GDPR Compliance Verification through Data Provenance

Alba Martinez Anton
Aix-Marseille Univ., LIS, CNRS
e-mail: alba.martinez-anton@univ-amu.fr

Clara Bertolissi
INSA Centre Val de Loire, LIFO, France
e-mail: clara.bertolissi@insa-cvl.fr

***Abstract*—With the exponential growth in the collection of personal data, ensuring compliance with data protection regulations such as the General Data Protection Regulation (GDPR) has become a critical challenge. In this work, we present a formal approach to GDPR compliance verification based on data provenance. We model the behavior of systems as provenance graphs, capturing the lifecycle of personal data across processes. Compliance patterns corresponding to key GDPR principles and rights are expressed as Prolog rules, enabling logical inference over these graphs. We present a verification tool which evaluates whether system executions meet legal obligations, and provides interpretable feedback in case of violations. We provide experimental validation on synthetic graphs to demonstrates the feasibility of our approach.**

Keywords-GDPR; provenance; verification; Prolog.

I. Introduction

With the increase in the number of computer systems and the proliferation of online services, the collection and processing of personal data has grown at an unprecedented scale. This growing dependence on data-intensive services makes the protection of personal data a critical concern. In response, legal frameworks such as the General Data Protection Regulation (GDPR) have been introduced to establish rights for individuals and responsibilities for data controllers and processors.

While the GDPR provides a clear legal foundation, its effective implementation remains a significant challenge. Verifying that a system complies with GDPR requirements involves analyzing both the system's specifications and its actual behavior during runtime. Being capable of enforcing, monitoring, and verifying compliance within the complex and dynamic ecosystems of modern digital infrastructures is still a challenge today. Indeed, compliance must be ensured at several levels: the design of processes, the configuration of data storage systems, and the execution of actions such as consent collection, data sharing, and deletion.

Manual compliance verification is not only tedious and error-prone, but often infeasible for complex systems with high data volumes and dynamic behaviors. Furthermore, existing auditing tools tend to focus on static policies or incomplete logs, and do not capture the full context of data usage over time. There is thus a growing need for formal, automated methods to evaluate GDPR compliance.

Fortunately, the increasing computerization of data processing offers an opportunity to support, and in some cases automate, parts of the compliance verification process. In particular, by leveraging structured representations of system behavior and data lifecycles, based on provenance models, it becomes possible to formalize and check compliance requirements seen as patterns in provenance graphs.

Approach. Our work proposes a formal approach to represent and verify compliance with key GDPR principles using provenance graphs. This representation enables us to track data processing chains and apply rule-based checks, thus bridging the gap between legal requirements and executable system behaviors. Our method complements existing compliance-by-design efforts by offering a verification mechanism capable of evaluating compliance post hoc, based on actual data processing traces.

Provenance provides a natural foundation for compliance verification because it captures who performed which operations on which data, when, and under which dependencies. Leveraging this representation, we identify the legal primitives relevant to GDPR, such as data items, data subjects, processing activities, purposes and rights-related actions, and map them to provenance entities, activities, and relations following the Open Provenance Model. This creates a uniform vocabulary in which legal requirements can be expressed independently of the underlying system. Then, we formalize GDPR principles, such as lawfulness of processing, the right to be forgotten, data minimization, and access rights, as declarative logic patterns over the provenance graph. These patterns specify constraints on the provenance graph structure, often involving specific sequences or combinations of events. For example, purpose limitation is formalized as a relationship between consent events, permitted purposes, and subsequent uses of personal data. Patterns are declarative and reusable: once defined, they apply to any provenance graph that adheres to the underlying schema, regardless of the system domain. This shifts the focus from tool-specific implementation details to a general, principle-driven method for analysing GDPR compliance through provenance.

The formal patterns constitute the foundation of our approach: they map regulatory concepts to precise constraints on the provenance graph, define reusable templates for expressing GDPR obligations, and support principled analyses of real system behaviors. To evaluate these patterns over concrete provenance data, we rely on a Prolog-based engine. The logical specifications are encoded directly as Prolog predicates. When the provenance graph is loaded, these predicates are instantiated with the corresponding entities, activities, and timestamps, allowing the engine to systematically traverse the graph and determine whether the constraints are satisfied or violated. This approach ensures coherence between the abstract legal formalization and the executable checks performed on real execution traces.

To assess scalability, we also developed a provenance graph generator based on modular building blocks (called "bricks") representing common user interactions. These building blocks can be assembled in various configurations to simulate realistic data flows in different application domains (e.g., social networks, online shopping, public administration). This enables us to test our approach on large and diverse graphs under controlled conditions.

Contributions. This work presents a comprehensive framework for the automated verification of GDPR compliance based on data provenance. It extends [1] by providing a detailed presentation of the generalized provenance model and by formalizing a significant subset of GDPR principles in the form of provenance patterns. Moreover, a graph generator was implemented, and an experimental evaluation using the Prolog prototype was carried out to assess performance results.

Our key contributions are:

- A formal representation of selected GDPR principles and data subject rights as logic-based patterns that can be evaluated over provenance graphs.
- A verification tool, implemented in Java and Prolog, that allows auditors to configure compliance checks, select specific rights or principles to verify, and receive detailed feedback on potential non-compliance or pending verifications.
- A method to produce synthetic but realistic datasets for experimentation. We generate realistic provenance graphs based on reusable components.
- An experimental validation showing the feasibility of our method on both manually crafted and large, synthetic provenance graphs, with performance analysis and scalability insights.

Overall, our work demonstrates how logical reasoning over provenance data can provide a powerful foundation for automated, explainable, and customizable compliance verification tools. This contributes to the broader effort of making digital systems not only more transparent, but also more accountable with respect to data protection regulations.

The structure of this paper is as follows. Section II discussed the state of the art. Section III introduces the context of the General Data Protection Regulation (GDPR) and motivates our work. Section IV presents the necessary preliminaries on provenance modeling and describes our extensions to capture GDPR requirements. In Section V, we detail our formal approach for modeling compliance requirements. Section VI outlines our methodology for semi-automated compliance verification together with a prototype tool implementation. Section VII reports on the experiments conducted to validate our approach. Finally, we conclude with the perspectives for future research in Section VIII.

II. Related Work

The General Data Protection Regulation has triggered a broad range of research efforts aimed at understanding, supporting, and enforcing data protection principles within digital systems.

Several studies have sought to analyze whether real-world systems comply with GDPR requirements [2][3][4]. These works often rely on case studies, audits, or systematic evaluations of privacy policies and data processing practices. While valuable for revealing compliance gaps, such approaches are mostly retrospective and provide limited automation or generalizability.

Another line of research focuses on developing technical solutions to support compliance checking in a more systematic way, see for instance [5][6][7]. These works are based on the formalization of legal norms themselves. This effort is crucial to support both design- and runtime compliance, as it provides the formal foundation upon which reasoning, verification, and automation can be built. Translating legal texts into formal logic is a complex and delicate task that requires capturing the subtleties of legal language, its contextual dependencies and normative intent in a structured, machine-interpretable form.

Various approaches have been proposed to bridge the gap between natural legal language and logical formalism. A widely adopted strategy involves the use of ontologies to model legal knowledge. As discussed in [8], ontologies provide a structured representation of legal concepts and their interrelations, making it easier to translate legal provisions into sets of logical rules. This modeling facilitates knowledge sharing, consistency checking, and reasoning over legal domains. While ontology-based approaches offer rich semantic models of GDPR concepts, they primarily support reasoning over declared system properties. In contrast, our approach focuses on operational compliance: provenance graphs capture the actual flow of personal data and the causal structure of system executions. This makes it possible to detect violations that arise from concrete system behavior, rather than from inconsistencies in the system's specification. Ontology-based models and provenance-based verification may be complementary: ontologies can supply high-level semantic annotations, whereas provenance provides the fine-grained, event-level evidence required for auditing real system executions.

Another notable development is LegalRuleML [9], an XML-based extension specifically designed to represent legal norms, their conditions of applicability, and the structure of legal arguments. LegalRuleML aims to support automated reasoning and interoperability between legal information systems, offering a standardized way to encode normative content.

Complementary to these formalization efforts, some research has focused on making logical representations accessible and verifiable by legal experts. For instance, Bartolini et al. [10] propose using RIO logic to express legal provisions in a structured but human-readable format. This representation allows legal practitioners to validate the formalization of regulations while preserving the rigor needed for automated compliance checking. Such approaches enhance the transparency and trustworthiness of logic-based systems used in legal contexts.

Since the adoption of the GDPR in 2016, considerable research has focused on designing systems that are compliant by design, in line with Article 25 of the regulation. In [11], the authors extract Technical and Organizational Measures (TOMs) directly from the GDPR text to support the development of a compliance-assisting tool. This tool relies on contractual relationships between users and organizations, implemented through knowledge graphs. This formalization allows the system to reason over contractual obligations and compliance requirements in a machine-interpretable way.

Building upon this foundation, [12] extends the previous work by incorporating the notion of consent and adapting the tool to the context of smart cities and IoT environments. The primary goal of this enhancement is to ensure GDPR compliance for resource-constrained or distributed infrastructures.

Other studies, such as [13], approach compliance by focusing on business process modeling. The authors propose a methodology for aligning business workflows with the obligations imposed on data controllers by the GDPR. While the work offers a structured approach to incorporating privacy by design, it remains theoretical, demonstrated only through a case study involving a telecommunications operator, and does not include a dedicated tool or automation component.

Similarly, [14] employs business process models to embed compliance features in IoT systems. Their work stands out by introducing smart contracts and blockchain technologies to facilitate automated compliance verification. Drawing on the methodology of [15], they address several key GDPR principles, including confidentiality, consent, data minimization, data protection, and data transfer.

Also, numerous frameworks and guidelines have been proposed to support developers in incorporating privacy requirements throughout the system lifecycle. For instance, Saltarella et al. [16] survey approaches that support system designers in aligning their architectures with GDPR principles from the outset. Similarly, Rhahla et al. [17] present a comprehensive study of academic and industrial tools intended to facilitate GDPR compliance, with a focus on development-time integration.

Compared to these approaches, that emphasizes design-time compliance or high-level modeling, our contribution aims to formalize and verify that runtime behaviors, specifically, actions performed on personal data, adhere to the GDPR's principles and constraints. In contrast to the above efforts, relatively few works have addressed the runtime verification of actions and behaviors performed by systems, especially from the perspective of compliance with GDPR obligations. Ensuring that concrete system operations, such as data access, sharing, deletion, or processing, respect legal constraints in practice remains a largely open challenge. Existing approaches often lack the granularity or formality needed to assess compliance at the level of individual actions or data flows.

In [6], the authors translate a subset of GDPR articles into temporal logic formulas to automate compliance verification. Their approach leverages MonPoly, a monitoring tool that processes system event logs by translating them into MFOTL logic and compares them against GDPR rules to detect potential violations. While effective for specific cases, this work remains limited to a small subset of GDPR provisions and does not address key principles such as purpose limitation.

Building on a broader scope, [18] introduces a framework developed within the European project Cloud4EU [19], which combines different techniques to verify GDPR compliance in cloud environments for the public sector. Their contribution lies in the semi-automated translation of regulatory texts into logical rules, enabling dynamic updates as new laws or amendments are introduced. These rules can be applied both to runtime log verification and to compliance-by-design approaches, where traces generated from business models are checked against GDPR requirements. Nevertheless, the use of LegalRuleML within this framework remains relatively basic. As the authors themselves highlight, the current metamodel is primarily geared toward legal representation and requires significant extensions to support advanced compliance reasoning.

Other lines of research explore the use of machine learning to automate parts of this process, especially the translation of legal texts. For example, [20] investigates text processing techniques to facilitate compliance reasoning. Similarly, Zimmeck et al. [21] apply natural language processing to automatically extract GDPR-relevant requirements from privacy policies, demonstrating the potential of NLP for large-scale analysis. However, these approaches still face challenges in accurately capturing certain GDPR concepts, whose semantic complexity and context-dependence are difficult to formalize.

Our work follows the same general line of formal specification of [6], which relies on temporal logic to reason about consent and event sequences, but we adopt a broader provenance-based perspective. By introducing a set of purpose-oriented and constraint-based provenance patterns, our framework supports the expression of GDPR principles that involve conditional or evolving requirements. This makes it suitable for capturing a wider range of compliance scenarios, including the verification not only of whether consent exists for a given piece of data, but also of whether the specific purpose stated in the consent request is actually respected. Moreover, in contrast with approaches based on temporal logic or high-level ordering constraints between events, our provenance-based representation provides explicit temporal information attached to provenance activities. Therefore, we can determine exactly when a consent was granted, how long it remained valid, and during which period a given piece of personal data was accessed, transmitted, or stored. By using in the analysis concrete time intervals extracted in the provenance graph directly from system logs, our approach captures forms of temporal misuse that cannot be detected using purely qualitative or symbolic temporal relations.

III. Context and Motivations

A. The General Data Protection Regulation

In order to safeguard the privacy of its citizens, the European Union adopted a directive in 1995 [22], introducing the notions of informed consent, the right of access and rectification, and the obligations of data controllers with respect to the collection, processing, and storage of personal data. This directive laid the foundation for the General Data Protection Regulation

(GDPR), which came into force in 2018 and has since become the cornerstone of European data protection law.

The GDPR establishes a comprehensive framework that regulates the collection, processing, and dissemination of personal data, and it applies to any organization, whether public or private, operating within the EU or targeting European residents. In its Article 4, the regulation defines key concepts to delineate its scope. Personal data is understood as any information relating to an identified or identifiable natural person, the data subject. Processing is broadly defined as any operation performed on personal data, from collection and recording to storage, modification, or dissemination. The notion of data controller refers to the person or organization that determines the purposes and means of the processing and is ultimately responsible for compliance. Consent is defined as a freely given, specific, informed, and unambiguous indication by the data subject of their agreement for personal data to be processed for a clearly stated purpose.

The regulation is articulated around a set of fundamental principles that ensure lawful, fair, and transparent processing, restrict the use of data to explicit and legitimate purposes, and require that only the necessary amount of information be collected and maintained accurately over time. It also limits storage to what is strictly necessary, obliges organizations to guarantee the confidentiality and integrity of data through appropriate safeguards, and finally, introduces the principle of accountability, by which controllers must be able to demonstrate their compliance at all times. In addition to these principles, the GDPR includes a wide range of rights for individuals, including the right of access, rectification, erasure (often referred to as the "right to be forgotten") as well as the rights to restrict processing, to data portability, and to object to certain forms of processing. Together, these provisions aim to give individuals effective control over their personal information in the digital society.

B. Motivations

Although the objectives of the GDPR are clear, ensuring compliance in practice represents a considerable challenge for organizations. This challenge is both organizational and technical, since it requires translating often complex legal requirements into operational practices and embedding them within the functioning of information systems. A particularly delicate issue concerns the notion of purpose. Organizations must not only declare the reasons why they collect and process personal data, but also guarantee throughout the entire lifecycle of the data that these purposes are respected and remain aligned with the consent initially given by individuals. Tracking and documenting these purposes across complex systems is a demanding task, especially when data is reused in multiple contexts.

Compliance also demands meticulous documentation of processing activities, the legal bases invoked, the consents obtained, and the technical and organizational measures put in place to ensure security. These obligations are generally verified through audits, which assess the adequacy of procedures. However, such audits are often labor-intensive and retrospective in nature. Automating them, at least partially, is a pressing need if organizations are to sustain compliance in a scalable and efficient manner.

Further difficulties arise from the necessity to ensure that modifications such as rectification or erasure are consistently propagated across all replicas. This issue of replication consistency is critical for respecting principles such as accuracy and storage limitation, as well as rights such as the right to erasure.

These different challenges leave traces within the systems themselves, in the form of logs or event records that capture the sequence of operations performed on data. By analyzing these traces, it becomes possible to assess whether certain principles or rights have been respected, thereby opening the way to automated forms of compliance verification. However, it is important to recognize that not all principles or rights lend themselves to such automation. Certain notions embedded in the GDPR, such as fairness and transparency, cannot be automatically verified because they depend on communication methods and contextual nuances that system traces cannot capture. The automation of legal verification therefore has limits: while it can significantly reduce human error, accelerate verification processes, and help maintain continuous compliance, it cannot fully replace human judgment when it comes to subjective assessments. For instance, the evaluation of whether a request for consent is expressed in clear and understandable terms (article 12 to 14) remains outside the reach of automated systems. Similarly, the principle of data minimization raises complex challenges, since determining whether a dataset is “necessary” (article 5.1.c) for a given purpose often requires domain expertise and contextual knowledge. In such cases, compliance is better supported by establishing precise specifications of procedures and processes, and then verifying that these specifications are respected. Furthermore, many parts of the GDPR, such as Chapters 6, 7, and 8, dealing respectively with supervisory authorities, cooperation mechanisms, and remedies and penalties, regulate institutional, procedural and enforcement aspects rather than the technical or operational rules of data processing.

Given these limitations, our work focuses on the subset of GDPR provisions that can be addressed through system-level analysis. More specifically, we concentrate on the principles of lawfulness, purpose limitation, and storage limitation, as well as the rights of access and erasure. These aspects are particularly amenable to verification through system traces and provenance data, since they can be formalized as constraints on the occurrence, timing, and consistency of processing events. Our approach therefore emphasizes the automation of articles directly concerned with event compliance, while considering semi-automated methods for those that require complementary verification.

In this way, we aim to contribute to the broader goal of bridging the gap between legal requirements and system implementation, by providing methods that support both automation where feasible and human oversight where necessary.

C. Examples

In this work, we concentrate on three aspects that we consider fundamental: the traceability of purposes, the enforcement of storage limitations, and the consistency of data erasure across replicas.

Purpose traceability requires that consented purposes accompany personal data throughout all processing activities, ensuring that data is not diverted from its legitimate use.

Example 1 (Purpose traceability): A system respects the purpose limitation principle when personal data is processed exclusively for the purposes explicitly consented to by the data subject. Consider the following situation in an online forum: Bob joins a discussion group, and as part of this action, his personal data is associated with the creation of cookies. Two different scenarios illustrate compliance and non-compliance.

Compliant scenario: Bob joins a cooking interest group. In this context, the forum generates an analytics cookie used solely to improve the performance of the website. The cookie remains internal to the platform and is accessible only to the maintenance team, fully aligned with the purpose to which Bob initially consented.

Non-compliant scenario: Bob joins the same cooking interest group, and the forum once again creates an analytics cookie. However, despite Bob having explicitly withdrawn his consent for any sharing with third parties, the cookie is nonetheless transmitted to an external kitchenware vendor. This constitutes a clear violation of the purpose limitation principle, as the data is processed beyond the scope of the consent provided.

Concerning storage limitations, temporal constraints must be verified to check that data is retained no longer than necessary and that requests from data subjects are honored within prescribed deadlines.

Example 2 (Data storage limitation): The storage limitation principle requires that personal data be retained only for as long as necessary in relation to the purposes for which it was collected. Automating compliance with this principle typically involves checking that data subject information is deleted or anonymized once the retention period has expired. Consider the case of Alice, who creates an account on an online forum and contributes content. The following scenarios illustrate compliance and non-compliance.

Compliant scenario: Alice registers on the forum and actively participates by posting reviews. After five years of inactivity, the system enforces its retention policy and deletes Alice's account data. Two years later, when Alice attempts to log in again, she receives an error message indicating that the account no longer exists. This demonstrates compliance with the predefined five-year retention limit.

Non-compliant scenario: Alice registers on the forum and posts several reviews. After seven years without activity, she attempts to log in again and is still able to access her account with all of her personal data intact. This persistence of data beyond the declared five-year limit constitutes a violation of the storage limitation principle.

Data erasure consistency, finally, is needed to guarantee that compliance-related actions such as deletion are properly applied across all copies of the data within the system.

Example 3 (Consistency of data deletion across copies): Compliance with the right to erasure requires that when a deletion request is made, all copies of the corresponding data within the system are removed. These copies may exist in different modules of the system, such as the main database, user activity logs, or temporary caches. Consider the following scenarios for David, who has an account in an online forum.

Compliant scenario: David requests the deletion of his account. The system processes the request by erasing his personal data from the main user database, cleaning the activity logs, and clearing cached data related to his profile. When David later attempts to log in, his account no longer exists, confirming that all copies of his data have been consistently deleted.

Non-compliant scenario: David deletes his account, and the system removes it from the main user database. However, his data remains in the activity logs used by the forum administrators. Weeks later, traces of David's activity are still accessible, revealing that the deletion request was not fully enforced across all system copies, thus violating the GDPR.

Together, these challenges highlight the necessity of systematic and automated approaches to compliance verification, which we aim to address through the use of provenance information.

IV. Preliminaries

In this section we present the Open Provenance model [23] and its generalization to represent GDPR requirements.

A. The Open provenance model

Provenance information is usually represented as a labeled direct graph (called *provenance graph*) that expresses how objects evolve in the system. A provenance graph is a triple $G = (\mathcal{N}, \mathcal{E}, L)$, where $\mathcal{N}$ is a set of nodes, $\mathcal{E}$ is a set of directed edges between nodes, and L is a set of labels. A node in the graph can represent an *artifact*, which is an immutable piece of data, a *process*, which is used to denote the action performed on an artifact and resulting in a new artifact; or an *agent*, which is used to denote the entity controlling or affecting the execution of a process. We denote the sets of artifacts, processes, and agents as $\mathcal{O}$, $\mathcal{P}$, and $\mathcal{A}$, respectively (with $\mathcal{N} = \mathcal{O} \cup \mathcal{P} \cup \mathcal{A}$ and $\mathcal{O}, \mathcal{P}, \mathcal{A}$ pairwise disjoint). Edges represent labeled casual dependencies between nodes. Role labels $\mathcal{R} \subseteq L$ define the function of an agent or an artifact in a process.

The Open Provenance model defines three main causal dependencies (see Table I): used, wasGeneratedBy, and wasControlledBy. The first two dependencies indicate a link between an artifact and a process: used connects a process to the artifacts used in the process and wasGeneratedBy links a process to the resulting artifact. Dependency wasControlledBy indicates which agent controls a process. The Open Provenance model also includes additional dependencies that can be derived from a provenance graph by composing dependencies. For instance, composed dependency wasTriggeredBy indicates that the execution of a process was triggered by another process, and dependency wasDerivedFrom denotes that an artifact was derived from another artifact. As done in [24], we make the role parameter explicit to be able to specify what role the first artifact played in the creation of the new one. Multi-step dependencies can be defined by transitive closure. For instance, wasDerivedFrom$^+$ captures, for an artifact, all artifacts used to derive it, possibly indirectly. Similarly, wasTriggeredBy$^+$ captures, for a process, all its direct and indirect triggering processes. The latter is also used to define the indirect use (used$^+$) and generation of artifacts (wasGeneratedBy$^+$).

To reason over temporal aspects and information evolution in a system, we extend the Open Provenance model with time observations, following [23]. We assume time is measured by an observer according to a single clock (or synchronized clocks), in such a way that time observations may be comparable. Moreover, observed time is expected to be compatible with causal dependencies, e.g., an artifact must exist before it is being used, a process generates artifacts before it ends, etc. In other words, causality implies time precedence. Provenance causal dependencies are decorated with time information expressed as timestamps associated with edges. We consider a series of ordered time points denoted as $ts_1, \ldots, ts_n \in \mathcal{T}$, with $ts_1 < ts_2 < \ldots < ts_n$, which represent specific points in time. We define a time interval $[ts_s, ts_e]$, as the interval of time between $ts_s \in \mathcal{T}$, starting time point, and $ts_e \in \mathcal{T}$, ending time point. We assume that a process runs for a time interval $[ts_s, ts_e]$, with $ts_s, ts_e \in \mathcal{T}$. All used or generated artifacts are used or created during this time window. Accordingly, in the provenance model with time observations, edges "wasControlledBy" are extended with the time interval $[ts_s, ts_e]$, and edges "used" and "wasGeneratedBy" are extended with a timestamp $ts \in [ts_s, ts_e]$ indicating that the artifact was used by the process (in the case of used) or that the artifact was created by the process (in the case of wasGeneratedBy) at a given time ts.

Following [24], we introduce two custom dependencies to capture the owner and the contributors of an artifact, as presented in the third block of Table I. In particular, owns captures the classical assumption in discretionary access control where the agent creating an artifact owns that artifact; contributedTo extends dependency wasControlledBy to identify the agents that have contributed, directly or indirectly, to the creation of an artifact. While in [24], the dependency wasControlledBy and, in turn, the dependency contributedTo were used only to link a process with an agent involved actively in a process, here we extend these dependencies to link a process to an agent involved actively or passively in the creation of an artifact. To this end, we explicitly model a role in contributedTo to denote in which way a subject was involved in the creation. If the role in contributedTo is *owner*, the casual dependency has the same meaning as the one in [24].

We also introduce the dependency notAvailable to capture that a previously created artifact is no longer available in the system (i.e., the artifact has been deleted).

As done previously, we can extend these dependencies with a timestamp. We suppose that an agent has owned an artifact since its generation (i.e., the timestamp associated with the edge owns matches the time parameter in the composing wasGeneratedBy dependency). Similarly, for the dependency contributedTo, its timestamp coincides with the time associated with the generation of the artifact. The notAvailable dependency has a timestamp indicating the time when the artifact has been deleted (thus matching the timestamp of the composing used dependency). It is worth noting that the definition of this last dependency uses a predicate **action** that, given a specific process instance, returns its type. This is needed because policies typically specify "types" of actions rather than process instances, as demanded by the provenance graph.

Provenance information can be represented graphically, as illustrated in Fig. 1. Following the standard notation for provenance graphs [23], artifacts are represented as circles, processes as rectangles, and agents as octagons. Edges are annotated with the type of dependency, the role of artifacts and agents in processes and the temporal information. For instance, in the graph of Figure 1, one can see that the artifact `data report` has been generated by the process `sendData` using the artifact `data request`.

B. Extending provenance information for compliance

In the context of the GDPR, it is crucial to represent personal data in a way that allows one to identify the processes acting upon it, the purposes for which it is used, the consent given by data subjects, and the degree of compliance with user rights.

To achieve this, we extend the provenance model so that it explicitly distinguishes personal data and better captures consent mechanisms. This extension is implemented through the introduction of additional artifact nodes and enriched node attributes.

a) Specific nodes.: When personal data is involved, individuals are typically presented with a template of possible choices for granting or denying consent (a privacy policy template). In the provenance graph, such a template is represented as an artifact node. Once the individual makes a decision, the corresponding consent is also modeled as an artifact. If consent is later modified, a new artifact is created to represent the updated state. This new artifact is linked to the previous one via a wasDerivedFrom edge, ensuring versioning of consent. For instance, in Figure 1, Bob's consent evolves from *consent_bob_v0* to *consent_bob_v1* after he adjusts the purposes he had initially authorized.

Purposes themselves are captured as attributes associated with consent artifacts. It is important to note that artifacts in the provenance model are immutable objects: any modification to data or consent does not overwrite the existing artifact but

Basic casual dependencies from [23]
$\mathsf{used}(p:\mathcal{P}, d:\mathcal{O}, r:\mathcal{R}, ts_u:\mathcal{T})$
$\mathsf{wasGeneratedBy}(d:\mathcal{O}, p:\mathcal{P}, r:\mathcal{R}, ts_g:\mathcal{T})$
$\mathsf{wasControlledBy}(p:\mathcal{P}, s:\mathcal{A}, r:\mathcal{R}, ts_b \in \mathcal{T}, ts_e:\mathcal{T})$
$\mathsf{wasTriggeredBy}(p_1:\mathcal{P}, p_2:\mathcal{P}, ts_u:\mathcal{T}) = \exists d \in \mathcal{O}, r_1, r_2 \in \mathcal{R}, ts_g \in \mathcal{T} : \mathsf{used}(p_1, d, r_1, ts_u) \wedge \mathsf{wasGeneratedBy}(d, p_2, r_2, ts_g)$
$\mathsf{wasDerivedFrom}(d_1:\mathcal{O}, d_2:\mathcal{O}, r:\mathcal{R}, ts_g:\mathcal{T}) = \exists p \in \mathcal{P}, r_i \in \mathcal{R}, ts_u \in \mathcal{T} : \mathsf{wasGeneratedBy}(d_1, p, r_i, ts_g) \wedge \mathsf{used}(p, d_2, r, ts_u)$
Multi-step casual dependencies from [23]
$\mathsf{used}^+(p:\mathcal{P}, d:\mathcal{O}, ts_b:\mathcal{T}, ts_e:\mathcal{T}) = \exists p_i \in \mathcal{P}, r \in \mathcal{R}, ts_{b'} \in \mathcal{T} : \mathsf{wasTriggeredBy}^+(p, p_i, ts_{b'}, ts_e) \wedge \mathsf{used}(p_i, d, r, ts_b)$
$\mathsf{wasGeneratedBy}^+(d:\mathcal{O}, p:\mathcal{P}, ts_b:\mathcal{T}, ts_e:\mathcal{T}) = \exists p_i \in \mathcal{P}, r \in \mathcal{R}, ts'_e \in \mathcal{T} : \mathsf{wasGeneratedBy}(d, p_i, r, ts_e) \wedge \mathsf{wasTriggeredBy}^+(p_i, p, ts_b, ts'_e)$
$\mathsf{wasTriggeredBy}^+(p_1:\mathcal{P}, p_2:\mathcal{P}, ts_b:\mathcal{T}, ts_e:\mathcal{T}) = (\mathsf{wasTriggeredBy}(p_1, p_2, ts_e) \wedge (ts_b = ts_e)) \vee (\exists p_i \in \mathcal{P}, ts_{e'} \in \mathcal{T} : \mathsf{wasTriggeredBy}(p_1, p_i, ts_e)$ $\wedge \mathsf{wasTriggeredBy}^+(p_i, p_2, ts_b, ts'_e))$
$\mathsf{wasDerivedFrom}^+(d_1:\mathcal{O}, d_2:\mathcal{O}, ts_b:\mathcal{T}, ts_e:\mathcal{T}) = (\mathsf{wasDerivedFrom}(d_1, d_2, r, ts_e) \wedge (ts_b = ts_e)) \vee (\exists d_i \in \mathcal{O}, r_i \in \mathcal{R}, ts_{e'} \in \mathcal{T} : \mathsf{wasDerivedFrom}(d_1, d_i, r_i, ts_e)$ $\wedge \mathsf{wasDerivedFrom}^+(d_i, d_2, ts_b, ts_{e'}))$
Additional casual dependencies
$\mathsf{owns}(s:\mathcal{A}, d:\mathcal{O}, ts_g:\mathcal{T}) = \exists p \in \mathcal{P}, r \in \mathcal{R}, ts_b, ts_e \in \mathcal{T} : \mathsf{wasGeneratedBy}(d, p, r, ts_g) \wedge \mathsf{wasControlledBy}(p, s, \mathtt{owner}, ts_b, ts_e)$
$\mathsf{contributedTo}(s:\mathcal{A}, d:\mathcal{O}, r:\mathcal{R}, ts_g:\mathcal{T}) = \exists p \in \mathcal{P}, ts_b, ts_e \in \mathcal{T} : \mathsf{wasGeneratedBy}^+(d, p, ts_g, ts) \wedge \mathsf{wasControlledBy}(p, s, r, ts_b, ts_e)$
$\mathsf{notAvailable}(d:\mathcal{O}, ts:\mathcal{T}) = \exists p \in \mathcal{P}, r \in \mathcal{R} : \mathsf{used}(p, d, r, ts) \wedge \mathsf{action}(p) = \mathtt{delete}$

TABLE I. CASUAL DEPENDENCIES FOR OUR MODEL

instead gives rise to a new artifact. This new version is linked back to its predecessor through a wasDerivedFrom dependency with the role *update*, thereby maintaining the integrity of the evolution history.

b) Specific attributes: Our model relies on two main types of attributes. The first serves to distinguish artifacts that constitute personal data. Since the GDPR explicitly governs the processing of personal data, only these artifacts are subject to compliance checks; other data in the system remains outside its scope unless it embeds personal information. To capture this distinction, we introduce the Boolean attribute dp. For example, $dp(email\_bob) = True$ indicates that the artifact $email\_bob$ represents personal data.

The second type of attribute encodes the purposes for which consent has been granted. A purpose corresponds to an action performed by a process (e.g., purchase, information request) captured by the predicate *action*, introduced in the previous section.

Since consent links personal data to authorized purposes, this relation is captured within a dedicated $purposes$ attribute on consent artifacts. This attribute can be formalized as a list of tuples, each associating an artifact (personal data) with the set of authorized purposes. We denote the $purposes$ attribute of the $cons$ artifact as $purposes(cons) = [(artefact_1, list_1), ..., (artefact_k, list_k)]$.

When the system does not allow purpose specification at the level of individual artifacts, a simplified representation is used: $k = 1$ and $artifact_1 = \_$, meaning that the consent purposes apply globally to all personal data.

C. Running example

To illustrate our approach, we introduce a running example that will be used throughout the paper. Figure 1 depicts an online forum modeled as a provenance graph. In this system, users can create accounts to join discussion groups, where they are able to post and reply to messages. The forum adopts a global approach to purposes: consent is defined at the account level and does not depend on individual data items.

In the example, Bob creates an account (*create_account* process) and provides several pieces of personal information such as his e-mail address and telephone number, which are stored by the system. At registration, an identity number is automatically assigned, along with a personal wall and a friends list, each of these elements being considered personal data under the GDPR. Bob must also specify the purposes for which his data may be processed. Initially, he authorizes its use for both marketing (including third-party sharing) and analytics (service improvement), a choice represented in the graph by the artifact $consent\_bob\_v0$. Later, he revises his preferences by withdrawing consent for marketing while keeping analytics enabled. This update produces a new artifact, $consent\_bob\_v1$, linked to the previous one as described in the preceding section.

Subsequently, Bob joins a cooking interest group. At this point, the system generates a cookie, which is sent to an external vendor of kitchenware, despite Bob's updated consent. This illustrates a non-compliant behavior with respect to the principle of purpose limitation.

The graph also shows the role of the Data Controller (DC agent). For example, Bob requests access to his personal data, and the controller provides a response. He later asks for the deletion of his phone number (*ask_erase* process), but this request remains unfulfilled, highlighting another case of non-compliance.

Notice that the graph, timestamps are expressed in minutes and, for readability reasons, node attributes (such as purpose or personal data markers) are not explicitly visualized in the provenance graph, even though they are formally associated with the corresponding artifacts.

Example 4 (Purpose attribute): In the scenario illustrated in Figure 1, purposes are defined globally. The $purpose$ attribute of the artifact $consent\_bob\_v0$ is expressed as:

$$purposes(consent\_bob\_v0) = \\ [(\_, ["thirdParties", "analysis", "improvement"])]$$

This notation means that Bob's consent applies uniformly to all his personal data, authorizing its use for third-party sharing, analysis, and service improvement.

Now consider an extended version of the forum that includes an integrated online shop. To better align with the principle of data minimization, the system allows users to grant different consent for each category of personal data. For instance, Bob may agree that his physical address, e-mail, and bank account can be used when purchasing products, but restrict refunds to rely solely on his bank account number. In this case, the consent attribute would be represented as follows:

$$\begin{aligned}&purposes(consent\_bob\_v0) = \\ &[(name\_bob, [\text{"}buy\text{"}, \text{"}refund\text{"}]), \\ &(address\_bob, [\text{"}buy\text{"}]), (email\_bob, [\text{"}buy\text{"}]), \\ &(bank\_account\_bob, [\text{"}buy\text{"}, \text{"}refund\text{"}])]\end{aligned}$$

This finer-grained modeling provides a more precise representation of user preferences, ensuring that data is processed strictly within the scope for which explicit consent has been granted.

Our work proposes to use provenance graphs to represent the evolution of the system over time and to model the principles and rights defined in the GDPR using patterns, defined as path expressions in the next section.

V. Modeling principles as provenance patterns

Some of the GDPR principles introduced in Section III are directly related to data processing, which makes the compliance verification easier to be automated or semi-automated.

In this section we formalize the selected GDPR principles as patterns in a provenance graph representing the system execution history, then we focus on user rights verification. A pattern is a path expression composed of conjunctions and/or disjunctions of causal dependencies. Possibly, temporal constraints or constraints on attribute values can be included in patterns. The constraints on attribute values ensure that the value of the attribute in the graph corresponds to the value in the pattern. If the attribute value is a list, as in the case of the *purposes* attribute, the constraint is verified if the value belongs to the list.

When we use negation in our patterns, we follow the negation by failure principle, similar to what is used in Prolog. This means that a pattern is considered negated (i.e., false) if the system fails to find any instance that satisfies it. In other words, rather than proving that the negated condition is logically false, we assume it is false because no matching example can be found.

A. Principles of the GDPR

In this section we start by formalizing the lawfulness principle, related to the notions of consent and purpose. Then, we model the storage limitation principle, related to the notion of legal retention limit.

1) Consent and purpose limitation: As previously introduced, consent and purpose are closely linked. Indeed, the GDPR stipulates in its basic principles that any data processing must be consented to and imposes purpose limitation as a principle. We present next how to formally define the patterns related to the purpose and the presence or absence of consent.

We write $\mathsf{isPurpose}(PU, D, C)$ if the purpose PU has been consented for the personal data D in the consent artifact C. If the piece of data is not specified, we simply write $\mathsf{isPurpose}(PU, \_, C)$, where _ stands for any personal data and the purpose is called *global*.

Personal data cannot be used if the data subject has not given consent for the intended purpose (Art. 6.1.a). To determine whether the data subject has consented to the use of his or her personal data D for the purpose PU, there must be a consent artifact C for which PU is a purpose for D. The following path expression formalizes this:

$$\begin{aligned}&\mathsf{consent}(C, D, PU, T) = \\ &\mathsf{wasControlledBy}(P1, S, \text{"}owner\text{"}, TB, TE)\wedge \\ &\mathsf{wasGeneratedBy}(C, P1, \text{"}consent\text{"}, T)\wedge \\ &\mathsf{isPurpose}(PU, D, C)\end{aligned}$$

where C represents the consent (artifact) created by the process "consent" $P1$ controlled by the data subject S as owner of the personal data D. The timestamps T, TB, and TE are included in the pattern because there may be multiple instances of the consent C within the graph. The consent may correspond either to the one initially provided by the data subject (as represented by the `consent` process in Figure 1) or to a subsequent modification (the `update` process in the same figure). Once the value of T is instantiated, it uniquely identifies a specific occurrence of C, allowing it to refer to a single artifact.

Example 5 (Pattern for consent):

In the scenario presented in Figure 1, the pattern $\mathsf{consent}(consent\_bob\_v0, email\_bob, \text{"}analysis\text{"}, T)$ is evaluated to true.

This is because the path

$$\begin{aligned}&\mathsf{wasControlledBy}(\texttt{consent}, Bob, \text{"}\texttt{owner}\text{"}, 16, 20)\wedge \\ &\mathsf{wasGeneratedBy}(consent\_bob\_v0, \texttt{consent}, \text{"}\texttt{consent}\text{"}, 17)\end{aligned}$$

exists in the provenance graph. The consent artifact $consent\_bob\_v0$ contains the global purpose *"analysis"* and the timestamp of its generation is indeed 17.

2) Consent revocation and update: As stated in Article 7.3 of the GDPR, the data subject must be able to withdraw his/her consent at any time. Consent revocation is modeled using the predicate *revoke* and the following pattern:

$$\mathsf{revoke}(C, T) = \mathsf{used}(P, C, \text{"}revokeConsent\text{"}, T)$$

where C refers to the consent artifact, T is the timestamp indicating the time of revocation, and P denotes the process responsible for revoking the consent. Since the revocation applies to all purposes of all associated artifacts, the predicate is independent of any specific data or purpose.

Instead of being entirely revoked, consent preferences can be modified by the data subject, either to add or remove purposes. In such cases, a new artifact is created in the provenance graph, derived from the previous consent artifact, through a process that performs an update initiated by the data subject.

The following pattern allows us to verify whether a consent artifact C has been updated by the data subject S at a given time T:

$$\begin{aligned}&\mathsf{nextConsent}(C, C1, T) = \\ &\mathsf{wasControlledBy}(P1, S, \text{"}owner\text{"}, TB, TE)\wedge \\ &\mathsf{used}(P1, C, \text{"}updateConsent\text{"}, TU)\wedge \\ &\mathsf{wasGeneratedBy}(C1, P1, \text{"}consent\text{"}, T)\end{aligned}$$

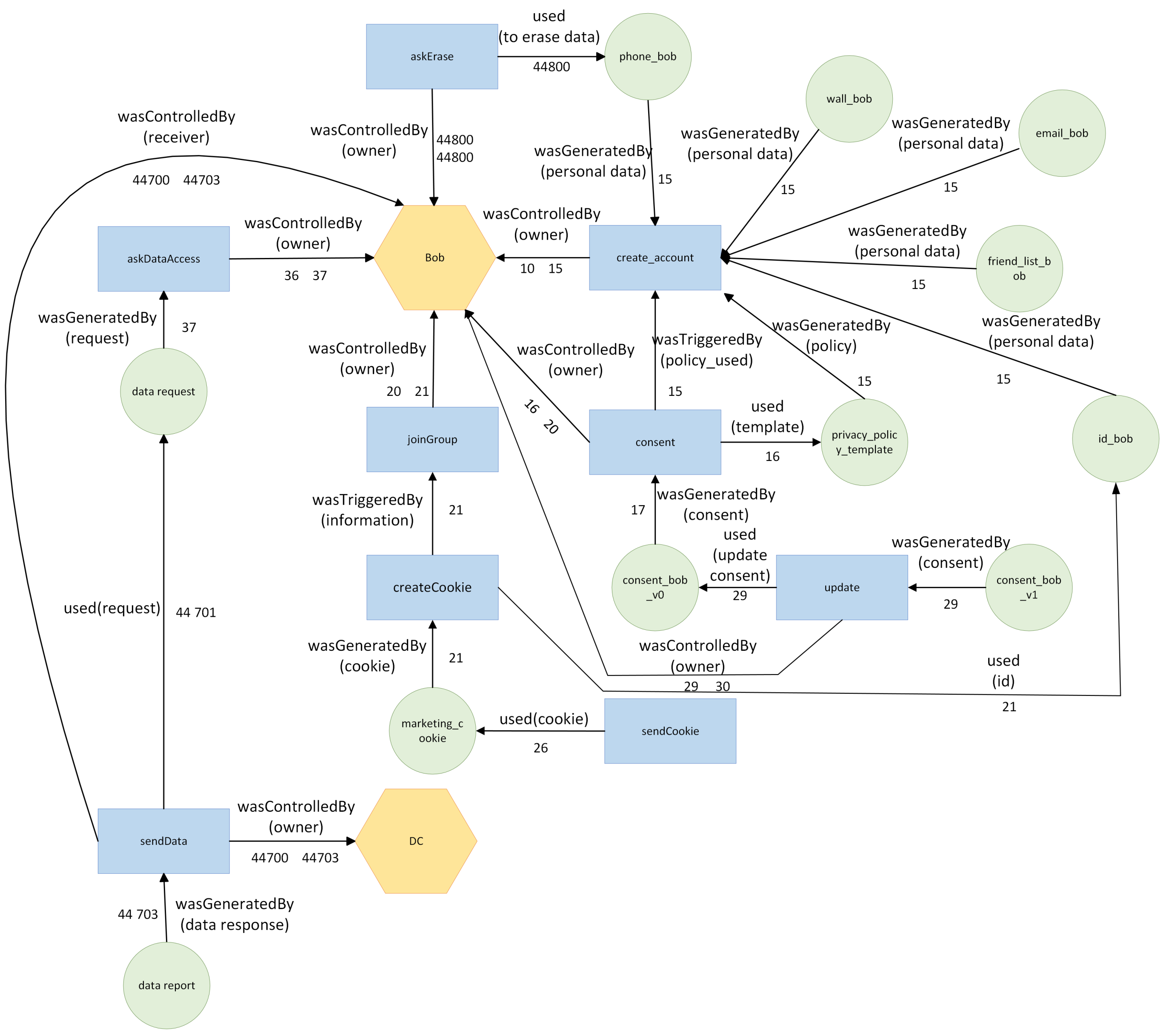

askErase
used (to erase data) 44800
phone_bob
wall_bob
email_bob
wasControlledBy (receiver) 44700 44703
wasControlledBy (owner) 44800 44800
wasGeneratedBy (personal data) 15
askDataAccess
wasControlledBy (owner) 36 37
Bob
wasControlledBy (owner) 10 15
create_account
friend_list_bob
wasGeneratedBy (request) 37
data request
wasControlledBy (owner) 20 21
wasControlledBy (owner) 16 20
wasTriggeredBy (policy_used) 15
wasGeneratedBy (policy) 15
joinGroup
consent
used (template) 16
privacy_policy_template
id_bob
wasTriggeredBy (information) 21
wasGeneratedBy (consent) 17
used (update consent) 29
consent_bob_v0
update
consent_bob_v1
createCookie
used(request) 44 701
wasGeneratedBy (cookie) 21
wasControlledBy (owner) 29 30
used (id) 21
marketing_cookie
used(cookie) 26
sendCookie
sendData
wasControlledBy (owner) 44700 44703
DC
44 703 wasGeneratedBy (data response)
data report

Figure 1. Extract from the provenance graph of an online forum.

Notice that $\text{nextConsent}(C, C1, T)$ refers to the artifact representing the update that directly follows the consent version denoted by C. It is important to note that since artifacts represent immutable objects, any modification to the original object leads to the creation of a new artifact. Consequently, for a given C, there can be only one corresponding instance of $C1$.

We can easily verify whether a consent artifact is the latest existing version, using the predicate *lastConsent* and the following pattern:

$$\text{lastConsent}(C) = \text{consent}(C, D, PU, T) \wedge \neg(\text{nextConsent}(C, C1, TU))$$

As explained earlier, $\neg(\text{nextConsent}(C, C1, TU))$ is interpreted as negation by failure. This means the pattern $\text{lastConsent}(C)$ holds true if C represents a consent and there is no pair $(C1, TU)$ such that $\text{nextConsent}(C, C1, TU)$ exists in the graph. In other words, C is considered the latest consent definition when no subsequent update artifact follows it.

Example 6 (Consent update):

In the provenance graph shown in Figure 1, Bob has updated his consent once, resulting in the creation of a new artifact, $consent\_bob\_v1$. The pattern $\text{nextConsent}(C, C1, T)$ can be instantiated with $C = consent\_bob\_v0$, $C1 = consent\_bob\_v1$, and $T = 29$. Consequently, $\text{lastConsent}(consent\_bob\_v0)$ evaluates to False, while $\text{lastConsent}(consent\_bob\_v1)$ evaluates to True.

3) Lawfulness: As stated in Article 5 of the GDPR, an artifact is considered to have been used lawfully if it was

Article	Pattern	Description
Principle: Purpose Limitation		
6.1(a)	$\mathsf{consent}(C, D, PU, T)$	consent C given for data D to be used for PU purpose at time T
7.3	$\mathsf{revoke}(C, T)$	consent C revoked at time T
7.3	$\mathsf{nextConsent}(C, C1, T)$	updated consent C into C1 at time T
7.3	$\mathsf{lastConsent}(C)$	C has no updates
Principle: Lawfullness		
-	$\neg\mathsf{isPersonal(D)}$	artifact D is not personal data and doesn't derived from personal data
5.1(a)	$\mathsf{validConsent}(D, PU, C, T, TG, TU)$	consent C for D and PU valid at time T
5.1(a)	$\mathsf{validConsent}^{+}(D, PU, C, T, TG, TU)$	consent C for personal data from which D is derived and PU valid at time T
5.1.(a)	$\mathsf{legal}(P, D, C, TG, TU)$	treatement of D by process P lawfull
Principle: Storage Limitation		
-	$\mathsf{usageInLimit}(D, P)$	process P used D during the authorized interval
-	$\mathsf{delationComplete}(D)$	D has been deleted before the end of the authorized period
-	$\mathsf{retentionValid}(D)$	authorized period to use D not expired
5.1(e)	$\mathsf{storageLimitation}(D)$	compliance of the storage limitation for D
Rigth: To Be Forgotten		
-	$\mathsf{askErase}(S, D, T)$	request from S to delete D at time T
-	$\mathsf{eraseNotDoneYet}(D)$	D not deleted yet but still in the allowed time
-	$\mathsf{erased(D)}$	D erased during authorized period
-	$\mathsf{copiesErased(D)}$	D replications erased during authorized period
17(a)	$\mathsf{eraseComplete}(D)$	compliant to the erase principle for the request on D
Right: To Access		
15.1	$\mathsf{askAccess}(U, TB, TE)$	user U asked for access at time TE
15.1	$\mathsf{rigthAccess}(U)$	U's request answered in time

TABLE II. PATTERNS EXPRESSING GDPR PRINCIPLES AND RIGHTS

processed solely for purposes to which the data subject has given valid consent. In the context of a provenance graph, this means that all paths leading from a consent artifact to a process must demonstrate that the processing was authorized.

More precisely, for each process P that uses a data artifact, the provenance path must confirm two conditions: the data subject had provided valid consent for the specific purpose at the time of processing, and this consent had not been revoked during the execution of P.

From the structure of the graph, two lawful processing scenarios can be distinguished:

- Non-personal data path: For the artefact used by P not to contain personal data, it is required that the artefact is not classified as personal data, with the attribute $dp = \mathit{False}$. it is also necessary that the artefact was derived only from data nodes annotated with the attribute $dp = \mathit{False}$. In this case, no personal data is involved, and thus no consent is required. This scenario corresponds to the path:

$$\neg\mathsf{isPersonal(D)} = (\mathsf{dp(D)} = \mathsf{False}) \wedge \\ \forall D'\big(\mathsf{wasDerivedFrom}^{+}(D, D', TS, TE) \\ \wedge\ (\mathsf{dp(D')} = \mathsf{False})\big)$$

- The artifact used by process P is personal data D, processed for a specific purpose PU, and:
  i) There exists a consent artifact C such that the data subject has given valid consent for purpose PU, and this consent was still valid (i.e., not revoked) at the time D was used by P. In this case, the data D is personal and is processed for a specific purpose PU by a process P. The processing is lawful if there exists a valid consent C that applies at the time of processing T.
  Consent is considered valid at time T if any of the following hold: consent C was given and never revoked; consent C was given and revoked, but after time T; consent C was updated after time T, meaning the version valid at T was still in force. This is expressed by the following path expression:

$$\mathsf{validConsent}(D, PU, C, T, TG, TU) = \\ \big(\mathsf{consent}(C, D, PU, TG) \wedge (TG < T) \wedge \mathsf{lastConsent}(C) \\ \wedge\neg\big(\mathsf{revoke}(C, TU)\big)\big)\vee \\ \big(\mathsf{consent}(C, D, PU, TG) \wedge (TG < T) \wedge \mathsf{lastConsent}(C) \\ \wedge\mathsf{revoke}(C, TU) \wedge (TU > T)\big)\vee \\ \big(\mathsf{consent}(C, D, PU, TG) \\ \wedge(TG < T) \wedge \mathsf{nextConsent}(C, C1, TG1) \wedge (TG1 > T)\big)$$

  ii) D was derived from other personal data artifacts D', and each of those D' has an associated valid consent for the same purpose PU at the time D is processed by P. The corresponding path expression is as follows:

$$\mathsf{validConsent}^{+}(D, PU, C, T, TG, TU) = \forall D' \\ (\mathsf{wasDerivedFrom}^{+}(D, D', TS, TE) \wedge dp(D') = \mathtt{False})\vee \\ (\mathsf{wasDerivedFrom}^{+}(D, D', TS, TE) \wedge\ dp(D') = \mathtt{True}\ \wedge \\ \mathsf{validConsent}(D', PU, C, T, TG, TU))$$

In both cases, the provenance graph provides a traceable path that allows us to verify compliance by following the data lineage and matching it with the consent.

Summarizing, the process P on an artifact D has been carried out legally if and only if the following pattern finds an instantiation in the provenance graph:

$$\neg\mathsf{isPersonal(D)} \vee (\mathsf{used(P, D, R, T)} \wedge \mathsf{action(P)} = \mathsf{PU} \\ \wedge\ \mathsf{validConsent}^{+}(D, PU, C, T, TG, TU)\ \wedge \\ dp(D) = \mathtt{True}\ \ \wedge \mathsf{validConsent}(D, PU, C, T, TG, TU))$$

Here P is the process that performed an action PU on the artifact D, C is the consent of the artifact containing the consent to perform PU, R represents a role and

$T, TG, TU, TG1$ are timestamps. This path expression is denoted $\mathsf{legal}\_DP(P, D, C, TG, TU)$. An action is lawful if consent has been given for all personal data used by P before the action is performed. If the data is not of a personal nature (first line of the expression), the action is automatically considered to comply with this principle.

Example 7 (Lawful processing): In the scenario depicted in Figure 1, a marketing cookie is created at time $T = 21$ by the process *createCookie*. This process operates under the purpose *thirdParties*, which refers to the creation of cookies intended for third-party use. To do so, it uses Bob's identity on the website, denoted *id_bob*. This processing is considered lawful if and only if at least one of the conditions defined previously is satisfied. In this case, it is the third condition (involving updated consent) that applies.

The assignment $P = \mathit{createCookie}$, $PU = \mathit{thirdParties}$, $D = \mathit{id\_bob}$, $T = 21$, $R = \mathit{id}$, $C = \mathit{consent\_bob\_v0}$, $C = \mathit{consent\_bob\_v1}$, $TG = 17$ and $TG1 = 29$ satisfies the third clause of the disjunction, since the original consent *consent_bob_v0* was given at $TG = 17$ and updated by *consent_bob_v1* at $TG1 = 29$, which occurs after the data usage at $T = 21$. As a result, the consent was still valid when the data was used, and the creation of the cookie is therefore lawful.

4) Storage limitation: The retention of personal data is governed by Article 5.1(e) of the GDPR, which states that personal data should not be kept longer than necessary for the purposes for which it is processed, except in specific cases such as archiving in the public interest.

In practice, this means that personal data should not be stored without use for a period exceeding the retention limit defined by the system, based on its operational requirements.

A system respects this storage limitation principle for a given piece of personal data D if the data has not remained unused beyond the authorized retention duration. More specifically, there must be no interval between two consecutive uses of D that exceeds the maximum permitted retention time ($TLIMIT$).

This condition can be expressed by the following pattern:

$$\begin{aligned}&\mathsf{usageInLimit}(D, P) = \mathsf{used}(P, D, R, T)\ \wedge\\&((\mathsf{used}(P', D, R', T')\ \wedge (T' > T) \wedge (T' - T < TLIMIT))\\&\vee \neg(\mathsf{used}(P', D, R', T') \wedge (T' > T)))\end{aligned}$$

The negation here is interpreted again using negation by failure. That is, for a process P that uses D, either there exists another process P' that also uses D within a time interval not exceeding the retention period, or P is the last process to have used D. The verification of whether the retention period is still valid after the last use is performed by the predicate. $\mathsf{retentionValid}$.

Between the last use of the personal data artifact D and the current time, two conditions can ensure compliance with the GDPR's storage limitation principle (Article 5.1(e)): either the data has been explicitly deleted, i.e., there exists a deletion event $\mathsf{notAvailable}(D, T)$ recorded in the provenance graph; or the data is still retained, but the defined retention period has not yet been exceeded, meaning the condition $TCURRENT - TU < TLIMIT$ holds, where TU is the last time the data was used.

The path expression confirming that the data was previously deleted is as follows.

$$\begin{aligned}&\mathsf{delationComplete}(D) = \mathsf{used}(P, D, R, TU)\wedge\\&\mathsf{notAvailable}(D, T) \wedge (T - TU <= TLIMIT)\end{aligned}$$

In the case that the data has not been deleted yet and the retention period has not been exceeded, the following path expression will have a match in the graph.

$$\begin{aligned}&\mathsf{retentionValid}(D) =\\&\mathsf{used}(P, D, R, TU) \wedge \neg(\mathsf{notAvailable}(D, T))\\&\wedge(TCURRENT - TU <= TLIMIT)\end{aligned}$$

In both cases, there is no need to verify that P is the last process to have used D. If the path exists for some process P, the time condition also holds for the actual last process P', since P' was executed more recently than P. This ensures that the data is either used regularly within the permitted retention period or deleted once that period has elapsed, thereby guaranteeing compliance with the storage limitation principle.

A system is compliant with the storage limitation principle for a personal data artifact D, if the following pattern has an instantiation:

$$\begin{aligned}&\mathsf{storageLimitation}(D) = \forall P(\mathsf{usageInLimit}(D, P)\wedge\\&(\mathsf{delationComplete}(D) \vee \mathsf{retentionValid}(D)))\end{aligned}$$

This ensures that personal data is never retained passively beyond its legal retention limit.

Example 8 (Limitation of personal data storage): Suppose that the forum shown in Figure 1 has a data retention period of 5 years. Bob has not logged into his account for 7 years. His email address and all his personal data have not been used since his last login. This situation is represented in the graph by the path $\mathsf{used}(P, \mathit{email\_bob}, R, TU) \wedge \neg(\mathsf{used}(P', \mathit{email\_bob}, R', TU') \wedge (TU < TU'))$ If the system is compliant, we would find the dependency $\mathsf{notAvailable}(\mathit{email\_bob}, T)$ with $T \in [TU, TU + TLIMIT]$. The sub-pattern $\mathsf{delationComplete}(\mathit{email\_bob})$ evaluates to True.

In the case where the system fails to delete the data, the pattern $\mathsf{delationComplete}(\mathit{emai\_bob})$ would evaluate to false, as would the pattern $\mathsf{retentionValid}(\mathit{email\_bob})$, because the retention period has been exceeded.

It is important to emphasize that these patterns yield a Boolean result depending on whether a matching instantiation exists in the provenance graph. However, this result must be interpreted with care: there is a difference between actual compliance, meaning that the processing is and will remain compliant over time, and the absence of non-compliance simply because a retention deadline has not yet been reached.

In the latter case, the pattern does not confirm lasting compliance, but rather the current absence of a violation. Therefore, an a posteriori check may be required.

For instance, if the verification in the previous example were performed three years after the last use of Alice's email, the pattern would return true, indicating compliance at that moment. However, it would still be necessary to later verify that the email was either deleted before the retention limit expired, or reused within the allowed time frame.

B. User rights

Just as certain articles related to data protection principles can be automatically verified through pattern-based approaches, the same applies to some rights, or at least to specific provisions within the articles that define them. In this section, we illustrate this with two key rights: the right to be forgotten and the right of access.

As with the principles, pattern-based detection may only identify a single instance of non-compliance, necessitating further verification over time. Moreover, some articles require additional manual checks, especially when they impose conditions on the content to be provided to individuals, or when they involve requirements concerning the clarity, granularity, or comprehensibility of the explanations and notifications to be delivered.

a) Right to be forgotten: In accordance with Article 17a of the GDPR, data subjects have the right to request the erasure of certain personal data, and data controllers are obliged to comply with such requests within a specified time frame.

To verify whether the right to erasure (Article 17a of the GDPR) has been correctly exercised, we first identify a relevant pattern in the provenance graph. Specifically, we look for a process P performing the action *askErase* on a data item D under the control of a user U identified as the data owner. This can be captured by the following path in the graph $\mathsf{used}(P, D, R, T) \wedge \mathsf{wasControlledBy}(P, U, "owner", TB, TE)$ where T, TB, TE are timestamps, and the action associated with process P is *askErase*. We define the $askErase$ pattern to formalize this detection as follows:

$$\begin{aligned}&\mathsf{askErase}(S, D, T) = \mathsf{owns}(S, D, TU) \wedge \mathsf{used}(P, D, R, T) \wedge \\ &\mathsf{wasControlledBy}(P, S, R', TB, TE) \wedge \\ &\mathsf{action}(P, "askErase") \wedge \\ &\neg\big(\mathsf{used}(P', D, R, T') \wedge \mathsf{action}(P', "askErase") \wedge (T' < T)\big)\end{aligned}$$

This pattern identifies the earliest request for deletion of data D, ensuring that we retrieve the first occurrence of such a request in time, i.e there exists no prior instance of a process P' asking for the erasure of the data. Since a user may issue multiple deletion requests before receiving a response, capturing only the initial request is essential to assess whether the controller met the deadline and thus complied with the regulation.

A system is considered compliant if and only if the requested data and all its copies are deleted within the authorized time frame. The deletion of a data item D, following a user's request, is captured by the pattern

$$\begin{aligned}&Erased(D) = \mathsf{askErase}(S, D, T) \wedge \\ &\mathsf{notAvailable}(D, TU) \wedge (TU - T < TLIMIT)\end{aligned}$$

with $TLIMIT$ being the deadline for performing the action in a compliant manner. Here $\mathsf{askErase}(D, T)$ denotes that the first request for erasure of D occurred at time T, $\mathsf{notAvailable}(D, TU)$ indicates that D became unavailable (i.e., deleted) at time TU, $TLIMIT$ is the regulatory deadline for performing the deletion.

To ensure compliance, it is not sufficient to delete only the original data item; all copies of that data must also be deleted. The deletion of derived or copied data D, resulting from a copy operation on D, is expressed by the following condition:

$$\begin{aligned}&copiesErased(D) = \mathsf{askErase}(D, T) \wedge \\ &\forall D'(\mathsf{wasDerivedFrom}^{+}(D, D', "copy", TU', TG) \wedge \\ &\mathsf{notAvailable}(D', TU') \wedge (TU' - T < TLIMIT))\end{aligned}$$

This states that for all data artifacts D' derived from D via one or more copy operations (denoted by $\mathsf{wasDerivedFrom}^{+}$), D' must also be made unavailable (i.e., deleted) within the same time constraint relative to the original request time T.

We can define full compliance for a data item D with respect to Article $17a$ using the following predicate:

$$\begin{aligned}&\mathsf{eraseComplete}(D) = \\ &Erased(D) \wedge copiesErased(D)\end{aligned}$$

This formalization ensures that both the original data and all its copies are deleted in a timely manner, satisfying the obligations imposed by Article 17a.

However, as previously mentioned, a compliance check can be performed at any time. This implies that there may be cases where a deletion request has been submitted, but the corresponding data has not yet been deleted, while still being within the allowed deadline.

We define the case of a pending deletion request using the pattern :

$$\begin{aligned}&\mathsf{eraseNotDoneYet}(D) = \\ &\mathsf{askErase}(S, D, T) \wedge \neg(\mathsf{notAvailable}(D, TU)) \wedge \\ &(TCURRENT - T < TLIMIT)\end{aligned}$$

where $TCURRENT$ represents the date on which the verification is carried out. $\mathsf{askErase}(S, D, T)$ indicates that a deletion request for data D was made at time T by the owner S; $\neg\mathsf{notAvailable}(D, TU)$ indicates that the data D is still available at time $TCURRENT$; $TLIMIT$ is the legal deadline to complete the deletion.

This pattern captures the intermediate state where the data has not yet been deleted, but the deadline has not been exceeded (hence, the system is not yet non-compliant).

Finally, to verify whether the system is globally compliant with respect to the deletion request for a personal data item D, we define the following pattern:

$$\begin{aligned}&\mathsf{eraseCompliant}(D) = \\ &(\mathsf{askErase}(S, D, T) \wedge \mathsf{eraseComplete}(D)) \vee \\ &\mathsf{askErase}(S, D, T) \wedge \mathsf{eraseNotDoneYet}(D)\end{aligned}$$

This expression states that compliance is satisfied either because the data and all its derived copies have been deleted within the deadline ($\mathsf{eraseComplete}(D)$), or because the request has been issued but the allowed deadline has not yet expired ($\mathsf{eraseNotDoneYet}(D)$).

Therefore, our framework allows us to distinguish between full compliance, ongoing compliance (pending fulfillment), and non-compliance (which occurs when the deadline has passed without deletion).

Example 9 (Right to be forgotten): We consider a scenario derived from the system shown in Figure 1. According to Article 17a of the GDPR, a data controller must delete a data subject's personal data within a specified period after receiving a valid erasure request. Suppose that this deadline is set to 30 days. Given that timestamps are expressed in minutes, this results in a deadline of $TLIMIT = 30 \times 24 \times 60 = 43.200$.

Bob, after receiving his personal data report from the forum, submits a request to delete his phone number $phone\_bob$. The provenance graph records this request at timestamp $T = 44.800$.

At the time of the compliance verification, 12 days have passed since Bob sent the request, i.e., $TCURRENT = 61.983$. We compute the elapsed time: $TCURRENT - T = 61.983 - 44.800 = 17.183 < TLIMIT$. This means that the request has been made and is still within the allowable period for compliance. As a result, the following instantiation holds: $\mathsf{eraseCompliant}(phone\_bob) = \mathsf{askErase}("bob", phone\_bob, 44800) \wedge \mathsf{eraseNotDoneYet}(phone\_bob)$.

The system is therefore considered compliant at this stage, even though the deletion has not yet been executed. However, a follow-up check must be performed after the deadline to ensure that the data, and all its copies, were actually erased in time.

b) Right of access: The GDPR grants data subjects the right to access all personal data held about them and to understand how their information has been processed. Article 15.1 specifies the obligations of the data controller in responding to such requests. While some aspects of this compliance check can be automated, it cannot be fully verified through event logs alone, as these do not capture the content of the response document. In this context, the defined predicates can verify that a response was issued following a request, but they do not guarantee that the response includes all the required information.

As with the right to be forgotten, an access request has been made if the following path is present in the graph:

$$\begin{aligned}
&\mathsf{askAccess}(U, TB, TE) = \\
&\mathsf{wasControlledBy}(P, U, "owner", TB, TE) \wedge \\
&\mathsf{action}(P, "askAccess") \wedge used(P, D, R, TU) \wedge \\
&dp(D) = True \wedge \mathsf{owns}(U, D, TU') \wedge \\
&\neg\big(\mathsf{wasControlledBy}(P', U, "owner", TB', TE') \wedge \\
&\mathsf{action}(P', "askAccess") \wedge (TE' < TE)\big)
\end{aligned}$$

The pattern $\mathsf{askAccess}(U, TE)$ retrieves the timestamp TE corresponding to the first access request initiated by the data subject U.

The system is considered compliant with the right of access if the data controller S' provides a response to the request within a legally defined time limit $TLIMIT$. Compliance can be verified if at least one of the two disjunctive conditions in the following pattern is instantiated in the provenance graph:

$$\begin{aligned}
&\mathsf{rigthAccess}(U) = \big(\mathsf{askAccess}(U, TB, TE) \\
&\wedge \mathsf{wasControlledBy}(P', S', R, TB', TE') \\
&\wedge \mathsf{action}(P', "sendData") \wedge (TE' - TE < TLIMIT)\big) \vee \\
&\big(\mathsf{askAccess}(U, TB, TE) \wedge \\
&(TCURRENT - TE < TLIMIT)\big)
\end{aligned}$$

Here, P and P' denote processes, and TB, TB', TE, and TE' represent timestamps. The first disjunct checks whether the response has already been sent in time, while the second allows for the possibility that the request has been made but the deadline has not yet passed.

The example that follows illustrates its application to the scenario depicted in Figure 1.

Example 10 (Right of access): Suppose that, in the case of the online forum, the data controller has a legal obligation to respond to a data access request within 30 days, which corresponds to a time limit of $TLIMIT = 43.200$ minutes.

According to the provenance graph, Bob submitted a data access request labeled $data\_request$ at timestamp $TE = 37$, and later received a response labeled $data\_report$ at timestamp $TE' = 44.730$. This sequence satisfies the structural requirements of the first conjunction in the pattern $\mathsf{rigthAccess}(U)$, as it includes the expected control and action events.

However, the time constraint is not satisfied: the response was provided $44.730 - 37 = 44.693$ minutes (approximately 32 days) after the initial request, which exceeds the allowed time limit of 30 days. Therefore, the system fails to comply with Article 15 of the GDPR in this instance and Bob's right of access was not respected.

VI. Compliance verification

In this section, we present a prototype tool designed to assist in the verification of GDPR compliance. In this work, we consider provenance graphs as input. We do not address the problem of constructing provenance graphs from traces or event logs. In practice, this task is non-trivial, as logs differ widely in structure, semantics, and level of detail, and therefore require context-specific parsing and mapping choices. Developing such extraction pipelines is outside the scope of the present work and constitutes an interesting direction for future research. We focus here on the verification aspects of the framework. The tool allows users to select specific principles and rights to examine, offering a flexible and targeted approach to auditing. In addition to verifying compliance, the tool provides informative feedback by flagging situations that may warrant further investigation or future monitoring. To support a more thorough audit process, it also extracts contextual data linked to potential non-compliance, enabling auditors to better understand the origin and scope of detected issues. This facilitates not only initial compliance checks but also ongoing assessments and documentation.

A. Methodology

To demonstrate and evaluate our approach, we rely on the case study representing a simplified online forum system of Section IV-C. This system is modeled as a provenance graph, where nodes represent data, users, system processes, and consent artifacts, and edges capture data flows and dependencies.

Our methodology proceeds as follows:

- For each relevant GDPR principle or data subject right (e.g., purpose limitation, consent, right of access, right to erasure), we define one or more formal compliance patterns. These patterns express the structural and temporal requirements that must be satisfied within the provenance graph. A summary of the patterns associated with each principle is given in Table II.
- We then encode both the provenance graph and the compliance patterns in a logic programming language (Prolog). The provenance information is translated into a set of facts, while the compliance patterns are encoded as rules using the templates defined in Section V.
- Our verification tool uses a Prolog solver to check whether the patterns are satisfied by the graph. If a pattern is not satisfied, the system identifies the precise point of failure and, when possible, provides suggestions for further analysis or actions needed to reach compliance.
- This approach supports both automated compliance verification and semi-automated auditing. For example, in the forum case study, the system can verify whether Bob's updated consent ($consent\_bob\_v1$) is correctly enforced in subsequent data usage, or whether his request for erasure has been processed within a reasonable timeframe.

This logic-based methodology enables precise, explainable, and extensible analysis of compliance with GDPR requirements, using provenance data as evidence.

B. Prototype Architecture

We have developed a Java-based auditing tool that verifies GDPR compliance by encoding the formal patterns from the previous section in Prolog [25]. The tool integrates a reasoning engine to analyze data processing workflows and identify potential violations. Our approach allows us to benefit from the efficient reasoning capabilities provided by Prolog solvers for path condition resolution rather than relying on an ad-hoc algorithm. In particular, we implemented rules for resolving path queries based on the obtained provenance graph.

The tool reports non-conformities, along with warnings that suggest further verifications when the analysis cannot be concluded at current time. Auditors can configure the analysis by selecting specific GDPR principles, relevant data types, and processing steps, enabling fine-grained and targeted audits. A preliminary version of this tool was introduced in [1].

The main components of the prototype are illustrated in Figure 2. Through the interface, the auditor selects the system to audit and specifies the GDPR principles, personal data, or data subjects to be analyzed. These inputs are translated into Prolog queries by a dedicated translator module. The solver then executes the queries over the provenance graph representing the system's data flows and returns the results to the interface. The output includes detailed information about any detected non-conformities.

We describe next the architecture of our prototype, explaining how its components interact and how the patterns have been adapted to deliver precise feedback to the auditor.

a) Interface: The interface, developed in Java using the JavaFX graphics library, allows auditors to configure the scope of the audit. They can select the system to analyze and optionally restrict the verification to specific processes, personal data, or data subjects whose rights should be checked. The system's activity must be provided as a provenance graph, encoded as a Prolog file, which is loaded through the interface.

The auditor can further refine the audit by choosing the GDPR principles to be verified or focusing on specific components rather than conducting a full-system analysis (which is the default behavior). These configuration options are sent to the translator module (see Figure 2), which generates the corresponding Prolog queries. The available audit options are presented through a menu, as shown in Figure 3.

The results returned by the solver are finally displayed to the auditor in textual form (Figure 4).

b) Translator: The translator module serves two main purposes. First, it converts the auditor's selections into parameterized Prolog queries. Second, it extracts relevant elements (such as personal data, data subjects, and processes) from the provenance graph. These extracted entities are used both to populate the interface with appropriate choices and to instantiate variables in the generated queries.

Based on the selected principles and nodes, the translator generates a list of Prolog queries and sends them, along with the system data, to the Prolog solver using the JPL library. For instance, in the case study shown in Figure 1, if the auditor wants to verify all GDPR principles related to Bob's email address, namely the lawfulness of processing, the right to erasure, and storage limitation, the translator produces the following queries:

$$\begin{array}{l}\mathsf{legal}(P, \text{"}\texttt{email\_bob}\text{"}, C, TG, T).\\ \mathsf{eraseComplete}(\text{"}\texttt{email\_bob}\text{"}).\\ \mathsf{storageLimitation}(\text{"}\texttt{email\_bob}\text{"}).\end{array}$$

In the case of the principle of lawfulness, the verification applies to all processes P involving Bob's email. The translator therefore also sends a list of relevant processes to the solver, enabling it to instantiate the variable P during query evaluation.

c) Reasoning module: The reasoning module integrates the Prolog solver, which is responsible for verifying path queries on the provenance graph. It also includes the formal definitions of GDPR predicates, each encoding a principle or a right, as introduced in the previous section.

When a path query is received from the translator, the reasoning module constructs the corresponding Prolog goal and invokes the solver to evaluate it. The solver applies the deduction rules (i.e., the Prolog program encoding causal dependencies and compliance patterns) to prove or refute the query.

Each GDPR principle is implemented as a Prolog rule, systematically decomposed into conjunctions and disjunctions of sub-predicates to enable granular compliance checking and

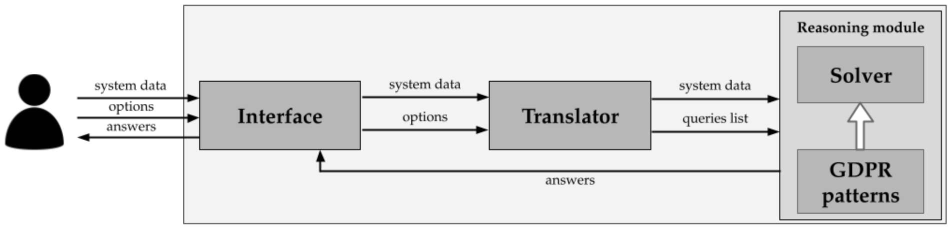
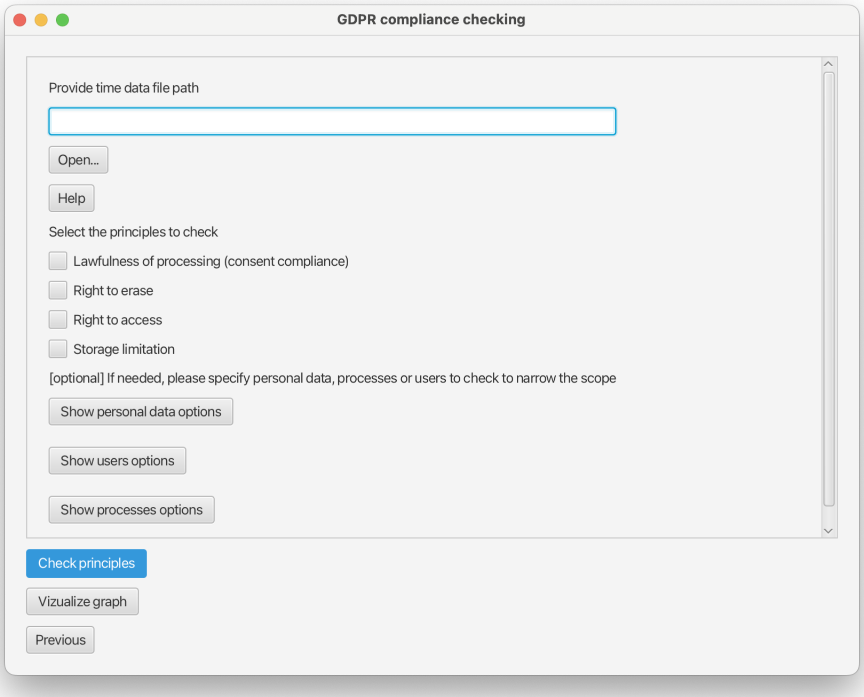


Figure 2. Tool Architecture



Provide time data file path

Select the principles to check

Lawfulness of processing (consent compliance)

Right to erase

Right to access

Storage limitation

[optional] If needed, please specify personal data, processes or users to check to narrow the scope

Figure 3. Prototype interface

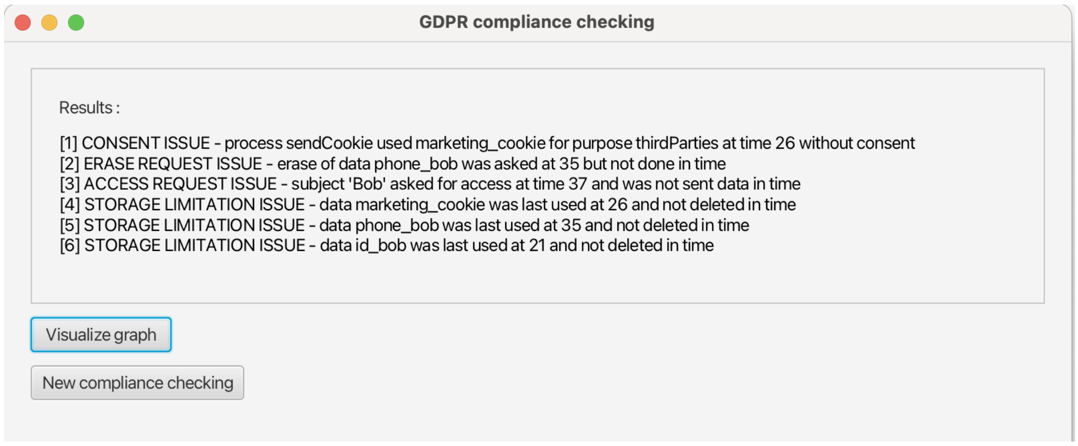
more informative feedback to the auditor. These rules typically follow the structure:

```
predicate(parameters) :-
( parameter check,
( compliance check; (deadline check,
 notify future verification);
(\+ compliance check, \+ deadline check,
notify non-compliance) ));
(\+ parameter check,
 notify missing parameters).
```

where \+ denotes negation. This structure ensures that, for each query, the solver can determine not only whether a principle is satisfied, but also whether additional verification is needed, or whether a compliance failure should be reported.

The patterns introduced in Section V are decomposed into Prolog sub-rules to provide interpretable feedback to the auditor. Each GDPR principle or right is defined through a main predicate composed of several sub-predicates, each serving a specific verification function:

- `check parameters`: This sub-predicate restricts verification to the relevant nodes in the graph. For instance, the principle of lawfulness only applies to personal data. A rule like `used(P,D,R,T), action(P) = PU, isPersonal(D)` ensures that the predicate `legal(P,D,C,TG,T)` is only applied if D is personal data (i.e., `dp(D) = True`) and if P is a process under audit that actually uses D.
- `verification compliance` and `verification deadline`: These sub-predicates form the core of the compliance patterns. The first verifies whether the required conditions are satisfied (e.g., whether a request has been processed), while the second checks whether the system is still within the allowed timeframe for fulfilling a request. If the action has not yet occurred but the deadline has not passed, the `display future check` predicate is



Results :

[1] CONSENT ISSUE - process sendCookie used marketing_cookie for purpose thirdParties at time 26 without consent
[2] ERASE REQUEST ISSUE - erase of data phone_bob was asked at 35 but not done in time
[3] ACCESS REQUEST ISSUE - subject 'Bob' asked for access at time 37 and was not sent data in time
[4] STORAGE LIMITATION ISSUE - data marketing_cookie was last used at 26 and not deleted in time
[5] STORAGE LIMITATION ISSUE - data phone_bob was last used at 35 and not deleted in time
[6] STORAGE LIMITATION ISSUE - data id_bob was last used at 21 and not deleted in time

Figure 4. Compliance results display

triggered to inform the auditor that further verification will be needed later. Not all principles include a deadline check (e.g., lawfulness does not), but rights like access and erasure do.

- `display non-compliant data`: This sub-predicate collects and returns the relevant information when non-compliance is detected, such as the involved data, process, and violated principle or right. It also causes the main predicate to fail (i.e., return `false`), signaling a violation.
- `display no parameters`: This sub-predicate is used when the selected nodes are not subject to the principle being checked. It provides feedback indicating that the rule does not apply in this context.

This modular structure enables precise compliance verification and interpretable output, which helps auditors understand and act on the results more easily.

The full predicate first verifies whether the instantiated parameters are valid. If they are not applicable, it immediately proceeds to the final step, providing feedback through the `display no parameters` predicate.

When the parameters are valid, the predicate returns *true* only if either the `compliance check` or the `deadline check` (which also includes relevant information) succeed. If neither check passes, the predicate triggers `display non-compliant data` to present detailed information about the violation and returns *false*.

The results are sent to the interface for displaying, including information about the personal data, time and processes involved in the possible violation.

VII. Experimental Validation

To validate our approach, we first tested the correction of the GDPR compliance patterns on small provenance graphs corresponding to the scenarios discussed in Section IV-C. For these initial experiments, we systematically modified key parameters, such as the purposes attached to consent, the timestamps of consent creation, or the timing of actions like data deletion or disclosure, in order to verify that the solver correctly reported all cases of non-compliance. Once the patterns were validated on these controlled examples, we evaluated the performance of the prototype on larger graphs produced by the synthetic graph generator described next.

The results show that the tool maintains acceptable performance for offline audits of graphs of moderate size. Patterns addressing lawfulness, storage limitation, the right to be forgotten, and the right of access are all verified within a few seconds even on the larger generated graphs. However, checks involving the consent principle require noticeably more time: as the graph grows, verifying the complex relationships between personal data, purposes, and consent updates can take significantly longer. Improving the efficiency of these consent-related verifications will therefore be an important focus of future optimization efforts.

A. Graph generator

In order to generate provenance graphs that reflect realistic scenarios, we developed a generator of provenance graphs. Our approach relies on the assembly of smaller subgraphs, referred to as *bricks*, each representing the execution of a process within the system. These bricks can be either general (e.g., accessing a webpage) or specific to a particular context (e.g., purchasing a product).

Each context is composed of 3 to 5 specific bricks, in addition to a set of 10 general-purpose bricks that are common across contexts. Some bricks are *triggered* bricks, meaning they are conditionally activated by certain actions. To capture this dependency, each standard brick capable of triggering another includes a reference, in its facts, to the corresponding triggered brick (subgraph).

Figure 5 illustrates this mechanism: the general brick sendMail, which defines the subgraph for sending an email, embeds a reference to the triggered brick sendAnalysisCookie, which represents the action of sending an analysis cookie. The variables marked as %VAR% are instantiated during graph generation. The keyword CAN indicates that the triggered brick is optional and will be included based on a random probability between 0.1 and 1.

```
wasControlledBy('%PROCESS:sendMail%','DC', 'owner', %T%, %TC%).
wasGeneratedBy('message','%PROCESS:sendMail%', 'mail to send', %T%).
used('%PROCESS:sendMail%','%DATA:mail%', 'address to send to', %T%).
%CAN:/TRIGGERED/sendAnalysisCookie%
used('%PROCESS:sendMail%', 'message', 'mail sent', %TF%).
```

Figure 5. sendMail brick

To determine the most effective parameters for data storage and retrieval, we considered three distinct usage contexts: a forum or social networking website, an online store, and a public service portal (e.g., a water management platform). For each context, we defined the types of data it may contain (such as email addresses or analytic cookies) as well as the corresponding building blocks (or bricks). Each brick specifies whether it can be reused for the same user. For example, while a user can create an account only once, they may update their information multiple times.

The graph generator takes four parameters, the first of which is mandatory: the name of the output file containing the generated graph encoded as Prolog facts. The other three parameters are optional: the number of users (or a file listing the user names), the desired context, and a repetition *factor*. The factor, i.e., an integer greater than or equal to 1, controls the repetition of reusable bricks, thereby increasing the graph's size. A factor of 1 prevents repetition, while higher values allow bricks to be reused more frequently. If the parameters are not provided, the generator randomly selects values and assigns a default factor of 1.

Bricks are randomly selected, instantiated, and incorporated into the graph. When a triggered brick is added, the generator ensures that all necessary sub-elements for executing the associated action are also included.

Figure 6 illustrates a graph generated with 20 users and a repetition factor of 1, within the public service context. The graph is visualized using the Neo4J tool.

B. Generated graphs results

First, to estimate verification times, we performed preliminary tests and imposed a maximum duration of four hours per process. For graphs containing up to 200 users, all GDPR principles except *Lawfulness* were verified well within this limit: the principle *Right of access*, for instance, was verified in 4s and the principle *Storage limitation* was verified in 20s. For the *Lawfulness* principle, however, the solver did not consistently complete within the allotted time once graphs reached this scale.

To keep the evaluation consistent and reproducible, we subsequently limited the verification time to five minutes per graph. Under this setting, even graphs of moderate size, such as the example in Figure 6, were fully checked in 18 seconds. Our detailed experiments therefore focus on graphs of up to 50 users, with repetition factors from 1 to 5 across the three defined contexts, which already provide a representative basis for assessing performance.

Our experiments confirmed that the verification time increases with the complexity of the graph, particularly in terms of the number of dependencies. This complexity is influenced both by the repetition factor and the nature of the context-specific bricks, which often introduce additional dependencies. As a result, graphs with larger file sizes tend to require more time for analysis. For example, graphs between 85 and 90 kilobytes (KB) in size were verified in an average of 17 seconds. This time increased to 41 seconds for graphs between 100 and 110 KB, and up to 1.5 minutes for those ranging from 110 to 125 KB.

However, file size alone does not fully determine verification time (see Figure 8). We observed significant variability in execution times for graphs of similar size. In particular, contexts with a high volume of personal data tend to incur longer verification times. This is largely due to the repeated evaluation of the `isPersonal` predicate, which plays a key role in determining whether specific compliance rules are applicable. The more frequently this predicate needs to be resolved, the longer the verification process takes. Indeed, by checking each principle individually, we observe that the time required to verify the principle of *Lawfulness* (and therefore the `validConsent` predicate that uses `isPersonal`) is 100 times higher than for the other principles (see Figure 7). For the graph in Figure 6, the verification time for the *Lawfulness* principle is 17s, whereas the verification of the *Right of access* principle takes only 3ms.

While our experiments did not cover very large provenance graphs, the framework supports a modular verification strategy. The interface allows selecting specific principles, processes, data types, or individual users, enabling the analysis of per-user (or per-principle) graphs. Such a decomposition is, in principle, more scalable and is naturally supported by the tool's interface. All the experiments were performed on a MacBook Pro equipped with an Apple M2 chip and 16 GB of RAM.

VIII. Conclusion

In this paper, we propose a modeling of the core principles and rights of the GDPR, based on the provenance model. Our representation captures these principles and rights by combining conjunctions and disjunctions of causal dependencies from the OPM, temporal constraints on timestamps, and conditions on node attributes.

These compliance patterns are flexible and can be extended to cover additional GDPR articles or other data protection regulations. For instance, the Health Insurance Portability and Accountability Act (HIPAA) imposes retention periods for medical records, a requirement that can be expressed similarly

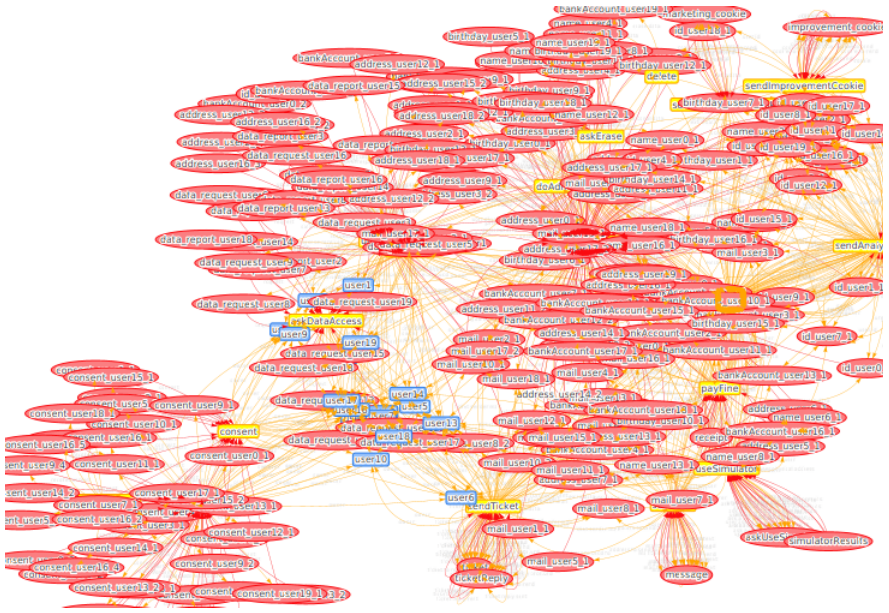


Figure 6. Neo4j representation of a provenance graph extract generated by the generator with 20 users and factor 1

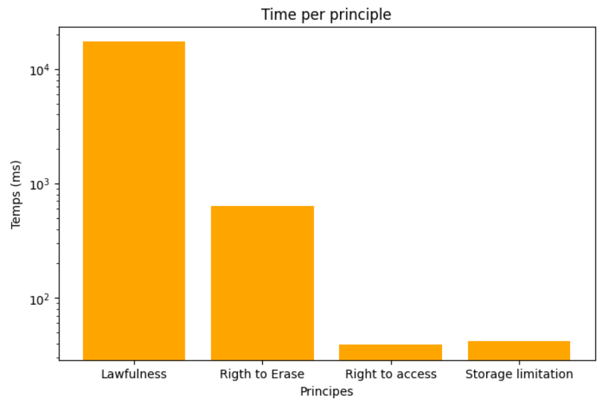


Figure 7. Time per principle for a graph of 20 users

to the right to be forgotten using path expressions with temporal constraints. Likewise, the structural similarity between GDPR and GDPR-UK articles allows our patterns to be easily adapted to ensure compliance with the latter regulation.

We validate our approach through a prototype tool that implements compliance patterns based on causal dependency paths, allowing partial automation of GDPR compliance verification using a Prolog solver. To demonstrate the feasibility of our approach, we conducted experiments on both small- and large-scale scenarios using a graph generator. This generator produces provenance graphs that can incorporate or exclude non-conformities, reflecting actions specific to each type of system. Increasing the variety of building blocks in the graph generator would allow more actions to be represented, thereby expanding the applicability of our approach.

Future work will focus on extending this approach to leverage real system logs or system traces for provenance graph generation, enabling the analysis of real-world scenarios beyond synthetic ones.

REFERENCES

[1] P. Di Salvo-Cilia, A. Martinez Anton, and C. Bertolissi, “Towards automated checking of gdpr compliance”, in *Proceedings of the CYBER 2024 Conference*, Extended abstract., IARIA, 2024.

[2] M. Miri, F. H Foomany, and N. Mohammed, “Isaca: Complying with gdpr, an agile case study”, *ISACA Journal*, vol. 2, Apr. 2018.

[3] J. Mohan, M. Wasserman, and V. Chidambaram, “Analyzing GDPR Compliance Through the Lens of Privacy Policy”, *arXiv e-prints*, arXiv:1906.12038, Jun. 2019, Art. no. arXiv:1906.12038. DOI: 10.48550/arXiv.1906.12038. arXiv: 1906.12038.

[4] A. Xiang, W. Pei, and C. Yue, “Policychecker: Analyzing the gdpr completeness of mobile apps’ privacy policies”, in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’23, Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 3373–3387, ISBN: 9798400700507. DOI: 10.1145/3576915.3623067.

[5] S. Chowdhury, L. Coles-Kemp, and G. Kambourakis, “Toward automated gdpr compliance checking”, *Computers & Security*, vol. 100, p. 102 089, 2021. DOI: 10.1016/j.cose.2020.102089.

[6] E. Arfelt, D. Basin, and S. Debois, “Monitoring the gdpr”, in *Computer Security – ESORICS 2019*, Cham: Springer International Publishing, 2019, pp. 681–699, ISBN: 978-3-030-29959-0.

[7] M. Hashem Eiza, V. Thong Ta, Q. Shi, and Y. Cao, “Secure semi-automated gdpr compliance service with restrictive fine-grained access control”, *Security and Privacy*, vol. 7, no. 6, Aug. 2024. DOI: 10.1002/spy2.451.

[8] B. Esteves and V. Rodríguez-Doncel, “Analysis of ontologies and policy languages to represent information flows in gdpr”, *Semantic Web*, vol. 15, no. 3, pp. 709–743, May 2024. DOI: 10.3233/SW-223009.

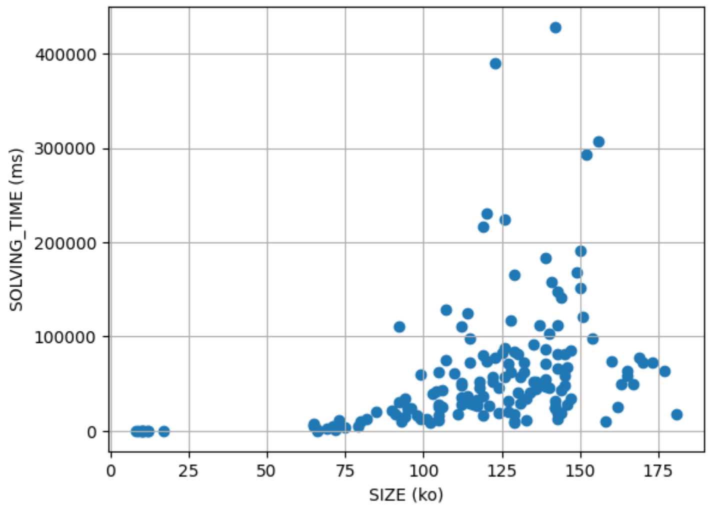


Figure 8. Dependency between graph size and resolution time.

[9] T. Athan, G. Governatori, M. Palmirani, A. Paschke, and A. Wyner, “Legalruleml: Design principles and foundations”, in *Reasoning Web. Web Logic Rules 2015*, ser. Lecture Notes in Computer Science, vol. 9203, Heidelberg: Springer, 2015, pp. 151–188.

[10] C. Bartolini, G. Lenzini, and C. Santos, “A legal validation of a formal representation of articles of the gdpr”, in *TERECOM@JURIX*, 2018.

[11] A. Tauqeer, A. Kurteva, T. R. Chhetri, A. Ahmeti, and A. Fensel, “Automated gdpr contract compliance verification using knowledge graphs”, *Information*, vol. 13, no. 10, 2022, ISSN: 2078-2489. DOI: 10.3390/info13100447.

[12] T. R. Chhetri et al., “Data protection by design tool for automated gdpr compliance verification based on semantically modeled informed consent”, *Sensors*, vol. 22, no. 7, 2022, ISSN: 1424-8220. DOI: 10.3390/s22072763.

[13] S. Agostinelli, F. M. Maggi, A. Marrella, and F. Sapio, “Achieving gdpr compliance of bpmn process models”, in *Information Systems Engineering in Responsible Information Systems*, C. Cappiello and M. Ruiz, Eds., Cham: Springer International Publishing, 2019, pp. 10–22, ISBN: 978-3-030-21297-1.

[14] M. Barati, O. Rana, I. Petri, and G. Theodorakopoulos, “Gdpr compliance verification in internet of things”, *IEEE Access*, vol. 8, pp. 119 697–119 709, 2020. DOI: 10.1109/ACCESS.2020.3005509.

[15] D. Basin, S. Debois, and T. Hildebrandt, “On purpose and by necessity: Compliance under the gdpr”, in *Financial Cryptography and Data Security*, S. Meiklejohn and K. Sako, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 20–37, ISBN: 978-3-662-58387-6.

[16] M. Saltarella, G. Desolda, and R. Lanzilotti, “Privacy design strategies and the gdpr: A systematic literature review”, in *HCI for Cybersecurity, Privacy and Trust: Third International Conference, HCI-CPT 2021, Held as Part of the 23rd HCI International Conference, HCII 2021, Virtual Event, July 24–29, 2021, Proceedings*, Springer, 2021, pp. 241–257.

[17] M. Rhahla, S. Allegue, and T. Abdellatif, “Guidelines for gdpr compliance in big data systems”, *Journal of Information Security and Applications*, vol. 61, p. 102 896, 2021, ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2021.102896.

[18] M. Palmirani and G. Governatori, “Modelling legal knowledge for gdpr compliance checking”, in *International Conference on Legal Knowledge and Information Systems*, 2018.

[19] *Cloud for europe*, https://cordis.europa.eu/project/id/610650, Accessed: 2025-11-25.

[20] R. E. Hamdani et al., “A combined rule-based and machine learning approach for automated gdpr compliance checking”, in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, ser. ICAIL ’21, São Paulo, Brazil: Association for Computing Machinery, 2021, pp. 40–49, ISBN: 9781450385268. DOI: 10.1145/3462757.3466081.

[21] S. Zimmeck, P. Wang, R. Kang, J. R. Reidenberg, and N. M. Sadeh, “Automated analysis of privacy requirements from privacy policies”, in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 441–458. DOI: 10.1109/SP.2019.00038.

[22] The European Parliament and the Council, “Directive 1995/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data”, *Official Journal of the European Communities*, Oct. 1995.

[23] L. Moreau et al., “The Open Provenance Model core specification (v1.1)”, *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[24] C. Bertolissi, J. Hartog, and N. Zannone, “Using provenance for secure data fusion in cooperative systems”, in *Symposium on Access Control Models and Technologies*, ACM, 2019, pp. 185–194.

[25] Prologia, Ed., *Prolog III : Manuel de référence*. Marseille: Prologia, 1996.