [1]. We adopt this approach in our work.

2.3 XACML Architecture

In our work, we refer to the architecture of the Extensible Access Control Markup Language (XACML) for describing the usage of our policy pattern. XACML [22] is a standard for defining access control policies in a machine-readable format. Its architecture structures policy enforcement across various systems and applications, enabling centralised access control management. Thereby, it defines four conceptual components that are required for policy enforcement, depicted in Figure 1: The *policy administration point (PAP)* includes policy creation, editing,

and storage tools. In the context of trust-based data sharing, this means both the provisioning of data offers and the negotiation of policy agreements. The *policy enforcement point (PEP)* intercepts data flows and enforces the policy validation by the policy decision point (PDP) to process resulting decisions. The *PDP* evaluates policies by validating agreements against context information retrieved by the *policy information point (PIP)* and returns decisions (permit or deny) to the PEP. The PIP provides context information required for policy evaluation by retrieving attributes from external systems, e.g., about actors or environmental conditions.

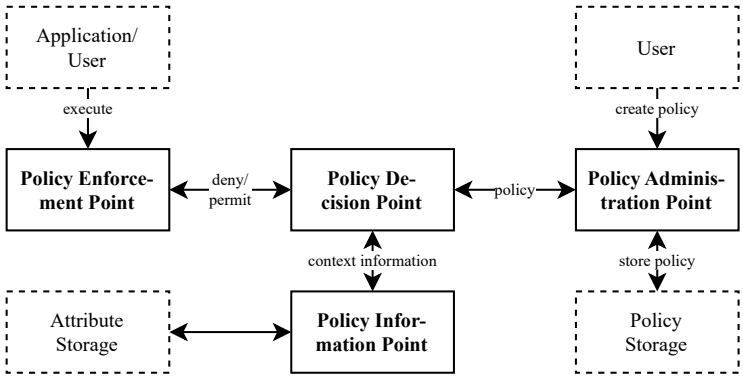

Fig. 1. XACML Architecture [22]. This model illustrates the interactions between four policy systems and to other interfaces (storage, application, user).

3 Data Usage Policy Pattern

The main contribution of this paper is a pattern that helps in decomposing data usage requirements. This pattern can be used for a template-based elicitation of requirements and their analysis, including conflict resolution (cf. Section 4). Our approach contributes to a systematic requirements engineering process and the subsequent technical design of data sharing systems that aim to establish the sovereignty of data sharing participants while putting the focus on policy definitions and implementations.

We structure the presentation of our pattern according to the guidelines provided by the EuroPLoP community[3], elaborating on the context in Section 3.1, problem and forces in Section 3.2, solution in Section 3.3, and consequences in Section 5.

[3] `https://www.europlop.net/patterns/` (accessed on 2025-01-05)

3.1 Context

An organisation has decided to share data with others. The process of data sharing involves at least two actors, one providing the data and one consuming it. The data on the provider-side is enriched with metadata including policies, building a dataset. These policies describe how data should be used, independent from their technical representation. Multiple policies can be added to the same dataset. The systems that are used to share and process the data must be able to enforce these policies.

To apply the pattern, we rely on the elicitation of data usage requirements. Data usage requirements have been elicited in natural language, either as plain text or following a template such as our constraint template [19].

3.2 Problem & Forces

> *How can semantic interoperability of data usage requirements be ensured in decentralised environments? How can validation and conflict resolution be supported throughout the requirements engineering process?*

Although policies are central to trust-based business-to-business data sharing, they are often treated superficially in existing use cases. In both literature and practice, very little attention is paid to the structured analysis of policies during requirements engineering, particularly in the design phase of data sharing systems. Current technical solutions primarily focus on simple data usage permissions and prohibitions.

Referring to our example from Section 1, the purpose-restricted policy is simplified to the validation of an organisation on data access in real-world dataspaces like Catena-X[4]. Catena-X is a collaborative network in the automotive industry designed to create a standardised data ecosystem for enhanced data sharing and interoperability across the value chain. By restricting the access to a verified contractual partner, it is demanded that data processing takes place according to the defined purpose. When implementing policy enforcement, thus, selective access control often replaces the validation of the policy during the actual data usage.

- *Compliance*: Policies typically arise from organisational conditions or legal regulations, which are often expressed in human-readable text. These formulations follow standards and can be verified for legal accuracy and completeness. However, there is no clear guidance on which parts of these conditions and regulations can be mapped to technical policies and how they can be verified for completeness and quality. For example, a 'purpose' is difficult to describe or verify technically. However, it can be narrowed down to the use of a specific algorithm in data processing or a specific application. These conditions can be verified technically.

[4] `https://catena-x.net/` (accessed on 2025-02-01)

- *Know-how*: The inadequate analysis of policies partly originates from a lack of understanding of a holistic end-to-end data sovereignty, presumably because no domain-specific software engineering methods are available. Currently existing dataspace solutions consider usage control and, thus, policy enforcement to be a nice-to-have or even a costly liability. The technical representation to express policies in dataspaces is ODRL [10]. While ODRL allows for a very precise technical expression of how data can be used, it is also complex and requires a high level of technical understanding. Therefore, the initial step of formulating data usage conditions as ODRL policies by individuals lacking ontology expertise is unrealistic. Returning to our example, it is important to clarify all aspects of a policy in order to be able to break down a 'purpose' technically.
- *Conflict identification & resolution*: In the existing implementation scenarios of policies, neither conflict identification nor resolution occurs. The requirements formulated by the data provider or the dataspace initiative (cf. Section 3.1) are taken for granted. This approach corresponds to a consent technique: the provider specifies the policy, and the consumer consents to it. Consequently, every data sharing system that wants to participate in such a data exchange must implement the entire set of policies that has been specified without further negotiation or analysis. Possible conflicts and their resolutions are not further considered. In the case of purpose-limited data usage, conflict resolution becomes increasingly difficult, as the interpretation of a 'purpose' is subjective. A structured comparison of two interpretations of a purpose can only be achieved through technical specifications.
- *Policy enforcement*: Different data sharing systems do not necessarily address the consistent enforcement of policies in decentralised environments [8]. A common assumption is that the policies attached to a shared dataset are defined by the data provider and, thus, must be implemented primarily in the consumer system. However, both data sharing participants can formulate data usage conditions at design time and runtime [18]. Therefore, it must be possible to analyse single components of a policy and design an appropriate system architecture. For example, the implementation of a purpose-restricted data usage policy may require a pre-processing in the provider's system.

To address the challenges of handling data usage requirements in business-to-business data sharing, we propose a structured policy pattern.

3.3 Solution

Structure policies independent from their technical representation, according to five core elements: data, action, actor, environment, and time. Make use of a grammar that helps to technically validate data usage requirements.

By ensuring consistency and accuracy in defining data usage requirements, this approach enhances compliance and aids in conflict resolution throughout the

requirements engineering process, leading to improved collaboration and trust in data sharing.

This section is structured as follows: In Section 3.3, we present a conceptual model for policies that serves as a basis for our pattern. First, we outline the pattern structure, then, we describe its elements.

Policy Model Figure 2 depicts a conceptual model that visualises the main policy components addressed in trust-based data sharing: the data itself (e.g., raw, anonymised), allowed or enforced actions (e.g., processing, deletion, distribution), and constraints of different types: the involved actors, their environment (e.g., organisation, system, location), or temporal conditions (e.g., time interval, date).

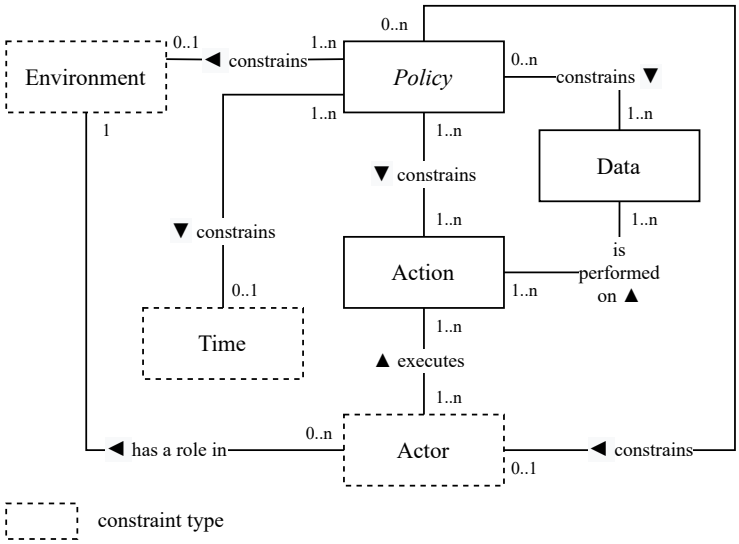

Fig. 2. Conceptual Model for Policies. It depicts five entities and their relations affecting a data usage policy.

Therefore, this model includes the entities data, action, time, actor, and environment related to the overall type *policy*. According to the ODRL [10], as one of the widely used policy representations in industrial data sharing, a policy describes the permissions, prohibitions, and duties of data usage.

– *Data* serves, next to the Policy, as the central element in this model. It is modified by Actions and can be constrained by Policies. In terms of representation, Data can exist in different forms, such as a single file or a data stream; it can also describe access to a system. Data can also be metadata for other data.

- An *Action* describes one or more operations that can be executed on a single or many Data. It can be of type *use*, but can also specify more detailed usages or duties. Examples can be found in the ODRL vocabulary[5], such as 'use', 'copy', or 'delete'. An Action is constrained by one to many Policies.
- An *Actor* is a human person or a system that executes an Action performed on Data. It is part of one or many Environments and can have one or multiple roles in these Environments. An Actor may be constrained by a Rule that limits possible Actions to their assigned attributes.
- An *Environment* refers to the organisational or system landscape in which Data is transferred and/or processed. It may delineate the roles that Actors can assume within that Environment or define some geographical boundaries.
- *Time* describes the temporal constraints of an Action. It can define a time interval, a point in time, or a frequent event.

In summary, a policy constrains data and actions that are performed on data. If the data usage should be restricted to a specific purpose, the policy may have zero to three constraints (environment, actor, and/or time).

Pattern Structure Our DUP pattern builds upon the conceptual model for policies presented in Figure 2. It comprises the definition of six pattern primitives (permission, prohibition, duty, actor, environment, time), classified into two types (action and constraint), and a context-free grammar to specify the policy syntax.

Pattern Primitives Each policy consists of two main pattern primitives, an *action* and a *constraint*, each with three sub-types, as depicted in Figure 3. In summary, we define six recurring pattern primitives of two types. The pattern primitives of type *action* focus on the specification of a policy's action, covering permission, prohibition, or duty (cf. Section 3.3). The pattern primitives of type *constraint* describe a particular constraint type: actor, environment, and time (cf. Figure 2). Figure 3 presents the hierarchy of the pattern primitives, leading to a policy structure pattern.

Policy Grammar We present expressions of the DUP pattern's structure as productions of a context-free grammar, using an extended Backus-Naur form as notation, with start symbol P for Policy. Each instance of the pattern defines one mandatory pattern primitive of type action A and an optional pattern primitive of type constraint C, as defined in Production 1.

$$P ::= A\,[C] \tag{1}$$

$$C ::= C_A \,|\, C_E \,|\, C_T \tag{2}$$

$$A ::= data\ action\_type \tag{3}$$

[5] `https://www.w3.org/TR/odrl-vocab/#action` (accessed on 2025-01-05)

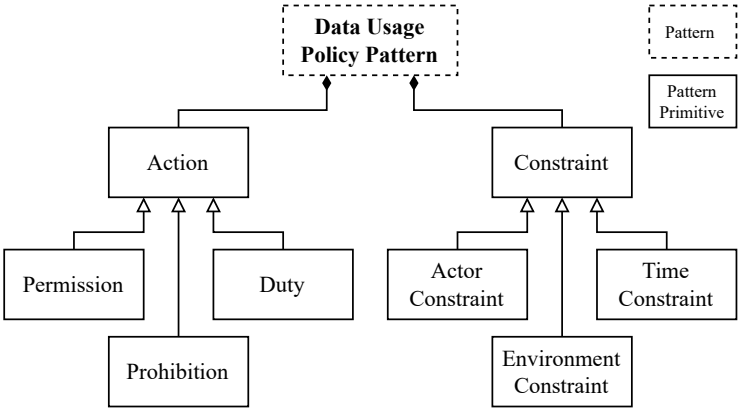

Fig. 3. Structure of the *Data Sharing Policy* Pattern. It shows the composition of the pattern consisting of two types of pattern primitives, an action and a constraint, as well as their sub-types.

$$data ::= \text{String} \tag{4}$$

$$action\_type ::= permission \,|\, prohibition \,|\, duty \tag{5}$$

There, the pattern primitive of type action A and the pattern primitive of type constraint C are non-terminal symbols that are detailed with further productions. A pattern primitive of type constraint C consists of one of the non-terminal types C_A (actor constraint), C_E (environment constraint), or C_T (time constraint) (see Production 2). A pattern primitive of type action A consists of a *data* followed by an *action_type* (see Production 3), where the *data* is described by a String (see Production 4), and an *action_type* consists of one of the non-terminal types *permission*, *prohibition*, or *duty* (see Production 5). Their productions are further refined in the following pattern descriptions.

Action Types All instances of our DUP pattern can be transferred into a system design using the XACML architecture. However, the interactions between the system components and their temporal sequence differ depending on the policy. We use a unified sequence diagram to visualise the data access respecting validating all policy types at runtime in Figure 4. In this diagram, we omit the policy and attribute storage from Figure 1 for simplicity, as they do not contribute to the overall comprehension. To showcase the application of a policy, we assume that it has been created or negotiated beforehand. Thus, in our context, the *policy* represents a policy agreement already reached. The objects and actors in the sequence diagram are used for illustration purposes and do not imply any conclusions about the system architecture.

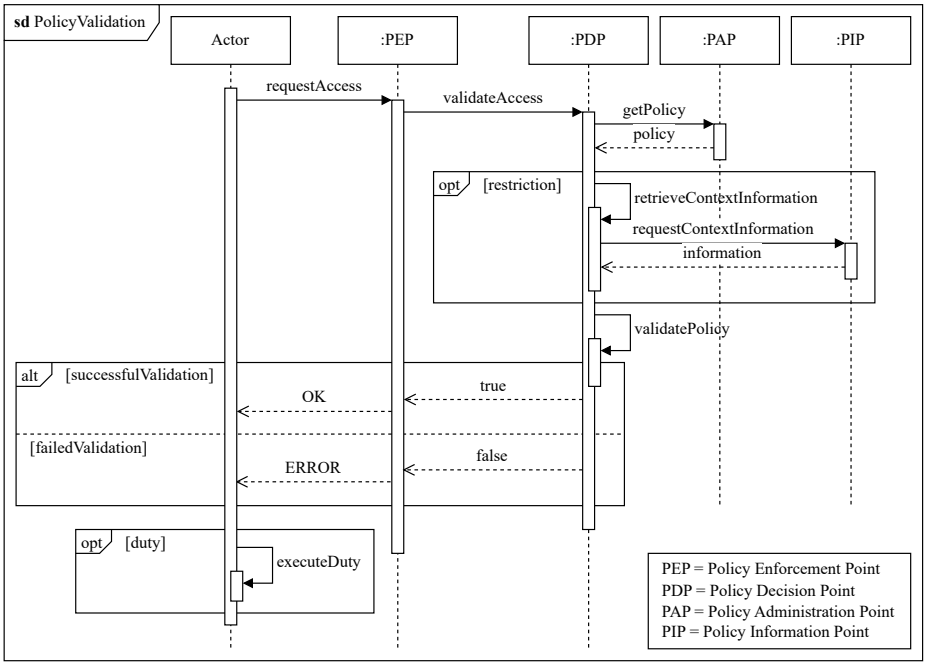

Fig. 4. Simplified Sequence Diagram for Data Access. It illustrates the interactions between the policy systems of the XACML architecture during policy validation. It shows different scenarios, a successful validation and a failed validation. It also provides for the optional retrieval of context information and the optional execution of duties.

Permission Data may be used in a defined manner, including a specific (mandatory) action of type *permission*. Constraints of actors, environments, or time are optional (see Production 1). As depicted in Figure 4, the number of the system interactions is limited since, depending on the request, an 'okay' is returned, and the data can be used. Considering data usage conditions that are applied in practice (cf. Table 5), policies without constraints are rare. For example, a policy could define data as being used by everyone in every context (R20) or allow general data distribution by stating, 'Data can be distributed'. For applying the policy grammar, the list of allowed *permissions* in Production 6 would need to be defined, e.g., as the set of ODRL Actions [10] or by using another list of actions.

$$permission ::= ODRL\_ACTIONS \tag{6}$$

Prohibition A particular action must not be executed; this includes specific actions or concrete operations. The system prohibits the request for data access for the black-listed actions. Taking a look at Figure 4, this means that, e.g., if data must not be distributed (R07, Table 5), the access would be denied

if the request was already indicating a policy violation according to provided attributes. In contrast to a permission policy, a prohibition is challenging to implement. The system interactions are likely to become more complex. For defining prohibited data usage, the policy action must be of type *prohibition* and the production must be further defined (see Production 7).

$$prohibition ::= ODRL\_ACTIONS \tag{7}$$

Duty Executing an action is required, necessarily associated with a constraint referring to the data usage, e.g., before or after using the data (R12, Table 5), the actor, or the environment. Therefore, when defining an action of type *duty*, a constraint must be defined, too, see Production 8. As shown in Figure 4, after data access has been granted, a duty must be performed before, during, or after data usage (as stated in the constraint). For example, a policy could define that data usage must be logged (R06) or data must be anonymised before usage (R01). Along with permissions and prohibitions, the production starting with *duty* in Production 8 must be completed.

$$duty ::= ODRL\_ACTIONS \tag{8}$$

Constraint Types Our DUP pattern includes three constraint types: actor, environment, and time.

Actor The actor who uses the data is restricted based on properties, e.g., identity or role. As specified in Productions 9 to 11, they must be of type *system* (R03, Table 5) or *human* (R14) both described as a String. As shown in Figure 4, the PIP validates the attributes included in the initial request to take a final decision.

$$C_A ::= human \,|\, system \tag{9}$$

$$human ::= \text{String} \tag{10}$$

$$system ::= \text{String} \tag{11}$$

Environment The environment of type *system* or *organisation* consisting of a String (see Productions 12 to 14) in which the data is used is restricted based on properties, e.g., the identity (R23, Table 5) or security attributes (R16). In Figure 4, the same simplified flow applies, as to every other constraint.

$$C_E ::= organisation \,|\, system \tag{12}$$

$$organisation ::= \text{String} \tag{13}$$

$$system ::= \text{String} \tag{14}$$

Time This pattern primitive describes a time frame for data usage or a temporal trigger for some required action. For example, a policy could state that data must be deleted at a defined datetime (R13, Table 5) or that data must be deleted periodically (R11). Overall, the time specification must be of type *one-time* or *repeating* described as a String (see Productions 15 to 17).

$$C_T ::= \textit{one-time} \,|\, \textit{repeating} \tag{15}$$

$$\textit{one-time} ::= \text{String} \tag{16}$$

$$\textit{repeating} ::= \text{String} \tag{17}$$

In practice, a policy can either focus on one aspect of data usage or combine multiple of them at once (cf. Section 3.1). In this work, when describing a policy, we refer to it as a single constraint (also ODRL Rule) and ignore possible nestings. Therefore, as defined in Production 2, the presented grammar does not allow for a combination of multiple constraints within one policy. We assume that multiple policies (with different constraints) can be added to one dataset.

4 Implementation

The DUP pattern provides a solid foundation for a structured policy analysis process that we present in the following, complemented by an application example.

4.1 Policy Analysis Process

For a structured policy analysis process, we define an agenda (cf. Section 2.1) divided into three steps, as shown in Figure 5: first, the policies must be elicited; next, policy patterns are decomposed; and finally, occurring conflicts are analysed and resolved. We conclude each section with a validation condition VC to verify whether the policy analysis has been conducted correctly and the defined policy grammar has been respected.

Policy Elicitation From the policy model in Figure 2, we derive a policy template shown in Table 1. In addition to the *component*, *type*, and *value*, we provide some guiding questions. The component types reflect the terms from the policy grammar. As it is not relevant for the further processing of the pattern, the difference between data and metadata is not addressed in the grammar.

The template is instantiated in the first step of the policy analysis process in Figure 5. Each step of this process further refines the policies according to the instantiated templates, as they continue to be filled and extended. We demonstrate this procedure in Section 4.2. If no requirements have been collected in advance, this template can also be used to identify initial requirements.

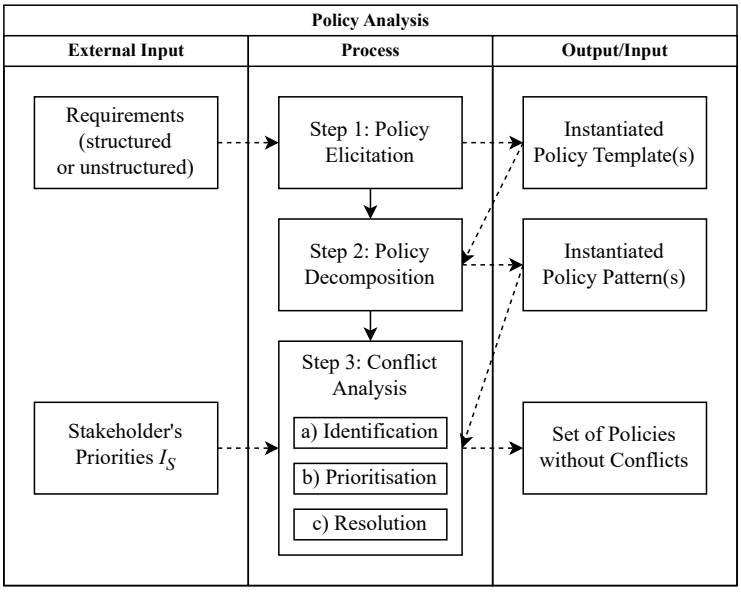

Fig. 5. Policy Analysis Process. This process includes three steps that process stakeholder input and produce structured requirements.

VC01 For each policy, exactly one policy template has been instantiated.

Policy Decomposition Next, based on the created policy templates, the DUP pattern is instantiated. This instantiation uses the policy grammar from Section 3.3. Following Production 1, every policy must contain one data and one action component, conforming to exactly one pattern primitive of type action. In addition, each policy can comply with an additional pattern primitive of type action constraint. The results of this step is an extended policy template. If a policy does not comply with the rules of the grammar, it is considered to be either incomplete or incorrect. Thus, it must be revised in collaboration with the owning stakeholder, returning to Step 1.

VC02 Each policy must be an instance of the DUP pattern.

Conflict Analysis With the help of the instantiated templates and policies, it is possible to easily compare individual policies to identify and resolve occurring conflicts. The conflict analysis consists of three main aspects: a) identification, b) prioritisation, and c) resolution (cf. Figure 5). As multiple policies may target the same dataset, we assume that the providing and consuming data sharing participants ensure that the policies they have defined themselves do not contain any conflicts. Therefore, conflicts may only arise between the policies of the data

Table 1. Policy Template.

Component	Type	Value
data	*What data is the target of the policy?* ☐ data ☐ metadata	[...]
action	*What action is described?* ☐ permission ☐ prohibition ☐ duty	[...]
actor	*Are there any actor constraints?* ☐ human ☐ system	[...]
environment	*Are there any environment constraints?* ☐ organisation ☐ system	[...]
time	*Are there any time constraints?* ☐ one-time ☐ repeating	[...]

provider and those of the data consumer. In general, these conflicts are most likely of type modality, occurring when two policies target the same data with the same action but with contradicting permissions and prohibitions [3].

Identification We identify two specific indicators of a conflict, one based on the action type and the other on the constraint type. An *action type conflict* arises if multiple policies refer to the same *data* and *action* but with a different type of action. For example, the provider defines that data must not be distributed (R07, Table 5), and the consumer defines that they want to share the data with a third-party (R10). A *constraint type conflict* occurs if multiple policies refer to the same *data* and (type of) *action* but with different restrictions of the same type. For example, a data provider requires the data to be deleted after three months, and the data consumer foresees a deletion after six months.

Prioritisation Once a conflict has been identified, it should be resolved. In doing so, multiple factors must be considered, e.g., the involved stakeholders and the priority of their requirements (policies). We define the following sub-process for approaching a successful conflict resolution (cf. Figure 6).

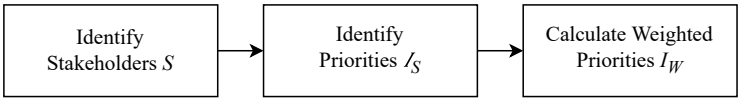

Fig. 6. Process of Resolving a Policy Conflict. This process includes three steps that need to be performed for resolving conflicting requirements.

1. *Identify stakeholders*: First, it is documented who the owner of each policy is. We assume a simple, bilateral data sharing scenario involving a provider and a consumer. The provider expresses the conditions under which they are willing to share their data, and the consumer defines what they want to do

with it. Since some of the policies may arise from regulations, we refer to the latter as a third stakeholder.
Each stakeholder has a weighting W_S: As the data source, the data provider may ultimately decide whether the data is shared. Therefore, the provider, with factor *2*, is weighted stronger than the consumer with factor *1*. Since regulations have an even higher impact, they are weighted with factor *3*.

2. *Identify priorities*: Each policy must be assigned a priority I_S. Amroune et al. [1] use a scale from 1 to 5 in their approach (cf. Section 2.2); we simplify this to 1 (low priority), 2 (medium priority), and 3 (high priority).
3. *Calculate weighted priorities*: The weighting of each policy is calculated based on the assigned stakeholders S, their weightings W_S, and the defined priorities I_S. The calculation is a simple multiplication of priorities and stakeholder weightings: $I_W = I_S * W_S$.

Resolution Finally, a conflict resolution strategy must be chosen. Essentially, policies with a higher weighting are preferred over those with a lower weighting, which leads to one of the following resolution strategies. We assume that the provider's policies form the basis and that these are adapted according to the chosen strategy, or that all resulting (non-conflicting) policies are part of a data offering.

- *Extended scope*: Data can be used for more than originally intended (by the data provider). For example, data can be used for multiple purposes or a greater duration.
- *Reduced scope*: Data can be used for less than originally intended (by the data provider). For example, the consumer introduces a deletion strategy that the provider did not include in the first place.
- *No changes*: Data sharing adheres to the data provider's policy. For example, the consumer wants another price for a dataset, but the provider does not provide any options to negotiate.
- *No sharing*: Data will not be shared under the stated conditions. This scenario could happen if a provider defines a policy the consumer cannot or does not want to implement.

If policies provide have priority, both stakeholders manually negotiate a resolution strategy. In addition, assuming that data usage conditions can always be negotiated in a negotiation process (either at design time or at runtime), we consider the calculated weightings as a tendency towards the best possible conflict resolution. The provider can withdraw their data offering at any time without explanation.

VC03 A group of policies targeting the same dataset do not show any conflict.

4.2 Application Example

We evaluate our DUP pattern using an application example that represents a simple and typical data sharing scenario, adopted from Pampus and Heisel [19]:

A delivery service aims to optimise delivery routes and combine shipments from various suppliers to supermarkets by utilising the suppliers' delivery information to calculate the carbon footprint. The delivery service serves as the data consumer and the supplier as the data provider.

The application example introduces four policies that we examine in the following.

- Data can be used to calculate the carbon footprint (P01).
- Data must be deleted after six months (P02).
- Data may not be passed on to other suppliers (P03).
- Every data transaction must be recorded and preserved for three years (P04).

We use the policies to demonstrate how the DUP pattern can be used for requirements elicitation and analysis including conflict resolution.

Policy Elicitation As part of the requirements elicitation, i.e., as a first step of the requirements analysis (cf. Section 4.1), the policies are identified in a structured way by instantiating the template from Table 1. As the specified pattern does not provide for a combination of policies, we split policy P04 into two separate policies: one describes the actual logging activity (P04a) and the other states when the logs should be deleted (P04b). For demonstration purposes, Table 2 provides an instantiated policy template for P04a. The instance of the DUP pattern is described in Equation 18.

$$\begin{aligned} P &\rightarrow A\,C \\ A &\rightarrow \mathit{data}\ \mathit{action\_type} \\ \mathit{data} &\rightarrow \text{“data transaction”} \\ \mathit{action\_type} &\rightarrow \mathit{duty} \\ \mathit{duty} &\rightarrow \text{“record”} \\ C &\rightarrow C_T \\ C_T &\rightarrow \mathit{repeating} \\ \mathit{repeating} &\rightarrow \text{“on data usage”} \end{aligned} \tag{18}$$

Pattern Decomposition Next, we can derive the policy patterns (one or many) according to the grammar. Here, the table format of the policy template helps with identifying the types of policy components.

- Policy P01 describes a 'purpose' that can be technically specified as a processing action *calculate* bound to an application for assessing the carbon footprint. The application represents the actor of type *system*. The result is an instance of the DUP pattern with a permission pattern primitive and an actor pattern primitive.

Table 2. Instantiated Policy Template (cf. Table 1) for P04a.

Component	Type	Value
data	☐ data ■ metadata	data transaction
action	☐ permission ☐ prohibition ■ duty	record
actor	☐ human ☐ system	–
environment	☐ organisation ☐ system	–
time	☐ one-time ■ repeating	on data usage

- Policy P02 describes an action that must be executed after a given time frame. This time restriction focuses on an event that happens once. This results in an instance of the DUP pattern with a duty pattern primitive and an time pattern primitive.
- Policy P03 describes a restriction of actors allowed to receive the data as a third-party. The instantiated pattern combines a prohibition and an environment restriction as the organisation represents the environment.
- Policies P04a and P04b describe an action that must be executed periodically. The instantiated patterns for both policies contain a duty pattern primitive and a time pattern primitive.

Following the productions of the policy grammar, in this step, we can verify the completeness and correctness of the elicited policies according to VC01 to VC03 (see Section 4.1). For example, if P02 missed a restriction, it would not be a valid policy. The results of this analysis step are listed in Table 3. Due to limited space, we have shortened the table to exclude the type column as used in Table 2. The pattern instances are listed in Table 4.

Conflict Analysis In the following, we introduce policy conflicts by adding requirements of the data consumer (delivery service) to the set of policies.

- The data should be used for research purposes (P06).
- The data must be deleted after three months (P07).
- The data must be distributed to other suppliers (P08).
- The logs should be deleted after one year (P09).

The new policies are listed along with P01 to P04b in Table 3. Following the conflict analysis process (cf. Figure 5), first, we identify the conflicts according to the indicators described in Section 4.1. Using different colours, we highlight the conflicting actions and restrictions in Table 3. Next, to prepare a selection of resolution strategies, we need to calculate the weighted priority of each policy. Thus,

we identify the stakeholders S and assign their weightings W_S, add individual priorities I_S, and, finally, calculate the weighted priorities I_W of all policies. For demonstration purposes, we assume priorities that best serve the presentation of the different resolution strategies in the following. Table 4 summarises all input and output values of this process.

Table 3. Comparison of Instantiated Templates for P01–P09.

Comp.	P01	P02	P03	P04b
Pattern	Actor-restricted Permission	Time-restricted Duty	Environment-restricted Prohibition	Time-restricted Duty
data	"delivery"	"delivery"	"delivery"	"records"
action	calculate	delete	distribute	delete
actor	CO_2 footpr.	–	–	–
env.	–	–	supplier	–
time	–	after 6m	–	after 3y

Comp.	P06	P07	P08	P09
Pattern	Actor-restricted Permission	Time-restricted Duty	Environment-restricted Permission	Time-restricted Duty
data	"delivery"	"delivery"	"delivery"	"records"
action	calculate	delete	distribute	delete
actor	research	–	–	–
env.	–	–	supplier	–
time	–	after 3m	–	after 3y

Conflict 1 P01 and P06 describe using data for calculations, resulting in an actor-restricted permission. The supplier allows usage for calculating the carbon footprint, and the delivery service wants to use the data for research purposes. By assigning a low priority, we assume the data provider is open for other purposes. Thus, the weighted priority of P06 is higher than that of P01. As a result, the chosen resolution strategy is to extend the scope of the provider's policy applied to the transferred dataset.

Conflict 2 P02 and P07 describe data deletion after a given timeframe. In this example, the delivery service's policy narrows the scope of the provider's policy. As the importance of P07 is weighted higher than that of P02, choosing the strategy *narrow scope*, a modified policy is applied to the transferred dataset.

Conflict 3 P03 and P08 address the same action and environment restriction but with contradicting action types, prohibition and permission. As the supplier assigns a higher priority and, as a data provider, is weighted higher than the

Table 4. Instantiated Patterns and Weighted Priorities I_W for P01 to P09.

ID	Instantiation of Pattern P	Stakeholder S	Weighting W_S	Priority I_S	I_W
P01	$P \rightarrow$ "delivery" "calculate" "CO_2 footprint"	Provider	2	1	2
P02	$P \rightarrow$ "delivery" "delete" "after 6 months"	Provider	2	1	2
P03	$P \rightarrow$ "delivery" "distribute" "supplier"	Provider	2	3	6
P04a	$P \rightarrow$ "transaction" "record" "data usage"	Regulation	3	3	9
P04b	$P \rightarrow$ "records" "delete" "after 3 years"	Regulation	3	3	9
P06	$P \rightarrow$ "delivery" "calculate" "research"	Consumer	1	3	3
P07	$P \rightarrow$ "delivery" "delete" "after 3 months"	Consumer	1	3	3
P08	$P \rightarrow$ "delivery" "distribute" "supplier"	Consumer	1	3	3
P09	$P \rightarrow$ "records" "delete" "after 3 years"	Consumer	1	3	3

consumer, the consumer must either accept the policy (no changes) or decide against data consumption (no sharing).

Conflict 4 P04b and P09 describe deleting data transaction records after three and one year. As P04b originates in regulations, it is always weighted higher than any other policy. The possible resolution strategies correspond to those of the third conflict.

4.3 Known Uses

To understand what requirements arise in practice and showcase known uses of the DUP pattern, we have conducted desk research, examining currently existing dataspace initiatives, including nationally and EU-funded projects and industrial products. For this purpose, we selected the few dataspaces that are already operational, according to our information, such as Catena-X, Mobility Data Space (MDS)[6], and Energy Data Space (EDS)[7], and approaches that provide a high level of maturity when it comes to the implementation of policy management (Prometheus-X[8], Boot-X[9]). We complemented our research with scientific publications representing the automotive, pharmaceutical, and food industries. Table 5 lists 27 data usage requirements extracted from publicly available reports and standards of these projects, representing policies of different action and constraint types. We highlight the *action* of each policy in italics. For example, policies R14 to R27 describe the allowed usage of data by restricting the duration of usage (time constraint, cf. R25), the organisations that may access the data (envrionment constraint, cf. R21), or the actors that must meet defined criteria (actor constraint, cf. R16). We can observe that prohibitions or duties are not commonly used.

[6] `https://mobility-dataspace.eu/` (accessed on 2025-02-01)
[7] `https://energydataspaces.eu/` (accessed on 2025-02-01)
[8] `https://prometheus-x.org/` (accessed on 2025-02-01)
[9] `https://www.boot-x.eu/` (accessed on 2025-02-01)

Table 5. Data Usage Requirements R from Selected Dataspace Initiatives.

ID	Description	Action Type	Constraint Type	Ref.
R01	Data must be *anonymised* before usage.	duty	time	[4]
R02	Data must not be *used* for a specific operation.	prohibition	actor	[20]
R03	Data can be *used* for a specific operation.	permission	actor	[20]
R04	Two data copies must be *used* at the same time.	permission	time	[20]
R05	Data must not be stored without being *aggregated*.	prohibition	time	[14]
R06	Data processing activities must be *logged*.	duty	time	Prometheus-X
R07	Raw data must not be *distributed*.	prohibition	*none*	[4]
R08	Data usage must be *logged* at an external service.	duty	actor	Mobility Data Space (MDS)
R09	Data usage must be *reported* to a defined actor (user/system).	duty	actor	Boot-X
R10	Data can be *shared* with third-parties that comply with GDPR.	permission	environment	Prometheus-X
R11	Data must be *deleted* periodically.	duty	time	[4]
R12	Data must be *deleted* after usage.	duty	time	[20]
R13	Data must be *deleted* at a defined datetime.	duty	time	Boot-X
R14	Data must be *used* by specified user roles.	permission	actor	Boot-X
R15	Data must be *used* by defined dataspace connectors.	permission	actor	MDS, Boot-X
R16	Data must be *used* in environments of a defined security level.	permission	actor	MDS
R17	Data must be *used* by defined dataspace applications.	permission	actor	Boot-X
R18	Data must be *processed* in its origin country.	permission	environment	[20]
R19	Data must be *used* at a defined geolocation.	permission	environment	Boot-X
R20	Data is publicly *available*.	permission	*none*	Energy Data Space (EDS)
R21	Data must be *used* by members of the dataspace.	permission	environment	Catena-X
R22	Data must be *used* by defined organisations.	permission	environment	Catena-X, EDS, Boot-X
R23	Data offers must be *accessible* to defined organisations.	permission	environment	EDS
R24	Data must be *used* by organisations of/at a specific type/location.	permission	environment	EDS
R25	Data must be *used* within a defined time interval.	permission	time	MDS, Boot-X
R26	Data must not be *used* more than a defined amount of times.	permission	time	MDS, Boot-X
R27	Data usage must *comply* with a defined purpose.	permission	*any*	Catena-X

Two special types of policies are represented in R20 and R27. If data is freely available, a policy does not contain a constraint type, but an action type *permission*. This obligation is reflected in our policy grammar. Referring to our example from Section 1, a purpose can be technically described by any constraints (also multiple ones). Concrete constraint types are context-dependent and must therefore be specified during the requirements engineering process.

5 Consequences

The DUP pattern comes with some benefits and liabilities, which we discuss in the following.

5.1 Benefits

The proposed pattern facilitates a structured elicitation and analysis of policies while addressing the forces from Section 3.2.

- *Compliance*: With the help of the DUP pattern, policies can be validated for syntactic completeness and correctness. This validation includes, on the one hand, the correct specification and combination of policy components and, on the other hand, the identification of missing ones. This forms the basis for ensuring technical compliance with organisational guidelines and regulations.
- *Know-how*: Due to the limited policy grammar guided by a simple template, even non-experts can easily define valid policies without the need to deal with technical representations like ODRL. This increases technical correctness and the trust of the stakeholders in each other, as well as the software systems that implement the policies. Creating a common understanding of the represented policies leads to clear accountability and fewer divergences in syntactic interpretations.
- *Conflict resolution*: The presented conflict analysis process allows for resolving conflicting policies at design time. This approach aims for designing an aligned and consistent system architecture for data sharing use cases, e.g., using the XACML architecture. Although we assume that stakeholders themselves take care of not causing any policy conflicts (cf. Section 4.1), the conflict resolution process can also be applied to validate their consistency, not necessarily between multiple stakeholders.
- *Policy enforcement*: Originating in selected domains (cf. Section 3.1), the DUP pattern may be applied to other contexts. The policy model also allows for a more fine-grained analysis of data usage conditions, e.g., by adding new (sub-) components and pattern primitives or further detailing the grammar's productions. Most important, the pattern and its application during the requirements engineering process allows for involving both data providers and consumers.

5.2 Liabilities

Our approach only considers simple, independent policies. Complex policy constructs are generally taken into account but may require the extension of the presented policy grammar. Such an extension may entail the introduction of new validation conditions for the policy analysis process. Furthermore, although we use the pattern to form the basis for an aligned technical representation of policies, semantic differences in interpretation, e.g., *use* in comparison to specific sub-actions like *read*, are not entirely excluded during application.

6 Related Work

In the context of dataspaces, Hosseinzadeh et al. [9] proposed a set of 14 policies, called *policy classes*, gathered from a set of selected use cases. These include requirements like 'Restrict the data usage for a group of users or systems' [9, p.399], 'Delete data [. . .] after (one) use' [9, p.399], or 'Notify a party when data is used' [9, p.399]. Following this approach, every policy that emerges in a use case can lead to a new policy class and be reused in another context, along with a specific policy evaluation or enforcement function. Later, the introduced classes were extended by seven additional policies and resulted in a set of 21 policies [5]. Comparing the DUP pattern and its pattern primitives to these, we provide an approach of a higher abstraction level. This approach increases generalisability and, thus, reusability. In addition, we have substantiated our work with a more recent case analysis (cf. Table 5).

Chadha [3] created a taxonomy of policy conflicts, dividing them into application-independent and application-specific conflicts. Depending on their types, the conflicts are handled differently at compile-time or runtime. Our conflicts are comparable to the type modality conflict (cf. Section 4.1). We supplement Chadha's approach with conflict resolution at system design time.

Gil et al. [6] dealt with conflict resolution at runtime. Their work presents an algorithm that detects ambiguities in ODRL policies. This solution goes beyond the conflict resolution required in the requirements engineering process. In contrast, we primarily identify conflicting logical components without going into details, such as whether time periods overlap, as demonstrated in [6].

Priebe et al. [21] introduced a pattern system for access control, including authorisation patterns, each describing the intention, context, problem, and solution. Putting its primary focus on security-related aspects, this approach only partly covers the context of policy patterns for business-to-business data sharing.

7 Discussion & Conclusions

Data usage conditions, expressed as policies, are essential for self-determined and autonomous data sharing by ensuring correct data handling and thus strengthening trust (cf. Section 1). Therefore, its consideration within the requirements

engineering process is of utmost importance. The usage of patterns allows for standardisation, reusability, and (automated) verifications. Since policies follow a structured design, patterns are particularly suitable for improving the quality and efficiency of their elicitation and analysis.

In this article, we proposed the DUP pattern for data sharing in the business-to-business domain. Starting from a conceptual model for policies, we presented a pattern consisting of six pattern primitives whose composition is defined by a policy grammar. In summary, this grammar consists of 17 productions that systematically structure data usage policies in data sharing. Following this, we described an approach for implementing our pattern by outlining a three-step process for policy analysis. This process includes conflict resolution as a central element. We concluded our work with a demonstration of the successful application of our DUP pattern in an example for a data sharing scenario as well as data usage requirements of real industrial dataspaces.

To answer RQ1, we focused on the structure of policies in business-to-business data sharing. We can observe that data usage policies follow a simple but powerful syntax independent of their technical representation. They are composed of actions and (optional) constraints that are refined to six pattern primitives. The abstraction level of our DUP pattern allows for an accurate and easily consumable usage during requirements engineering, being both human- and machine-readable.

Referring to RQ2, we structured the elicitation and analysis of policies during requirements engineering using a step-by-step process. The defined validation conditions can be used to ensure the correct instantiation of policy templates and patterns. They may guide requirements engineers and involved stakeholders through the phases of policy elicitation, decomposition, and conflict resolution, and can be technically validated. In addition, they could be used to implement policy validation functions and verifying them at system runtime.

7.1 Further Considerations

Although we limited the scope of this work to business-to-business data sharing, we believe that the DUP pattern and the policy analysis process, due to their level of abstraction and modular structure, could also be eligible in more specific or related contexts. If required, the grammar can be easily extended to refine existing parameters, such as the action type or the constraints, or specify new ones. A more specific scenario could be the applicability of our policy pattern for the design of application programming interfaces, considering service level agreements (SLAs) as a specific type of policy. SLAs define clear quality characteristics that are essential for reliability and trust in business-to-business data sharing. They may represent government regulations and legal obligations [24] that serve as input for the requirements elicitation process.

Related contexts for the DUP pattern include privacy-related scenarios. These scenarios could target business-to-customer data consumption or business-to-business sharing of personal data. When sharing personal data, e.g., involving

intermediaries or trustees, the pattern can be used to specify end-user preferences (privacy policies). For conflict resolution, these end-users could then be added as a new type of stakeholder with a custom weighting (cf. Section 4.1).

7.2 Future Work

Our approach could be further evaluated in different contexts, e.g., taking into account policies that focus on privacy-relevant data. Furthermore, the policy grammar could be extended so that it can be used for tools such as Antlr[10] or Xtext[11]. Future research areas could consider embedding the policy analysis process in a holistic requirements engineering process, as well as providing tool support.

Acknowledgments. We would like to thank Olaf Zimmermann, our supervisor in the shepherding process, for his valuable feedback and suggestions as experienced pattern author, which significantly improved this paper. We also appreciate the insights provided by the participants of the writers' workshop, listed in alphabetical order: Filipe Correia, Marcelo Nunes, Diogo Maia, Daniel Reis, Tiago Boldt Sousa, Francesco Urdih, and Uwe Zdun. Additionally, we acknowledge that this research was partially supported by the German Federal Ministry for Economic Affairs and Climate Action (funding number: 13IK040F).

This version of the contribution has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Amroune, M., Zarour, N.E., Charrel, P.J., Inglebert, J.M.: Composition of Aspectual Requirements: A Multi-criteria Process for Conflict Resolution. Journal of Software Engineering **8**(2), 75–88 (2014)
2. Biehs, S., Stilling, J.: Identification of Key Requirements for the Application of Data Sovereignty in the Context of Data Exchange. In: Proceedings of the 57th Annual Hawaii International Conference on System Sciences. pp. 4248–4257 (2024)
3. Chadha, R.: A cautionary note about policy conflict resolution. In: MILCOM 2006-2006 IEEE Military Communications conference. pp. 1–8. IEEE (2006)
4. Cirillo, F., Cheng, B., Porcellana, R., Russo, M., Solmaz, G., Sakamoto, H., Romano, S.P.: Intentkeeper: Intent-oriented Data Usage Control for Federated Data Analytics. In: 2020 IEEE 45th Conference on Local Computer Networks (LCN). pp. 204–215. IEEE (2020)
5. Eitel, A., Jung, C., Brandstädter, R., Hosseinzadeh, A., Bader, S., Kühnle, C., Birnstill, P., Brost, G., Gall, M., Bruckner, F., et al.: Usage Control in the International Data Spaces (March 2021)

[10] `https://www.antlr.org/` (accessed on 2025-04-13)
[11] `https://eclipse.dev/Xtext/` (accessed on 2025-04-13)

6. Gil, G., Arnaiz, A., Higuero, M., Diez, F.J., Jacob, E.: Context-Aware Policy Analysis for Distributed Usage Control. Energies **15**(19), 7113 (2022)
7. Heisel, M.: Agendas – A Concept to Guide Software Development Activites. In: Horspool, R.N. (ed.) Proc. Systems Implementation 2000. pp. 19–32. Chapman & Hall London (1998)
8. Hellmeier, M., Pampus, J., Qarawlus, H., Howar, F.: Implementing Data Sovereignty: Requirements & Challenges from Practice. In: Proceedings of the 18th International Conference on Availability, Reliability and Security. pp. 1–9 (2023)
9. Hosseinzadeh, A., Eitel, A., Jung, C.: A Systematic Approach toward Extracting Technically Enforceable Policies from Data Usage Control Requirements. In: Proceedings of the 6th International Conference on Information Systems Security and Privacy. pp. 397–405 (2020)
10. Iannella, R., Steidl, M., Myles, S., Rodríguez-Doncel, V.: ODRL Vocabulary & Expression 2.2 (2018), `https://www.w3.org/TR/odrl-vocab/`, Accessed: 2024-09-14
11. Jarke, M., Otto, B., Ram, S.: Data Sovereignty and Data Space Ecosystems. Business & Information Systems Engineering **61**(5), 549–550 (2019)
12. Mahendra, P., Ghazarian, A.: Patterns in the requirements engineering: A survey and analysis study. WSEAS Transaction on Information Science and Application **11**, 214–230 (2014)
13. Mitchell, R.K., Agle, B.R., Wood, D.J.: Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts. Academy of management review **22**(4), 853–886 (1997)
14. Munoz-Arcentales, A., López-Pernas, S., Pozo, A., Alonso, Á., Salvachúa, J., Huecas, G.: Data Usage and Access Control in Industrial Data Spaces: Implementation Using FIWARE. Sustainability **12**(9), 3885 (2020)
15. Otto, B.: Quality and value of the data resource in large enterprises. Information Systems Management **32**(3), 234–251 (2015)
16. Otto, B., ten Hompel, M., Wrobel, S.: Designing data spaces: The ecosystem approach to competitive advantage. Springer Nature (2022)
17. Otto, B., Jarke, M.: Designing a multi-sided data platform: findings from the International Data Spaces case. Electronic Markets **29**(4), 561–580 (2019)
18. Pampus, J., Heisel, M.: An Empirical Examination of the Technical Aspects of Data Sovereignty. In: Proceedings of the 19th International Conference on Software Technologies. pp. 112–122 (2024)
19. Pampus, J., Heisel, M.: Pattern-based Requirements Elicitation for Sovereign Data Sharing. vol. 254, pp. 147–156. Elsevier (2025)
20. Pettenpohl, H., Tebernum, D., Otto, B.: WFDU-net: A Workflow Notation for Sovereign Data Exchange. In: DATA. pp. 231–240 (2021)
21. Priebe, T., Fernández, E.B., Mehlau, J.I., Pernul, G.: A pattern system for access control. In: Research Directions in Data and Applications Security XVIII: IFIP TC11/WG11. 3 Eighteenth Annual Conference on Data and Applications Security July 25–28, 2004, Sitges, Catalonia, Spain. pp. 235–249. Springer (2004)
22. Rissanen, Erik: eXtensible Access Control Markup Language (XACML) Version 3.0 (January 2013), `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`
23. Sardinha, A., Araújo, J., Moreira, A., Rashid, A.: Conflict Management in Aspect-Oriented Requirements Engineering. Information Sciences and Technologies Bulletin of the ACM Slovakia **2**(1), 56–59 (2010)

24. Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., Pautasso, C.: Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. Addison-Wesley Signature Series (Vernon), Addison-Wesley Professional (2022)