TÉCNICO LISBOA

Building GDPR-Compliant Web Applications with RuleKeeper

Mafalda Baptista Ferreira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Nuno Miguel Carvalho Santos

Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. Nuno Miguel Carvalho Santos
Member of the Committee: Prof. Miguel Nuno Dias Alves Pupo Correia

January 2021

For my grandfather Joaquim,

Acknowledgments

First, I would like to express my special gratitude to my supervising professor, Nuno Santos, for all the guidance and support throughout this last year. Without his dedication, this thesis would have not been possible.

I would like to thank my parents, my sister, my grandmother and my extended family, Paulo and Filipa, for their endless support and for doing their best to accommodate my needs when times were rough. Also my dog Nori, for keeping me company during this era of social isolation and for forcing me to take breaks to walk him outside.

Last but certainly not least, a special thank you to João, for putting up with me during my several meltdowns. Words cannot express how lucky I am to have such caring and inspiring person in my life.

Resumo

Atualmente, as aplicações web oferecem vários serviços aos seus utilizadores mas obrigam-nos a partilhar os seus dados, muitas vezes publicamente. Em 2018, a União Europeia publicou uma abrangente legislação - o Regulamento Geral sobre a Proteção de Dados (RGPD) - que define um sistema de leis que promove a implementação de extensos mecanismos de segurança visando a proteção dos dados dos utilizadores e prevenção de violações de privacidade. Infelizmente, a maioria dos sistemas atuais tende a ser otimizado em prol do desempenho, custo e confiabilidade, secundarizando a segurança e não suportando adequadamente o desenvolvimento de aplicações web sensíveis ao RGPD. Consequentemente, não só os utilizadores permanecem expostos a vários riscos, incluindo a divulgação de dados confidenciais, como as próprias organizações podem incorrer em multas elevadas caso não respeitem o regulamento. Adicionalmente, devido às limitações das frameworks web existentes, os programadores enfrentam vários desafios ao desenvolver as suas aplicações compatíveis com as rigorosas políticas de proteção de dados do RGPD.

Considerando estes desafios, esta tese visa o estudo das implicações do RGPD em aplicações web e propõe requisitos que as organizações devem seguir quando gerem os seus sistemas de informação. Particularmente, propõe RuleKeeper, uma framework para aplicações web que oferece garantias de privacidade e segurança dos dados, respeitando o RGPD. Apresenta também Purposeful Data Objects, uma nova abstração para construir aplicações web cumprindo as restrições do RGPD, e GPSL, uma linguagem declarativa que especifica as políticas do RGPD garantindo que os requisitos mencionados acima podem ser descritos rigorosamente e interpretados automaticamente.

Palavras-chave: RGPD, Privacidade dos Dados, Políticas de Privacidade, Frameworks Web, Desenvolvimento Web

Abstract

Modern web applications provide many useful services to end-users but require them to share their data, sometimes publicly and globally. In 2018, the European Union issued a comprehensive legislation - the General Data Protection Regulation (GDPR) - that defines a system of laws aimed at promoting the deployment of extensive security mechanisms for the protection of users' data and prevention of privacy breaches. Unfortunately, most modern systems tend to be optimized for performance, cost, and reliability, leaving security as a secondary goal and failing to provide adequate support for the development of GDPR-aware web applications. As a result, not only the web users remain prone to numerous risks, including the exposure of sensitive data, but the organizations themselves may incur high fees in the case of non-compliance with the GDPR. Moreover, due to the limitations of existing web frameworks, application developers face numerous challenges in building their applications in adherence to the strict GDPR data protection policies.

Considering these challenges, this thesis studies the implications that GDPR holds in web applications and clarifies the requirements organizations need to follow when managing their information systems. Particularly, it proposes RuleKeeper, a novel web application framework, tailored to provide data security and privacy protections according to GDPR-compliant policies. Additionally, this thesis presents Purposeful Data Objects, a new abstraction for building secure-by-design GDPR-compliant web applications, and GPSL, a policy language for specifying GDPR policies in such a way that the requirements expressed above can be laid out rigorously and interpreted automatically.

Keywords: GDPR, Data Privacy, Privacy Policies, Web Frameworks, Web Development

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis addresses the problem of GDPR-compliance in information systems. The GDPR, issued by the European Union, sets a collection of laws that aim to ensure the deployment of extensive security mechanisms for protection of users' data and privacy. However, building information systems as off now has been at odds with compliance regarding the GDPR. Frameworks have become an essential part of web development, providing not only scalability, integration, and robustness, but also being easy to integrate and time-saving. As these frameworks are not GDPR-compliant by default, building information systems with the support of such frameworks implicates the need to enforce GDPR policies without disrupting them. This thesis studies the restrictions imposed on both data subjects, whose data is being processed in such information systems, and organizations, who are responsible for processing such data, and presents a solution for the problem of enforcing GDPR-compliance in information systems, without disrupting existing web frameworks.

1.1 Motivation

The continuous technological growth and globalization have boosted the free flow of data and increased the scale of collection and sharing of personal data. Today, people tend to share their data publicly and at an unprecedented scale to use many of the services provided by both private and public companies and authorities. Given that the protection of natural persons concerning the processing of personal data is a fundamental right [1, 2], the need for establishing some data privacy and security standards has arisen. To protect citizens, the European Union has issued a regulation that sets a collection of rules and guidelines that aim to ensure the deployment of extensive security mechanisms for the protection of users' data and privacy – the General Data Protection Regulation (GDPR) [3].

Since many data processing and storage platforms are based on web applications, the existence of these mechanisms has an essential role in the data privacy and security regarding web applications because web users face numerous risks whenever they entrust their data to web applications. One of such risks consists in the exposure of sensitive data which is considered one of the most serious web application security risks since many applications and APIs do not properly protect sensitive data, such as financial, healthcare, and personally identifiable information (PII). In fact, over the last few years, this has been the most common impactful attack on organizations [4]. Software exploits, weakly protected data, mishandling of server-side services, or questionable practices carried out by organizations are just a few examples of hazards that may lead to data breaches and loss of privacy. This can also lead to credit card fraud, identity theft, or other crimes. As a result, not only the web users remain prone to the aforementioned risks, but the organizations themselves may incur high fees in the case of non-compliance with the GDPR.

Moreover, due to the limitations of existing web frameworks, application developers face numerous challenges in building their applications in adherence to the strict GDPR data protection policies. First, new actors need to be introduced to the system, such as the data subject, data controller, and data processor, requiring the developer to separate their responsibilities in the system. Second, GDPR introduces the concept of personal data. However, since the developer is not obligated to follow a standardized structure when defining the application data model, information systems do not, by default, distinguish data type categories, such as personal data, sensitive data, health related data, or the others. From the concept of personal data arises the concept of data ownership, which requires an association between personal data and the corresponding data subject, that may or may not have consented to the processing of its data. However, more often then not, the developer implements application-specific operations that query data from the database, with no regard of who it belongs. As so, there is no way to ascertain if the data subject that owns the data that is being processed consented to it or not.

In our work, we aim at building a web framework that enables web developers to write applications that can handle user data in compliance with GDPR policies while preserving the maintainability and performance delivered by the modern web frameworks. Such framework shall allow the specification of data protection policies in accordance with the GDPR and hard guarantee the enforcement of such policies throughout the entire application life cycle. It should also help prevent privacy breaches, increase transparency, hand over control of data to its owner, and assist organizations in avoiding penalties due to noncompliance.

1.2 Contributions

This thesis starts by analyzing the GDPR, studying the implications its principles hold in web applications. We notice that many of these requirements go beyond traditional access control or obligation policies, and report the need to specify the organization's privacy policy in a clear and concise way. Then, we explore approaches that allow the development of web applications such that it will be possible to guarantee data privacy and security according to the GDPR. As a result, this work makes the following contributions:

- Presents a detailed study of the GDPR to determine its implications concerning personal data processing and the challenges it raises, namely ambiguities, indefinitions and omissions. We clarify the requirements organizations need to follow when managing their information systems;

- Introduces a new abstraction for building secure-by-design GDPR-compliant web applications named Purposeful Data Objects (PDOs);
- Proposes GPSL, a policy language for specifying GDPR policies in such a way that the requirements expressed above can be laid out rigorously and interpreted automatically;
- Proposes RuleKeeper, a web development framework for 3-tier web applications which provides an API for the development of PDO-based web applications and a system for enforcing compliance with GPSL policies.
- An implementation of a RuleKeeper prototype and of a use case application to to perform a real-world validation of our system.
- An experimental evaluation of the RuleKeeper's prototype, measuring the performance impact in existing web applications.

1.3 Thesis Outline

The rest of the report is organized as follows. Section 2 presents some necessary background on the GDPR and the relevant related work. Section 3 provides a detailed analysis of the GDPR, covering the main requirements and discussing potential ambiguities in the law. In Section 4 we describe the system and Section 5 describes how we implemented its prototype. Section 6 describes how we evaluated our system. Finally, Section 7 concludes the report.

Chapter 2

Background and Related Work

This chapter provides some necessary background on the GDPR (Section 2.1). Thereafter, it presents the state of the art focused on the GDPR and security policies for web applications. Section 2.2 provides a study concerning the best practices and limitations for GDPR compliance. Section 2.3 presents a study of the most relevant security policy languages and enforcement mechanisms. Lastly, Section 2.4 provides an overview of the existing information flow and mandatory access systems that share some common goals with ours: the enforcement of end-to-end security policies for web applications.

2.1 Brief Introduction to the GDPR

The General Data Protection Regulation (GDPR) is the core of Europe's data privacy and security legislation. It came into force on May 25, 2018, and it refers to the regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 and repealing Directive 95/46/EC (General Data Protection Regulation) [3]. This regulation sets a collection of rules regarding the protection of natural persons, respective to the personal data processing, and the free movement of such data to protect their fundamental right to the protection of personal data, as stated in **Art. 1** of the **GDPR** [3].

2.1.1 Key Data Protection Roles

The processing of personal data includes any action performed on personal data, whether automated or manual (e.g., accessing or storing the data). This process involves at least two entities, described in **Art. 4** of the **GDPR** [3]. First, the legislation introduces the **data subject**, the entity whose personal data is processed. Then, it defines two main roles involved in personal data processing: the **data controller** and the **data processor**. The **data controller** is responsible for determining the purposes and means of the personal data processing (e.g. *web services*), and the **data processor** is responsible for processing the personal data on behalf of the data controller (e.g. *cloud servers*). A single entity can secure both roles. Typically, as stated in **Art. 37** of the **GDPR** [3], the data controller and the data processor are expected to designate a **Data Protection Officer** (DPO), who oversees the data protection strategy and implementation within a given company and ensures proper compliance with GDPR requirements.

2.1.2 Personal data

Art. 4 of the **GDPR** [3] defines **personal data** as any information relating to an identified or identifiable natural person, where an identifiable natural person is a person that can be identified directly or indirectly, in particular by reference to an identifier. This identifier can be a name, an identification number, location data, an online identifier (for instance, Internet protocol addresses and cookie identifiers), or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person. The GDPR does not provide an exhaustive list of all data types that can be classified as personal data, leaving it open to any data of any type and regardless of the medium, that allows the identification of a natural person.

Special data categories: Within the personal data category, there are some special categories that require particular attention because its processing is subject to more stringent limits. According to **Arts. 8-10** of the **GDPR** [3], personal data can be classified as *sensitive data*, can belong to *children* (i.e., less than 16 years old), and can be related to *criminal convictions and offences*. **Sensitive data** comprises personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and it also includes genetic and/or biometric data for uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation. As stated by **Art. 9** of the **GDPR** [3], data concerning health includes all information relating to the past, current, or future physical or mental health of a natural person, including the provision of health care services, any number or symbol assigned to a natural personal with the intention of identifying them for health purposes and any information derived from genetic data and biological samples.

2.1.3 Processing Requirements

We divide the GDPR scope of application in two main categories: the *processing requirements* and the *data subject rights*. The processing requirements refer to the GDPR requisites regarding the actual processing of the personal data. The data subject rights concern the options that must be given to the data subjects in order for them to exercise the rights that the GDPR provides to them. The first category is covered in this section, and the second in the following one.

According to the GDPR guidelines, the processing of personal data must adhere to a set of requirements. The controller shall be responsible not only for meeting them but also for demonstrating compliance. **Art. 5** of the **GDPR** [3] briefly describes these principles:

- *Lawfulness, Fairness and Transparency*: Personal data must be processed lawfully, fairly, and in a transparent way concerning the data subject. The data subject should be made aware that his personal data is being collected and to what extent it is or will be processed and, in some instances, he must give his consent to this processing for one or more specific purposes.

- *Purpose Limitation*: Personal data must be collected for specified, explicit and legitimate purposes and should not be processed in a way that is incompatible with them. Additionally, these purposes should be explicit and established when personal data is being collected.

- *Data Minimization*: Personal data must be adequate, relevant, and limited to what is necessary in relation to the purposes for which they are processed. This means that only the essential and indispensable personal data should be processed.
- *Accuracy*: Personal data must be accurate and kept up to date. It should be ensured that, if there is inaccurate data, every reasonable step should be taken to rectify or erase the data without delay.
- *Storage Limitation*: Regarding the *Data Minimization* requirement, it should also be ensured that the period for which the personal data is stored is limited to the minimum.
- *Integrity and Confidentiality*: Personal data should be processed safeguarding its security and confidentiality, including preventing unauthorized or illicit access or use of personal data.
- *Accountability*: The controller has the responsibility to enforce and demonstrate compliance with the above principles.

2.1.4 Data Subject Rights

Art. 12 of the **GDPR** [3] states that natural persons should have control of their own personal data and advises the controller to facilitate the exercise of the data subjects' rights, including mechanisms to request them and, if applicable, obtain, free of charge, the appropriate response.

- *Right to Access* (**Art. 15** of the **GDPR** [3]): The data subject must be given the right to know if his personal data is being processed, and when that is the case, it should also be given the right to access the personal data concerning him and information regarding the processing. The controller must grant a copy of the personal data undergoing processing.
- *Right to rectification* (**Art. 16** of the **GDPR** [3]): The data subject must be given the right to complete and to rectify inaccurate or incomplete personal data concerning him, without unreasonable delay.
- *Right to erasure* (**Art. 17** of the **GDPR** [3]): The data subject must be given the right to obtain from the controller the erasure of personal data concerning him, without unreasonable delay, as long as some conditions apply. The controller must erase the data in an irreversible way, ensuring all links to it and all copies and replicas are also erased.
- *Right to restriction of processing* (**Art. 18** of the **GDPR** [3]): The data subject must be given the right to obtain from the controller restriction of processing regarding his personal data, in some predetermined circumstances.
- *Right to data portability* (**Art. 20** of the **GDPR** [3]): The data subject must be given the right to receive the personal data that he provided to the controller, in a structured, commonly used, and machine-readable format and to transmit that data to another controller.
- *Right to object* (**Art. 21** of the **GDPR** [3]): The data subject has the right to object, for reasons regarding his particular situation, at any time, to the processing of personal data concerning him.

- *Right to withdraw consent* (**Art. 7** of the **GDPR** [3]): The data subject must be given the right to withdraw his consent at any time and should not affect the lawfulness of processing based on consent before its withdrawal. It must be as easy to withdraw consent as it is to give it.

2.1.5 GDPR Impact on Organizations

Given that the regulation itself is extensive, far-reaching and considerably poor on specifics, each data controller must interpret and apply the regulation principles to their specific case. This new set of rules has a significant direct financial impact in the organizations since the infringement of the regulation can result in penalties, which can be administrative fines. Those fines are designed to convert noncompliance into a costly mistake and therefore strengthen the enforcement of those rules. The cost depends on the type of violation, and in the most costly case, the administrative fines can reach up to € 20 million or 47% of the global revenue (whichever is higher) plus the compensation the organization may have to give to the data subjects, regarding damages to them, as stated in **Arts. 83-84** of the **GDPR** [3].

Consequently, to reduce or even eliminate this cost, organizations must now start implementing data privacy and security mechanisms by law, which may reflect in a newer and more conscious investment. **Art. 25** of the **GDPR** [3] also states that organizations must follow privacy by design and by default regarding data protection, meaning that the controller must implement the data protection principles since the conception of a new product/process throughout all the process of development.

2.2 Existing Research on GDPR Compliance

Some recent studies aim to investigate the best practices and limitations in GDPR compliance and the extent to which the data protection policies prescribed by this regulation have been implemented in reality. We mention some of the most in-depth studies performed until the time of this writing.

2.2.1 GDPR-compliant Privacy Policies

The study by Mohan et al. [5] leverages on the importance of clear and concise privacy policies and proposes some recommendations for GDPR-compliant privacy policies supported by the principles laid out by the regulation, specially, **Art. 5** of the **GDPR** [3]. They propose a set of seven questions that any person reading the privacy policy must be able to answer, translated from the GDPR principles:

- (i) *“Who collects the user data and who are the users of data?”*
- (ii) *“What personally identifiable data is collected?”*
- (iii) *“When will the collected data expire and be deleted?”*
- (iv) *“Why is the data being collected?”*
- (v) *“How can the user request to exercise their rights?”*

- (vi) *"Does the controller take measures to ensure safety and protection of data?"*
- (vii) *"Does the controller notify users appropriately when changes are made to the privacy policy?"*.

The authors proceed to analyze the privacy policies of ten large-scale cloud services that claim to comply with the regulation, such as Bloomberg, Instagram, Uber, and iCloud and identify several vulnerabilities, according to their principles, that could potentially lead to noncompliance and result in fines. Some of these vulnerabilities comprise unclear and vague data sharing and retention policies, weak data protection policies, and the lack of fine-grained control over user data.

This study concludes that most privacy policies do not meet all the requirements of GDPR-compliance and, as the GDPR has some grey areas open to interpretation, some companies unfortunately use the vague specifications of the GDPR to their benefit. As we aim to implement a privacy policy that is compliant with the GDPR, this study provides very useful insight on how to design it.

2.2.2 The Right to be Forgotten

In close relationship with the GDPR is the Right to be Forgotten (RTBF). It is a privacy ruling that allows Europeans to request the "delisting" of certain URLs from search results that may contain inaccurate, inadequate, irrelevant, or excessive information relating to the requester's name. Comparing to other privacy techniques, the RTBF is unique in how it addresses multiparty privacy conflicts since even the access to lawfully published, truthful information about a person may be restricted by search engines.

To arrive at a verdict regarding the delisting of a URL, search engines like Google must balance the public's interest in that information versus the individual's right to privacy. Google's decision process is manual, where they assign each request to one or more reviewers, who make their decision based on a set of defined criteria, such as the identity of the requester, the type of content referenced by the URL, and the type of source of the information. Google produces a report every year containing statistics of the requests and it was possible to observe a decline in directory-related sites since only half of them remained online – this is likely the result of the GDPR coming into effect.

A recent study on Google's RTBF [6] shows that, although users are exercising their rights, sometimes it is not a straightforward decision, for example, when the public interest outweighs the individual's right to privacy. This means that the process of decision regarding a user's request cannot be automated unless the organization decides to accept all the requests after confirming the validity of the request.

2.2.3 Cookie Banners

Since the GDPR came into effect, the majority of popular websites in Europe began to display cookie consent notices, also called "cookie banners", to ask users for consent prior to setting cookies [7]. The frequency of these notices along with the fact that most of the time they cover the main parts of the website, caused users to become annoyed and look for solutions such as blockers. The issue with this type of solutions is that they take on default values, leading to data collection taking place without user consent since many times the default is to opt-in. Besides that, cookie banners are very different from each other (both in a graphical and usability manner), making users believe that their choice does not matter, and making them click anywhere for the notice to disappear.

This recent paper [7] presents an experimental study on how the notice's design properties affect users' behavior, aiming to collect the features that will most likely make them interact with them instead of being annoyed while also providing enough detail to make them aware of a website's data collection practices. This is important to our work since the user's consent is one of the components we need to design and implement, so we should take into consideration the produced recommendations.

2.2.4 Impact of the GDPR on Storage Systems

Storage systems tend to be optimized for performance, cost, and reliability. A study from 2019, focused on the impact of GDPR in storage systems [8], illustrates properly the struggle between such goals and GDPR compliance. The authors examine the GDPR articles and propose a set of 6 features that a storage system must implement in the storage layer to achieve GDPR-compliance, matched to such articles and requirements. These features comprise: (i) **Metadata Indexing** for quick and efficient access to groups of data, (ii) **Timely Deletion** for guaranteeing that no personal data id retained for an indefinite period of time, (iii) **Monitoring and Logging** for demonstrating compliance, (iv) **Access Control** for imposing fine-grained and dynamic restrictions to data access, (v) **Encryption** for data security and (vi) **Managing Data Location** for restrictions regarding geographical locations of the personal data.

The authors also introduce two interesting concepts about the degree of compliance: *real-time vs. eventual compliance*, and *full vs. partial compliance*. The first one measures the response time: GDPR compliance is deemed to be "real-time" when it is enforced synchronously and "eventual" otherwise. The second one measures capability and describes full compliance when the system natively supports all features, and partial when external infrastructure or policy components are needed. They use Redis to implement such features and present results that conclude that full compliance drops Redis throughput to 5% of its original and with eventual compliance, 30% of the original. This demonstrates that achieving GDPR compliance can deteriorate severely the system's performance.

Another study [9], partly developed by the same authors, also describes the tension between the GDPR requirements and the operational practices of modern computing systems. It portrays seven principles and practices of modern systems that collide with the GDPR – named the *seven sins*, and discuss real-world implications and consequences of such avoidable practices. The practices can be enumerated as follows: *Storing Data Forever*, *Reusing Data Indiscriminately*, *Walled Gardens and Black Markets*, *Risk-Agnostic Data Processing*, *Hiding Data Breaches*, *Making Inexplicable Decisions* and *Security as Secondary Goal*. Both studies illustrate very well the challenges of retrofitting existing systems to abide by the GDPR, the implications for system designers, and the ensuring performance issues.

2.2.5 GDPR Compliance by Construction

Advocating a compliant-by-construction approach, Schwarzkopf et al. share a vision in how to achieve GDPR compliance in web service's backends, oriented towards data subject's rights and how to provide

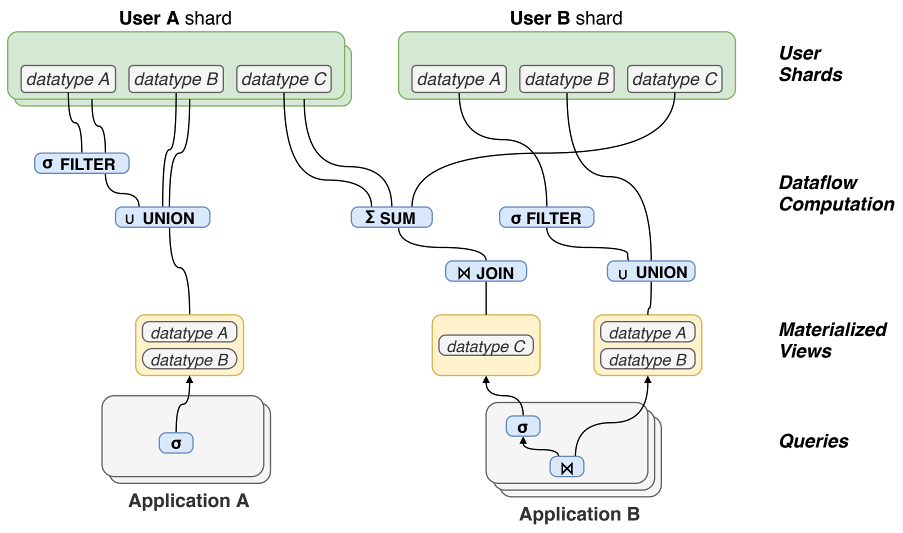


Figure 2.1: User shard's architecture with stateful dataflow. Yellow components represent the materialized views, blue components represent the queries and the user shards are represented by green.

them control of their own data. The authors propose an architecture based on the concept of data ownership and on the need to associate all data to the data subject it belongs to.

The idea behind this architecture is to logically separate users' data, introducing **data shards**, where each user has their own structured shard of the storage backend. Data shards highlight the data ownership concept since each data shard stores all information concerning a user, and such data shard does not contain information related to other users. Then, as users interact with the web service, their shards are continuously updated and queried. **Materialized views** are built to combine user's data and are the target of applications' queries. They also propose a novel abstraction: a **partially stateful dataflow** model, that links each user shard to the materialized views that will be affected. This model allows the development of streaming dataflow computations over the user shards and is expected to grant users unprecedented control over their data. Figure 2.1 presents the described architecture.

With both these abstractions, the exercise of the data subject rights is attainable as described next. The **Right to Access** is exercised by sending the user a copy of its shard. The **Right to Erasure** involves removing the correspondent user shard from the system. **Right to Data Portability** results from a combination between the previously mentioned rights, where its user shard is retrieved and then removed. Then, the **Right to Object** is achieved by adding *guard* operators that verify if the user has objected, and, if so, prevent the flow of data.

This vision proposes declarative specifications, including the specification of personal data classes, interesting access control mechanisms, cross-system abstractions, and schemas for mapping the data. Although it advocates a *compliance by construction* approach, as we do, it still presents some major limitations. For one, it only facilitates the exercise of data subject rights, lacking the mechanisms to enforce GDPR-compliance concerning the personal data processing itself. In addition, the user shard architecture does not incorporate any concept of purposes, consent, or even data security, and does not allow the customization of policies by the DPOs for specific organizations.

2.3 Security Policy Specification Languages

In the context of our work, a policy specification language is necessary to specify the rules of GDPR policies. When enforced, these policies will ensure the confidentiality, integrity, and accountability regarding personal identifiable information. To evaluate a policy specification language, we need to consider several essential features. The language needs to be expressive and flexible enough to describe, in a non-ambiguous way, the organizations' GDPR policies. At the same time, it must have limited complexity since GDPR policies will be specified by users with little programming knowledge, such as the Data Protection Officers. Furthermore, the policy enforcement mechanisms should not significantly slow down the performance of the information systems nor require significant integration effort. It should also be possible to detect potential conflicts between policies that express incompatible rules. In this section, we study related languages and analyze their strengths and limitations in light of these requirements.

2.3.1 XACML Language Family

XACML [11] is a declarative XML-based policy language for supporting access control policies in distributed systems. It is suitable for attribute-based access control (ABAC) systems. XACML is used to define requirements that must be matched in order to give access to a resource and is based on a response/request protocol, where a decision engine is queried using real access requests against the XACML policies and returns the decision: *Permit* or *Deny*.

The main language artifacts of XACML are: rules, policies, and policy sets, where rules can be combined into one policy, and policies into a policy set. The policies and policy sets, in turn, can be combined into larger policies, granting policy scalability. It allows to define rules regarding different resources, actions, subjects, and environments. The **rule** is the elementary unit and is composed of a **target**: specifying the set of requests to which it applies, a **condition**, an **effect**: indicating the consequence of a *True* evaluation for the rule, **obligation** expressions, and **advice** expressions. Attributes are associated with a user, action or resource and serve as inputs to the decision of whether a given user may access a given resource in a particular way.

Despite its flexibility, XACML is rather user-unfriendly as it is very complex, verbose, and requires extensive knowledge and practice [12]. Listing 1 shows a simple policy that specifies an authorization rule to be enforced when a user tries to access patients' data (lines 4-29). This access is permitted when the user has the *doctor* role (lines 31-35) and works in a approved list of hospitals (lines 36-41). To improve the policy readability, we omitted the URNs that identify the XML namespaces. To alleviate this problem, it was developed ALFA [13], a high-level language that simplifies the specification of policies and then generates automatically the correspondent XACML policies. However, even though ALFA makes the policy specification more user-friendly, the verbosity and overhead of policy enforcement remains unchanged since the generated policies are still in XACML. Since XACML is a defined OASIS standard [14], there are several implementations and extensions, as we mention next.

PPL [15] is an XACML extension that enables privacy-preserving access control by adding data handling and credential capabilities based on the notion of *trusted credentials*. It allows a data controller to define

```xml
<xacml3:Policy PolicyId="PatientInformationAccess" RuleCombiningAlgId="deny-overrides">
  <Description>Authorization Policy for Accessing Patients' Information</Description>
     <xacml3:Rule Effect="Permit" RuleId="PatientInformationAccess.doctorAuthorization">
        <xacml3:Target>
            <xacml3:AnyOf>
                <xacml3:AllOf>
                    <xacml3:Match MatchId="string-equal">
                        <xacml3:AttributeValue DataType="string">/patient</AttributeValue>
                        <xacml3:AttributeDesignator AttributeId="resource-id" DataType="string" Category="resource"/>
                    </xacml3:Match>
                </xacml3:AllOf>
            </xacml3:AnyOf>
            <xacml3:AnyOf>
                <xacml3:AllOf>
                    <xacml3:Match MatchId="string-equal">
                        <AttributeValue DataType="string">access</AttributeValue>
                        <AttributeDesignator AttributeId="action-id" DataType="string" Category="action" />
                    </xacml3:Match>
                </xacml3:AllOf>
            </xacml3:AnyOf>
        </xacml3:Target>
        <xacml3:Condition FunctionId="and">
            <xacml3:Apply FunctionId="all-of">
                <xacml3:Function FunctionId="string-equal"/>
                <xacml3:AttributeValue DataType="string">Doctor</AttributeValue>
                <xacml3:AttributeDesignator AttributeId="role" DataType="string"/>
            </xacml3:Apply>
            <xacml3:Apply FunctionId="string-at-least-one-member-of">
                <xacml3:Function FunctionId="string-equal"/>
                <xacml3:AttributeValue DataType="string">Hospital X</AttributeValue>
                <xacml3:AttributeValue DataType="string">Hospital Y</AttributeValue>
                <xacml3:AttributeDesignator AttributeId="hospital" DataType="string"/>
            </xacml3:Apply>
        </xacml3:Condition>
    </xacml3:Rule>
</xacml3:Policy>
```

Listing 1: XACML simple authorization policy example for accessing patients' data.

access permissions and restrictions to resources and specify how the collected personal data will be processed. The data subject can define permissions and restrictions to his own personal data and his data processing preferences. PPL uses a new handling mechanism named *obligation* that can be defined as "a promise made by a data controller to a data subject in relation to the handling of his/her personal data" [15] and takes into consideration temporal constraints. It also introduces the concept of *sticky policies*, which translate into the obligations settled between the data subject and data controller.

A-PPL [16] is an extension to PPL that incorporates the definition of accountability policies, and transparency of personal data processing [17]. Accountability obligations may derive from regulations, such as the GDPR, and by introducing accountability rules is possible to define policies on data retention, data location, logging, and notification. These rules allow to set a validity duration of a data usage purpose, control which regions a resource can be used for a certain purpose (and therefore limiting the region among which the data can be exchanged) and enable the creation of evidence that can be presented to verify compliance with policies, user preferences, or regulations. Since both languages are extensions of XACML, it perpetuates the same issues of complexity and user unfriendliness [18].

EPAL [19] is a privacy authorization language for enterprises, developed by IBM. It shares the same policy enforcement model as XACML, but lacks significant features required for complex enterprise policies, both for privacy and for access control in general [20].

2.3.2 Datalog Language Family

Datalog [21] is a database query language based on the logic programming paradigm. It has been widely used for the specification of declarative security policies. These policies vary from being used for network protocols [22], for storage security [23] [24] or authorization languages, such as some relevant Datalog variants that we briefly present next:

SecPAL [25] is a decentralized, declarative privacy authorization language developed by Microsoft Research. Policies are expressed using logical clauses, which comprise sets of assertions in the form:

$$E \text{ says } f_0 \text{ if } f_1, ..., f_n \text{ where } c$$

where E is a constant, representing the principal, f_i are facts and c represents the assertion constraint. When an access request is received, it is mapped into logical authorization queries and the decision is made by matching data handling policies against the user preferences.

SecPAL4P [26] *(SecPAL for Privacy)* extends SecPAL by enabling the specification of personal data handling and it is designed to make preferences and policies as human-readable as possible. It allows users to define their preferences on how services should process their personal data, in terms of granted rights and required obligations, using SecPAL assertions. Similar to SecPAL, SecPAL4P comes with a human-readable syntax and semantic simplicity while being expressive and flexible enough for authorization preferences. Unfortunately, SecPAL4P is presented as formal definitions and algorithms, that formalize the notions of preferences and policies, as well as satisfaction and compliance, but the policy specification is not available publicly.

```
package patient_information

default allow = false

allow {
    action_is_access_patient
    user_is_doctor
    doctor_works_in_approved_hospital
}

action_is_access_patient {
    input.action == "access"
    input.resource == "patient"
}

user_is_doctor {
    input.user.role == "doctor"
}

doctor_works_in_approved_hospital {
    approved_hospitals := ["Hospital X", "Hospital Y"]
    input.user.hospital == approved_hospitals[_]
}
```

Listing 2: Simple authorization policy example in Rego for accessing patients' information

Rego [27] is a high-level declarative language that extends Datalog with support for structured document models (e.g. JSON). Rego is based on if-statements, replicating real world policies and making the policy specification more human friendly while still understandable to machines. This simplicity is one

	Specification Complexity	Expressiveness	Privacy-Oriented	Verbosity
XACML	High	Medium	No	High
PPL	High	High	Yes	High
A-PPL	High	High	Yes	High
SecPAL4P	Medium	High	Yes	Unknown
Rego	Low	Medium	No	Low

Table 2.1: Comparison of policy specification languages.

of the main advantages of Rego, since it allows policy authors to not over complicate the logic and hence optimize performance. Rego queries consist of assertions on structured data stored in JSON documents. Rego rules can be defined in terms of scalars, composite values, variables, and references. The policy in Listing 2 implements in Rego the same XACML policy as Listing 1. As so, this simple policy also specifies an authorization rule to be enforced when a user tries to access patients' data (lines 6 & 11-14). This access is permitted when the user has the *doctor* role (lines 7 & 16-18) and works in a approved list of hospitals (lines 8 & 20-23). A typical policy in Rego tends to be more simple and user-friendly than the same policy in XACML. It should also be noted that the complexity difference is expected to increase as we add more information and rules.

2.3.3 Discussion

Up until this point, we made a concise survey of the most relevant policy specification languages and engines, namely XACML, PPL, A-PPL, SecPAL4P and Rego. Table 2.1 summarizes this discussion.

XACML is a very expressive language for access control purposes. PPL and A-PPL are more oriented towards privacy, extending XACML regarding privacy and accountability capabilities, respectively. This expressiveness has the cost of a high specification complexity and verbosity, which makes them user-unfriendly. Since our policies aim to be specified by users with limited programming skills, such as Data Protection Officers, XACML and its extensions are too complex for our scenario. SecPAL4P offers no public implementation, so we are unable to specify our policies with it.

Rego policies are very simple, flexible, easy to use, and maintainable. Although it is not privacy oriented, its flexibility allows us to enhance it with the necessary extension points. It also supports structured document models, such as JSON, which is a valuable feature when building web applications. Thus, both the approach and advantages of Rego are expected to be a better suit for our system.

2.4 Policy Enforcement Systems for Web Applications

Over the years, many systems have been proposed to enforce end-to-end security policies for web applications. In this section, we provide an overview of the most relevant ones for our work according to two categories: information flow control systems and mandatory access control systems. Lastly, we provide a brief discussion summarizing the strengths and weaknesses of these systems.

2.4.1 Information Flow Control Systems

Over the years, several systems have been proposed to enforce end-to-end security policies for web applications, by employing information flow control mechanisms. In this section, we provide an overview of the most relevant systems that share some common goals with ours.

Servlet Information Flow (SIF)

SIF [28] is a framework used for developing web applications for the Java Servlet Framework while expressly assuring confidentiality and integrity through information flow control (IFC) security policies. SIF enforces end-to-end security at the language level. It tracks all information flows within the web application generated by the clients, ensuring that the application only uses data in compliance with the specified policies. Because it is assumed that a) web clients can potentially be malicious and b) web applications, although benign, can have bugs, the confidentiality and integrity security policies are enforced on server-side.

In SIF, information flow tracking is done through label propagation. Web applications must be developed in Jif 3.0, an extension of the original security-typed Jif language [29] that adds integrity annotations over data types. SIF web applications are compiled against the Jif signatures to check if they do not release confidential information wrongfully to clients and do not use low-integrity information from clients in high-integrity contexts. The applications are linked at runtime to the Java classes and monitored via web request handlers, which can be defined by the application developers. Thus, rather than trusting entirely in the web applications, SIF shifts that trust to the framework itself and to the Jif 3.0 compiler.

SIF delivers a good performance in taint tracking, presents valid ideas on how to handle requests since it allows the application code to define how client requests are handled, acting as a middleware for the requests, and tracks information flow over multiple requests from the same client. However, SIF presents several shortcomings: (i) the policy must be specified alongside the application code, (ii) the policies are global and are unable to reflect the data subjects preferences, (iii) it is not context-aware and (iv) it has no concept of GDPR principles, such as consent and storage limitation.

Fabric

Fabric [30] is a federated system and a high-level IFC language for building secure distributed information systems. It allows shared access to information and computation between cooperating but mutually distrustful entities. It keeps track of information flows across network nodes, whether to improve performance or to meet security requirements, easing the development of such applications. Just as SIF, the Fabric programming language extends Jif 3.0 to implement access and information flow control and adds support for distributed programming and transactions. This allows the prevention of confidentiality and integrity violation by untrusted nodes. Fabric aims to tackle several issues which are common to our goals, such as persistence, consistency, security, and distributed computation. However, it presents the same shortcomings of SIF and to use the security component, it is needed to implement the whole decentralized system, adding unnecessary overhead.

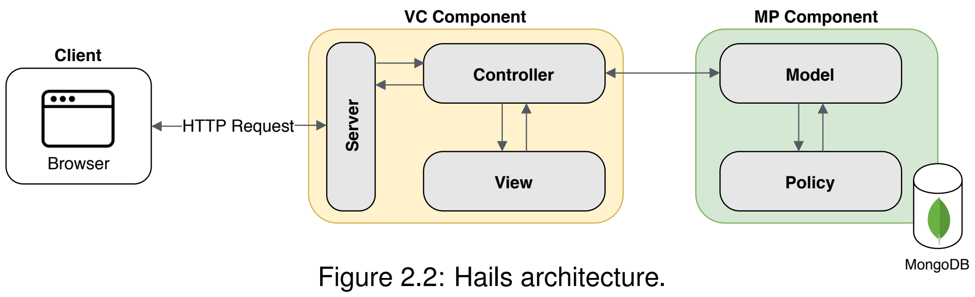


Figure 2.2: Hails architecture.

Resin

Resin [31] is a language runtime that aims to mitigate security vulnerabilities, such as SQL injection, XSS, server-side script injection, or password disclosure in web applications, by using the concept of *data flow assertion*. Similar to SIF, it also allows to specify application-specific rules. By using application-level data flow assertions, developers can explicitly specify a data flow plan and have that assertion checked at runtime, using interpreters like Python or PHP, to capture dynamic properties. An example of an assertion to prevent SQL injection attacks may be “Any user input data must flow through a sanitization function before it flows into a SQL query”.

Resin provides three different mechanisms for implementing data flow assertions: *policy objects*, *data tracking*, and *filter objects*. The policy objects contain code and metadata for checking assertions specific to each data type. These policy objects are then associated with the application data and used to perform data tracking in runtime as the data flows through the application. Filter objects are used to define the data flow boundaries at which the assertions are checked.

Resin provides a good approach for mitigating security vulnerabilities such as SQL injections. However, although there is some reuse in existing code and data structures, just as SIF, adding a RESIN assertion or a filter object to an application requires deep changes to the application code.

Hails

Hails [32] is a web framework that employs a combination of language-level, OS-level, and browser-level confinement mechanisms. It is designed to extend the MVC architecture by adding mandatory access control and a declarative policy language, resulting in a model-policy-view-controller (MPVC) architecture. Figure 2.2 describes Hails’ MPVC architecture. Just as the MVC architecture, the MPVC model represents the application’s persistent data structure, the controller handles the requests, and the view is a presentation layer for the end user.

Hails’ MPVC model adds a *Policy* component which associates each model with a policy specifying how the associated data may be used. MVPC applications are built from two mutually distrustful components: the MP component that comprises model and policy logic, and the VC component that comprises view and controller logic. They communicate through an API provided by the MP component that allows other components (VC’s) access the data regarding the defined policies. Hails follows the data throughout the system and supports policies to restrict information flow control in and out of the application. It specifies policies as confidentiality and integrity labels that define which principals are allowed to read

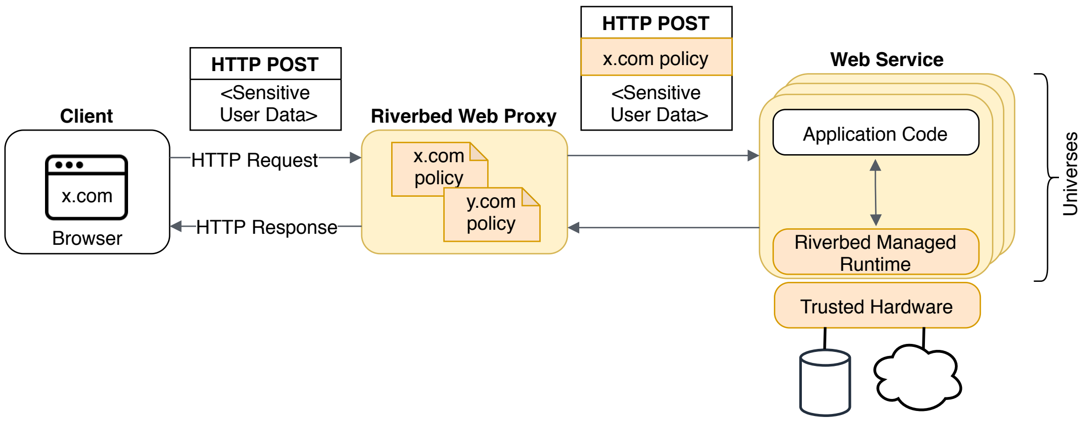


Figure 2.3: Riverbed's architecture. Yellow and orange components represent the Riverbed add-ons.

and write data, respectively. A principal can be a user, a remote website, an MP, or a VC. The trusted runtime checks if remote principals satisfy the associated labels before allowing communication.

In contrast to Fabric, Hails can store documents with different security labels for individual elements. It follows a suitable approach regarding the database system and allows the labels to be associated with each field and as functions of other fields. It is useful for systems that combine mutually distrustful applications written by various entities and is designed for web applications. However, it provides weak security guarantees on the client side, since it allows a malicious VC to coerce a user to declassify sensitive data. Just as the previously mentioned systems, the specified policies are global, not context-aware, and not sensitive to GDPR privacy concepts.

Riverbed

Riverbed [33] is a framework for building web services that safeguard the privacy of user data according to some GDPR constraints. Riverbed allows users to define restrictions regarding the processing and storage of their sensitive data by a remote service, using simple and human-readable policies.

Figure 2.3 depicts Riverbed's architecture. For each user HTTP request, a Riverbed web proxy on the user's device adds a special HTTP header containing the correspondent data flow policy. Since many services run atop managed runtimes, Riverbed modifies the runtime to automatically taint the incoming HTTP data with the associated user policies and whenever new data is generated from tainted data, it also marks it as tainted. If a process attempts a disallowed externalization, the Riverbed runtime terminates it. The Riverbed proxy only sends data to a server after the backend code remotely attests to the proxy, providing evidence that the server is running an IFC-enforcing Riverbed runtime. If this attestation succeeds, the proxy allows access to the user's data, labeling it with the associated policies.

Riverbed innovates relating to other instances of IFC systems by detaching the Riverbed IFC system from the application code. This is because in traditional IFC systems it is difficult to create and maintain partial ordering of labels. As the previously presented systems, it follows a similar approach for following the data by labelling it, but unlike them, the application code is unaware of the taint tracking and application developers are unable to read, write, create or destroy taints and data flow policies.

With Riverbed, it is easier for developers to comply with regulations such as the GDPR because users can give explicit consent using the Riverbed policies. However, Riverbed has a major drawback. It only allows user-defined policies regarding their own personal data. This implicates that a DPO is unable to specify a global policy regarding an organization to be enforced by all users at the same time. As the proposed policy language is also not expressive enough to describe an organization's GDPR policy, Riverbed is unsuitable for building secure-by-design GDPR-compliant web applications.

2.4.2 Mandatory Access Control Systems

Following a different approach, systems that employ mandatory access control mechanisms have also been proposed to enforce end-to-end security policies for web applications. In this section, we provide an overview of the most relevant systems that share some common goals with ours.

Qapla

Qapla [34] provides a policy compliance middleware for database-backed systems. It is focused on mitigating application bugs that may result in lost of data confidentiality. Qapla's authors advocate for policy decoupling from application code, by implementing policy enforcement independently of application correctness. Not only this approach does not require the developer to instrument the policy enforcement mechanisms as it also improves policy maintainability, since the application developers do not need to update such mechanisms whenever policy changes.

Qapla's policies specify confidentiality requirements, being expressed as a function of the database schema. For enforcing Qapla's policies, is provided a reference monitor. This monitor intercepts all application queries, searches for the applicable policies and rewrites the query to be compliant with such policies. Query rewriting is performed before the query is executed, filtering the non-compliant records, according to the applied policies. Only then, the query is executed in the database. Qapla applies a whitelist principle to query execution, by only allowing it if at least one policy applies. This guarantees that a query is not wrongfully executed due to the accidental omission of policies.

Qapla's policies can express fine-grained row, column and cell level policies and can also include the authenticated user and time. Each policy specifies how queries must be restricted to be compliant, through the use of WHERE clauses.

Qapla's approach is very sophisticated and advocates for policy decoupling and maintainability as we do. However, it has no concept regarding GDPR principles and it is not enough to ensure strict compliance with the GDPR, presenting several shortcomings. GDPR principles involve much more than simple access to data, including the concepts of data subject's consent, storage limitation, data ownership, amongst others. Therefore, although Qapla allows data privacy policies to be defined, such as GDPR policies, it does not ensure GDPR compliance by default. It is still required that the developer study the regulation, make its own interpretation and then try to transform the presumed principles into Qapla policies, based on database identifiers, making it very difficult to ensure full compliance with the GDPR. As the policy specification is very technical and oriented towards the database implementation, it is very user unfriendly and unsuitable to be used by Data Protection Officers.

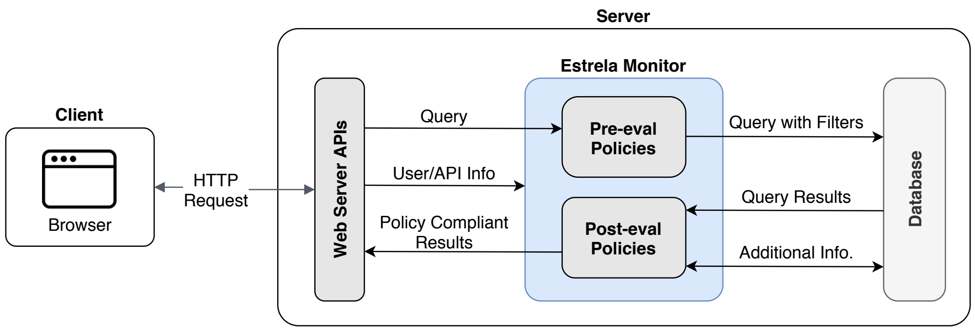


Figure 2.4: Estrela's Architecture.

Estrela

Estrela [35] is a framework built for enforcing contextual and granular data access policies. It ensures compliance with data protection laws and regulations, focused on database-backed applications. It leverages the notion of contextual data flow policy by allowing the enforcement of policies based on the data access context and the application's state. It follows the principle of separation between policy enforcement and application development. Estrela policies are specified alongside the database schema. They can be classified as *pre-eval* policies or *post-eval* policies, and have the following structure:

$$t_i.f_i, ..., t_n.f_n; \varphi; \mathbb{A} : \mathbb{P}$$

$\mathbb{P}$ describes the executable code to be applied when the policy is enforced, that can either rewrite the query or filter the result. Since policy enforcement depends on the tables and fields that are queried, policies are based on them: $t_i.f_i$ refers to a column identifier concerning a database schema, and the policy $\mathbb{P}$ is executed when these fields are accessed. $\mathbb{A}$ is an optional API identifier that allows developers to specify API-specific restrictions. This is what allows the policy to use application-specific information, as different APIs may access the same data under different circumstances, and therefore need to enforce different policies. To specify whether the policy is applied before or after the query is executed, φ describes if the policy is a *pre-eval* policy or a *post-eval* policy, respectively. This pre and post condition evaluation allows to restrict database access before the query is issued and to impose finer grain constraints on the results, after the query evaluation.

Figure 2.4 describes Estrela's architecture. It is focused solely on the policy enforcement mechanism. Before the data is retrieved from the database, the pre-eval policies are applied and the corresponding filters are employed. Then, the filtered query is executed in the database and the post-eval policies are applied, modifying the query result according to the imposed constraints.

Estrela tackles some of the same issues as Qapla, but follows a different approach by allowing the specification of per user preferences instead of imposing global policies that affect all users. As Riverbed, users can express their preferences regarding the processing of their sensitive data, but it is also possible to define global policies to be enforced by all users. It also leverages some interesting techniques for enforcing contextual and granular policies concerning data protection regulations. However, concerning the goals of our work, Estrela provides the same shortcomings as Qapla.

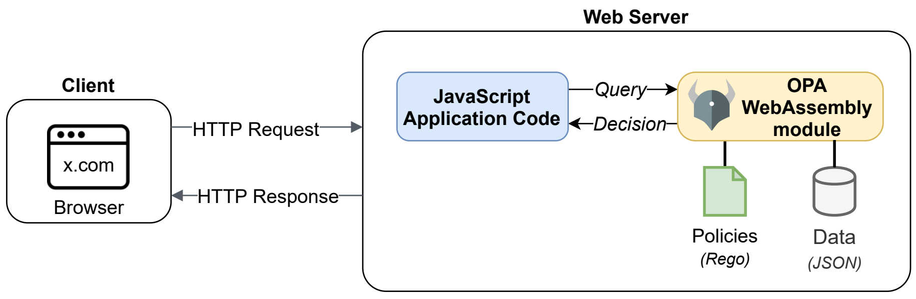


Figure 2.5: Open Policy Agent's architecture as WebAssembly runtime.

Open Policy Agent

Open Policy Agent (OPA) [36] is an open source and lightweight policy engine that unifies policy enforcement across different runtime environments. It makes policy decisions detached from the policy enforcement mechanism and can be integrated as a library and evaluated in the same system process. Its policies are written in Rego (described in Section 2.3.3) and its decision is based on arbitrary hierarchical structured data (JSON), independent of the domain model. It was designed from scratch with latency-sensitive applications in mind, enforcing policies with minimal performance impact.

OPA allows to make context-aware policy decisions by storing base documents containing data that can support the policy decision. OPA exposes domain-agnostic APIs that can be called by any service to manage and enforce policies. It provides two main interfaces: **Evaluation**, for asking for policy decisions and **Management**, for deploying policies, checking status and logs, amongst others. OPA supports the evaluation of policies through a REST API, Go API and WebAssembly runtime. Figure 2.5 describes OPA's architecture, integrated as WebAssembly runtime in a web application. Whenever a service needs to make an authorization decision, it queries the OPA Policy Engine, which accepts structured data (JSON) as input, describing the request. It then evaluates the query input against the policies and the stored data (also in JSON) and returns a JSON decision. This decision can be a boolean, describing the allow/deny answer, or an arbitrary structured data.

Although the OPA stored data and the Rego policies can encompass both user preferences regarding their data and global policies defined by the DPO, it is not oriented towards the GDPR. As Estrela, it is still required that the developer studies the regulation, makes its own interpretation and then attempts to transform the presumed principles into Rego policies. However, Rego policies are much more simple than Estrela policies and support structured document models (as *JSON*), a valuable feature when building web applications.

2.4.3 Discussion

We have also presented some prior work regarding policy-enforcement systems for web platforms, mostly employing information flow control and its variants or mandatory access control. This study provided some context and meaningful insights concerning relevant mechanisms to allow the enforcement of end-to-end security policies for web applications. However, apart from Riverbed, the mentioned systems are not specifically geared towards the application of GDPR policies.

Systems such as SIF, Fabric, or Hails use IFC techniques for enforcing fine-grained security policies. However, these systems presented some major drawbacks. First, the security policies must be specified alongside the application code, implicating that developers need to change their web application to fit the policy enforcement. This does not enforce our maintainability goal of detaching the application from the policies' enforcement and allowing the policy enforcement to be added to any 3-tier web application. Riverbed, also based on IFC but unlike previous systems, does not require developers to manually annotate code with labels or specify security policies alongside the application code. However, Riverbed only allows user-defined policies regarding their own personal data, meaning that a DPO is unable to specify the organization's security policy to be enforced by all users at the same time. In addition, its policy language is not expressive enough to describe an organization's GDPR policy.

As for mandatory access control systems, Qapla provides fine-grained access control in database-backed application. It allows the specification of policies that restrict the information users are allowed to access and then rewrites the application queries based on such policies. Estrela follows a different approach, leveraging some interesting techniques for enforcing contextual and granular policies concerning data protection regulations. However, both these systems raise the same challenges: poor policy management and maintainability, low usability by DPOs, and are also not compliant with the GDPR by default. Lastly, OPA provides a lightweight policy engine for a unified enforcement of Rego policies. However, it does not provide mechanisms to ensure proper compliance with all the GDPR requirements (such as data subject rights) and is not compliant with the GDPR by default. All these limitations have prompted us to develop RuleKeeper.

Summary

This chapter introduced some background on the GDPR and presented the state of the art focused on the GDPR and the enforcement of end-to-end security policies for web applications. Some recent studies begin to investigate the best practices and limitation in GDPR compliance, illustrating the challenges of retrofitting existing systems to abide by the GDPR, the implications for system designers, and the ensuring performance issues. Although some of the researched security policy specification languages and policy-enforcement systems for web applications tackle some of the aforementioned limitations, none of them can fully ensure full compliance with the regulation.

To built a system that allows the specification of data protection policies in accordance with the GDPR by default, we first need to fully understand the implications of GDPR compliance in information systems. The next chapter provides a detailed analysis on the regulation principles and lays down a set of principles that organizations must follow when managing their information systems. This analysis will set the base for the further work introduced in the subsequent chapter.

Chapter 3

GDPR Implications on System Design

The GDPR requires that many organizations adapt their information systems to process personal data in compliance with these regulations. However, existing information systems (e.g., for running intranets and website portals) are not typically GDPR-compliant by design. The web development frameworks that are commonly used for implementing them tend to be agnostic to the GDPR and offer limited native mechanisms to support a seamless development of GDPR-compliant web applications. Enforcement of the GDPR is further complicated by the fact that some of the guidelines laid out by the EU regulations lie in grey areas and are prone to subjective interpretation. Motivated by these challenges, this chapter presents an extensive requirement analysis of the GDPR. We start by providing an overview of our study and then present our synthesis of the key requirements imposed by the GDPR.

3.1 Study Overview

Information systems play an essential role in organizations as their purpose is to support decision making, coordination, control, analysis, and visualization in an organization [37]. They are described as combinations of hardware, software, and telecommunication networks that people build and use to collect, create, and distribute useful data, typically in organizational settings [38]. Our goal is to analyze the data protection requirements imposed by the GDPR and study the impact that these requirements have to the design of information systems. In particular, we focus on 3-tier web-based information systems. Next, after further characterizing these systems, we present an overview of our analysis.

Information systems model: 3-tier architecture is a well-known software application architecture that divides the application logic into three tiers: the *presentation* tier, the *logic* tier and the *data* tier. In a typical system model of a 3-tier architecture, the user interacts with the application through the **presentation tier** (user interface). The heart of the application, the **logic tier**, is responsible for processing the data stored and managed by the **data tier**, using the business logic. The data stored in the data tier is structured in databases, organized via a schema, and managed by a DBMS.

To further improve the software development experience, web frameworks provide a set of programming abstractions based on software patterns, such as the MVC pattern [39]. The main difference be-

No.	Articles	Requirement	Clarity	Observations
C01	*5.1 b)*	Purpose Limitation	Well defined	Purposes must be the same for all data subjects and its collection should be discriminated
C02	*5.1 c)*	Data Minimization	Well defined	It should be assessed which data is in fact needed based on the purposes of the processing
C03	*5.1 a); 6*	Lawfulness of Processing	Well defined	Performance of a contract should be associated with specific purposes
C04	*5.1 a); 12*	Transparency of Processing	Well defined.	The elaboration of a clear and complete privacy policy is very challenging
P01	*5.1 e)*	Storage Limitation	Ill-defined	It is not specified any period or criteria to delete or mark data as expired; It is also not explained when to consider that a purpose has ended
P02	*5.1 d)*	Accuracy Preservation	Ill-defined	It is not defined clear criteria to evaluate data's accuracy and a deadline to fix data's inaccuracy
P03	*5.1 d); 30; 33; 34*	Accountability	Ill-defined	It is not specified which activities must be monitored and which mechanisms can be used to achieve it
P04	*5.1 f); 32*	Security of Processing	Ill-defined	It is not specified how to calculate the appropriate risk and lacks information on how to enforce any type of the required security
S01	*15*	Right to Access	Well defined	Its exercise depends on the organization's policy
S02	*16*	Right to Rectification	Well defined	
S03	*17*	Right to Erasure	Well defined	
S04	*18*	Right to Restriction of Processing	Well defined	
S05	*20*	Right to Data Portability	Well defined	
S06	*21*	Right to Object	Well defined	
S07	*22*	Right to Withdraw Consent	Well defined	

Table 3.1: Principles that organizations must follow when managing their information systems.

tween a 3-tier architecture and a MVC architecture is that in the first one all communication goes through the logic tier, and in the second one, the communication is triangular. From a historical point of view, full-stack web frameworks started to become popular, starting with the *LAMP stack* and culminating with frameworks such as *MEAN/MERN stack* and *Ruby on Rails*[1].

There are several types of users that can interact with the system through the presentation layer, locally or remotely. These users can be internal or external users and their interaction with the system is based on their role in the organization via access control mechanisms. Users are then managed by a system administrator, which is a privileged user that carries out tasks that require special permissions, such delegating the users' roles and managing the access control. Information systems can also be composed of many external systems, such as backup systems, logging systems, amongst others.

A synthesis of the key GDPR requirements: We provide a detailed analysis of the regulatory requirements to fully understand the implications of GDPR compliance in information systems. Table 3.1 presents a synthesis of the key requirements that we have identified by extensively analyzing the articles of the GDPR. In total, we identified 15 general requirements which we have organized in accordance to the party that we deem to be mostly related to them: *Data Controller* (Cx) and *Data Processor* (Px) re-

[1]LAMP stack stands for its main component technologies: **L**inux, **A**pache, **M**ySQL, and **P**HP/**P**erl/**P**ython. MEAN stack stands for **M**ongoDB [40], **E**xpress [41], **A**ngular [42] and **N**ode.js [43]. MERN is similar to MEAN, but with **R**eact [44] instead of **A**ngular.

quirements highlight the responsibilities of the data controller and data processor, respectively, and *Data Subject* (Sx) rights concerns the rights that must be facilitated to data subjects. For each requirement, the table indicates the GDPR articles that support it, our subjective assessment in terms of its clarity, and a few relevant observations. We believe this list offers a comprehensive set of guidelines that can be used beyond the scope of this thesis to help identify open research challenges in GDPR compliance.

Next, we present the three groups of GDPR requirements that we identified. For each requirement, we provide a summary of the related GRPR articles, an explanation of our interpretation of those articles, and our analysis regarding the level of information systems' compliance with the requirement in question.

3.2 Data Controller Responsibilities

The data controller is responsible for collecting personal data and determining the purposes and means of its processing (e.g., *web services*), as described in **Art. 4** of the **GDPR** [3]. According to **Art. 24** *(Responsibility Of The Controller)* of the **GDPR** [3], the data controller must implement appropriate technical and organizational measures to ensure and to be able to demonstrate that processing is performed in accordance with the regulation. These measures include the implementation of appropriate data protection policies by the controller. In this section, we describe the set of principles that the organizations' data controller must follow when managing its information systems.

C01: Purpose Limitation

> **Art. 5** *(Processing of Personal Data)* of the **GDPR** [3] states that personal data should only be *"collected for specific purposes and should not be processed in a manner that is incompatible with those purposes"*. Additionally, these purposes should be explicit and established when personal data is being collected.

Interpretation: The scope for which personal data is collected or processed must be constrained to well-defined and explicitly-stated purposes. For example, when a user provides its email to an entity to subscribe to a newsletter, that email *(personal data)* should be associated with a purpose *newsletter* and not be processed for any other incompatible purpose, such as creating an account in a website. The data controller must define which purposes exist and the data they need to collect.

Challenges: This article does not clarify if the purposes must be the same for all data subjects, which leads to a sophisticated issue. If the collected purposes are generic, the controller is unable to change the purposes at any point, given that would result in different purposes for different data subjects. This also limits the freedom of the data subjects, since either they agree to all collected purposes or they do not agree to any, preventing them to allow the processing of their data for only one specific purpose.

Compliance in information systems: Just as there is no concept of personal data, there is also no concept of purpose. In information systems, data is used as resource. Although it can be collected for specific purposes by a given organization, information systems are able to use the already collected data for incompatible purposes. An example of such a violation is the administrative fine of 18€ million that the Austrian Data Protection Authority imposed against Österreichische Post AG, a postal service in Austria, for using their costumers' data (collected for postal service purposes) to calculate the probability of their political affiliation and then using that information for targeted marketing [45]. In general, purpose limitation is not natively supported by existing web development frameworks. It must be enforced by the application logic and implemented by the web application developer.

C02: Data Minimization

> **Art. 5** *(Processing of Personal Data)* of the **GDPR** [3] states that personal data should be *"adequate, relevant, and limited to what is necessary in relation to the purposes for which they are processed"*. However, this principle does not apply to purposes such as public interest, scientific or historical research purposes, or statistical purposes.

Interpretation: Each purpose should be associated with the minimum data necessary to fulfill it, limited to the necessary and relevant personal data. This requirement complements C01, as it restricts the blind and careless association of personal data to the corresponding purposes. To be compliant with this requirement, the data controller must specify which personal data item is required for each purpose.

Compliance in information systems: Just as information systems do not have a concept of purpose, they are also unable to identify the minimum data necessary to achieve a given purpose. For example, many systems have web forms in their applications that allow users to create accounts. They can ask anything in those forms, even if they do not need it. As so, unless implemented otherwise by the web application developers, web application frameworks by default do not limit the amount of data that applications can process, and hence these do not typically abide by the data minimization requirement.

C03: Lawfulness of Processing

> **Art. 5** *(Processing of Personal Data)* and **Art. 6** *(Lawfulness of Processing)* of the **GDPR** [3] state that the data processing is lawful in the following cases: when *"the data subject has given its consent for one or more specific purposes, when it is necessary for the performance of a contract, to protect vital interests, or if it is relevant to the public interest"*, amongst others.

Interpretation: For the data processing to be lawful, it needs to meet at least one of the conditions stated above, i.e., it needs to have a *lawfulness base*. This lawfulness base can be described as *the valid lawful reason to process the personal data and without one the processing is not lawful*. This implies that purposes must have a lawfulness base (i.e., a valid lawful reason to process the personal data) and that the data controller is responsible for defining the lawfulness base associated with each one. When the lawfulness base is *consent of the data subject*, then the data controller also needs to collect the data subject's consent before it can collect and process the data in question.

Challenges: There is a subtle difference in two specific lawfulness bases: *(consent of the data subject* and *performance of a contract).* They both depend on the data subject's input. However, "performance of a contract" (which is widely employed by organizations) is not associated with any specific purpose, making it seem like, when the data subject performs a contract, it is implicitly consenting to all purposes of data processing, which may not be the case. For instance, a clinical analysis laboratory uses "performance of a contract" as the lawfulness base to manage the patients' analysis. The laboratory also has a monthly newsletter. A given data subject should be allowed to withdraw its consent for the newsletter while agreeing on the contract that authorizes medical data collection and processing for clinical analysis purposes. To allow for such flexibility, a contract should be associated with a set of purposes, which would then be implicitly consented when the data subject performs the contract. This way, we give the data subject the freedom to give and withdraw its consent to purposes that are not bound by a contract.

Compliance in information systems: For the data processing to be lawful, it needs to have a valid lawfulness base and in some cases, it is needed the data subject's consent. By default, information systems do not check the source of the data, let alone check for consent. Existing web development frameworks also do not provide native support to obey the lawfulnss of processing requirement.

C04: Transparency of Processing

> **Art. 5** *(Processing of Personal Data)* of the **GDPR** [3] states that personal data shall be *"processed lawfully, fairly, and in a transparent manner in relation to the data subject"*. **Art. 12** *(Transparent information, communication and modalities for the exercise of the rights of the data subject)* of the **GDPR** [3] tells that *"the controller shall take appropriate measures to provide any information referred to in* ***Arts. 13*** *and* ***14****, and any communication under* ***Arts. 15*** *to* ***22*** *and* ***34*** *relating to processing of the data subject in a concise, transparent, intelligible and easily accessible form, using clear and plain language"*.

Interpretation: This principle advises that individuals should be informed about the collection and use of their personal data. In order for organizations to fulfill this principle, it is necessary to elaborate a clear, concise, and complete *privacy policy* that discloses the ways the organization gathers, uses, discloses, and manages a customer or user personal data.

Challenges: Although it seems very simple to comply with, this requirement is the source of many fines, as many organizations develop privacy policies with several deficiencies [5]. For example, the French data protection authority fined Google 50€ million for not properly informing users how their data was collected across its services in order to present personalized advertisements [46].

Compliance in information systems: According to this requirement, the data subject needs to be informed about the collection and use of their data. The medium to convey this information can be the elaboration of a privacy policy. In typical information systems, the privacy policies – whenever they exist – are normally elaborated in an ad-hoc fashion and written in english. There is no native support by existing web development frameworks to help application developers to specify these policies let alone to automatically enforce them at runtime while the application processes the data subjects' data.

3.3 Data Processor Responsibilities

According to **Arts. 4** and **28** *(Processor)* of the **GDPR** [3], the data processor must process the personal data on behalf of the controller, while relying only on documented instructions from the controller (e.g. *cloud providers*). In this section, we describe the set of requirements that the organizations' data processor must follow when managing GDPR-compliant information systems.

P01: Storage Limitation

> **Art. 5** *(Processing of Personal Data)* of the **GDPR** [3] states that personal data shall be *"kept in a form which permits the identification of data subjects for no longer than is necessary for the purposes for which the personal data are processed"*.

Interpretation: Personal data should only be stored while it is necessary to fulfill the purpose for which it was collected. Therefore, when the data is no longer necessary, the processor must erase it without further delay. Interestingly, to obey this requirement, the processor requires specific input from the data controller. In particular, the data controller must define for how long it is necessary to retain the data, which will depend on the specific purposes for collection and processing (domain-dependent). Although the controller is responsible for defining such deadlines, we attribute the responsibility for this requirement to the processor since it must put in place the technical measures for compliance.

Challenges: This requirement is only broadly defined by the regulation, which leaves several unanswered questions, such as: (i) How soon should the *"expired"* data be deleted? (ii) When should it be considered that a purpose has ended? (iii) How soon should the data be marked as expired? Not only the GDPR omits any details on how to specify the time frames / conditions for storage limitation, but it is complex to define the concept itself of *purpose deadline*. The *"expired"* data can be deleted in real-time, synchronously, or it can be deleted in batches within a specific period (e.g every hour). The data can also be marked as expired as soon as it expires, in a synchronous way, or marked in batches. Performing these operations synchronously raises the need of considerable amount resources. Specifying a time period may create inaccuracies as it implies arbitrarily guessing how long that period will be can be regardless of potential contextual conditions that might affect it. The data processor has to decide the point in time for marking data for deletion and erasing the expired data, which also involves finding a sweet spot between a three-pronged trade-off: resource efficiency, performance, and full compliance. This article also fails to state the conditions that make the data subjects prone to be identifiable.

Compliance in information systems: Due to the continuous adoption of models such as big data analytics and machine learning, in many systems the data is stored forever. There is no automatic or manual system that forces these systems to delete their data, whether they are using it or not. There is also a wide adoption of storage systems that aim to guarantee better performance, scalability, and reliability. These properties are typically achieved through data replication. These practices make it difficult to erase the data for those who try and impossible for those who do not [9]. To show the extent of this issue, even Google Cloud states that it can take up to 6 months to delete a user's data from all active systems and backups [47]. In short, many systems do not have native support to enforce the storage limitation principle. The same can be said of existing web application frameworks.

P02: Accuracy Preservation

> **Art. 5** *(Processing of Personal Data)* of the **GDPR** [3] states that personal data must be *"accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that personal data that are inaccurate, having regard to the purposes for which they are processed, are erased or rectified without delay"*.

Interpretation: Given that the processor holds the general responsibility for curating the data subject' data, we consider that it should also be the processor to ensure that the personal data it holds is accurate and consistent. If it is not accurate, the processor should erase it or update it accordingly. The inaccuracy of the data depends on the purposes for which it is processed; for example, driver's licence may be considered to be inaccurate if it the expiry date has been reached. The data processor is responsible for determining if the data is accurate and for triggering the mechanisms for fixing any data inaccuracies. It should also specify what should happen in the system if it is not possible to fix the problem.

Challenges: The regulation does not define unequivocally which criteria should be adopted to validate the accuracy of personal data. Moreover, the notion of *accuracy* is highly dependent on the business case. The regulation also omits the definition of specific deadline or criteria for the rectification/erasure of data if it is found to be inaccurate – similar to the omission in the GDPR for storage limitation.

Compliance in information systems: Apart from the verification checks implemented by the application logic, typical information systems do not have built-in mechanisms that verify the accuracy of the data. For example, if a web application collects personal data that consists of current preferences of European citizens to use for a set of statistics in 2016, it is able to reuse the same data for some new statistics in 2020. This data is probably not accurate, but the system cannot automatically verify it. Even if the system was able to verify the accuracy of the data, it would not, by default, delete or update that data according to the indications of the GDPR.

P03: Accountability

> **Art. 5** *(Processing of Personal Data)* of the **GDPR** [3] states that controller shall be *"responsible for, and be able to demonstrate compliance with the regulation"*. **Art. 30** *(Records of processing activities)* complements it by affirming that each controller/processor shall *"maintain a record of processing activities under its responsibility"*. Regarding external communication, **Art. 33** *(Notification of a personal data breach to the supervisory authority)* and **Art. 34** *(Communication of a personal data breach to the data subject)* both state that when the personal data breach is *"likely to result in a high risk to the rights and freedoms of natural persons, the controller shall communicate the personal data breach to the data subject and supervisory authority without undue delay"*.

Interpretation: Art. 5 and **Art. 30** aim to ensure that controller is accountable for what it does with personal data and on how it complies with the other GDPR requirements. The controller must keep an account of its own processing activities and document any required records that demonstrate compliance to the GDPR, e.g., records of data subjects' consent. **Arts. 33** and **34** are straightforward since they only refer the obligation to notify the authorities and data subjects in the event of personal data breaches. A personal data breach can be defined as a security incident that has compromised the confidentiality, integrity, or availability of personal data, such as cyber attacks (e.g ransomware), unauthorized access, stolen devices, and accidental hazards. To detect these events and allow for internal audits, the processor needs to deploy adequate monitoring systems that can supervise and keep curated logs of all meaningful activities related to personal data processing. For this reason, although the controller is also responsible for notifying the authorities of personal data breaches, we attribute the main responsibility for upholding the GDPR accountability requirement to the processor.

Challenges: Although **Arts. 33** and **34** seem rather simple to interpret, their implementation implicitly involves the deployment of an extensive and possibly quite complex backend structure. For instance, to detect an unauthorized access, the processor needs to keep logs of all of the accesses with the necessary information to evaluate if those accesses were authorized or not. Given that the GDPR does not describe the level of detail at which the monitoring activities must be carried out, the processor must decide for itself which activities should me monitored and which should be ruled out. Such a decision is not trivial as an increase in the coverage of monitored events tends to entail a degradation in performance and an increase in resource consumption [8].

Compliance in information systems: While some information systems are already equipped with some logging capability, it is difficult to assess to what extent its native mechanisms suffice to cover all the accountability needs in the context of the GDPR. For instance, it may be necessary to keep finer grained logs for operations that handle personal data than for operations that process less sensitive data. Existing systems rarely include dedicated mechanism for notifying data breaches. On top of that, many organizations tend to handle data breaches with little transparency and questionable practices, e.g., dealing with a personal data breach or a privacy violation by concealing it or falling prey to data extortionists. For instance, Uber kept a data breach, where 57 million accounts were stolen, secret for more than a year after paying a $100,000 ransom [48].

P04: Security of Processing

Art. 5 *(Processing of Personal Data)* of the **GDPR** [3] states that the personal data should be *"processed in a manner that ensures its appropriate security, including protection against unauthorized or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organizational measures, in a timely manner"*. **Art. 32** *(Security of Processing)* of the **GDPR** [3] states that those measures may include *"pseudonymisation and encryption of personal data, the ability to ensure the ongoing confidentiality, integrity, availability and resilience of processing systems and services and the ability to restore the* *availability and access to personal data in a timely manner in the event of a physical or technical incident"*.

Interpretation: The data processor should take on appropriate measures to ensure the security of personal data. These measures should prevent at least the issues enumerated by said articles, such as unauthorized access, unlawful processing, loss of data, destruction of data and damage of data, ensuring a level of security appropriate to the risk.

Challenges: Data security is one of the most important aspects involving the protection of personal data, and yet one of the most vague requirements described in the GDPR. The data processor needs to deploy a cross-cutting security infrastructure that can mitigate each of the risks stated above. For instance, to protect the data from unauthorized access, the processor should have an access control system that implements a set of authorizations and/or restrictions for the organization. Regarding data loss, the organization depends on an appropriate and fully functioning backup system. The DPO should define the frequency of the backups and what data they should contain. However, such a wish-list is quite extensive and requires considerable investments from the organizations to keep the systems protected against constantly evolving security threats. The security of processing is one of the top-priority requirements for GDPR compliance.

Compliance in information systems: Despite the increasing number of security mechanisms already provided by many web development frameworks (e.g., support for HTTPS), information systems tend to be optimized for performance, cost, and reliability, leaving aside the security as a secondary goal. The GDPR states that the organizations must implement appropriate technical and operational measures to ensure a level of security appropriate to the risk. This requires a constant evolution and adaptation of their systems which depend on many factors, such as the state-of-the-art security defenses, their cost of implementation, and the level of risks and impact to the rights and freedoms of data subjects.

3.4 Data Subject Rights

Art. 12 *(Transparent information, communication and modalities for the exercise of the rights of the data subject)* of the **GDPR** [3] states that the *"controller shall facilitate the exercise of data subject rights under **Arts. 15** to **22**"*.

Interpretation: Natural persons should have the control of their own personal data. To achieve that, the GDPR advises the controller to facilitate the exercise of the data subjects' rights, including the provision of mechanisms that the data subject can leverage to exercise its rights and, if applicable, obtain, free of charge, an appropriate response from the controller.

S01: Right to Access: Art. 15 *(Right of access by the data subject)* of the GDPR [3] states that the data subject must be given the right to know if his personal data is being processed, and, when that is the case, it should also be given the right to access the personal data concerning him and additional information such as: the categories of the personal data, the purposes of the processing, amongst others. The controller must grant a copy of the personal data undergoing processing.

S02: Right to Rectification: Art. 16 *(Right to rectification)* of the GDPR [3] states that the data subject must be given the right to complete and to rectify inaccurate or incomplete personal data concerning him, without unreasonable delay.

S03: Right to Erasure: Art. 17 *(Right to erasure - "right to be forgotten")* of the GDPR [3] states that the data subject must be given the right to obtain from the controller the erasure of personal data concerning him, without unreasonable delay, as long as some conditions apply, such as the personal data cannot be necessary regarding the initial purposes to which it was collected and/or processed. The controller must erase the data in an irreversible way, ensuring all the links to it and all the copies and replicas are also erased.

S04: Right to restriction of processing: Art. 18 *(Right to restriction of processing)* of the GDPR [3] states that the data subject must be given the right to obtain from the controller restriction of processing regarding his personal data in some cases, such as when the accuracy of the personal data is contested by the data subject.

S05: Right to Data Portability: Art. 20 *(Right to data portability)* of the GDPR [3] states that the data subject must be given the right to receive the personal data that he provided to the controller, in a structured, commonly used and machine-readable format and to transmit those data to another controller.

S06: Right to Object: Art. 21 *(Right to object)* of the GDPR [3] states that the data subject must be given the right to object, for reasons regarding his particular situation, at any time to processing of personal data concerning him, including profiling based on those provisions and direct marketing.

S07: Right to withdraw consent: Art. 7 *(Conditions for consent)* of the GDPR [3] states that the data subject must be given the right to withdraw his consent at any time and should not affect the lawfulness of processing based on consent before its withdrawal. In addition, it must be as easy for the data subject to withdraw consent as it is to give it.

Challenges: The exercise of data subject rights depends on the organization's policy and cannot be executed in the same way for all organizations. Requests can be declined by the controller, if justified. So, without the input of a data controller, it is not possible to fulfill most requests. For instance, Google presents an issue with the *Right to Be Forgotten* right, where they describe their process to decide which URLs to delist as manual [6]. Thus, organizations must specify in their policy how the system should implement such rights, stating which rights can be exercised automatically, without the data controller's involvement and which ones must go through the data controller for approval.

Compliance in information systems: Data subject rights can only be exercised via an explicit request from the data subject, and in case of an information system, this request must be conducted by an operation implemented in the system. However, typical information systems do not allow the data subject to access, rectify and erase its data. Web application frameworks also do not provide native support for such features. Thus, unless these features are implemented by the web developer and incorporated into the application, data subjects do not have appropriate means for exercise their rights.

Summary

This chapter provided a detailed analysis on the regulation principles to fully understand the implications of GDPR compliance in information systems. Unfortunately, not only some of them lie in grey areas and are up to interpretation but also are not implemented by default in information systems, raising a lot of challenges for organizations who want to be compliant with the regulation. To mitigate these challenges for organizations, we lay down a set of principles that organizations must follow when managing their information systems. With this set of principles and challenges in mind, the next chapter introduces RuleKeeper, a web framework that allows the development of GDPR-compliant web applications.

Chapter 4

System Design

This chapter introduces RuleKeeper, a web development framework that allows the development of GDPR-compliant web applications. In particular, it implements a new access control model that allows for the enforcement of declarative GDPR-aware policies. Section 4.1 delineates the fundamental goals and requirements that motivate the design of RuleKeeper and presents the assumptions and threat model underlying our system's design. Section 4.2 presents RuleKeeper's architecture, and Section 4.3 the Purposeful Data Object model – the central abstraction of our system. Section 4.4 describes the programming model of our framework, and Section 4.5 the formal language for the GDPR policies specification. Lastly, Section 4.6 explains RuleKeeper's policy lifecycle and enforcement mechanisms.

4.1 Design Goals and Threat Model

The goal of RuleKeeper is to allow the specification of data protection policies in accordance with the GDPR and hard-guarantee the enforcement of such policies throughout the entire application lifecycle so as to help prevent privacy breaches, increase transparency, and hand over control of data to its owner.

RuleKeeper is designed to be integrated into 3-tier information systems. In the design of our system, we consider the following three main requirements that it must be able to accomplish:

GDPR-policy compliance: Personal data processing must be restricted according to a privacy policy that expresses constraints imposed by the GDPR and the organization's DPO. It must provide a policy specification method that can accommodate the diversity of organizations' policies and be expressive enough to take into account all major actors and data handling restrictions contemplated in the GDPR. Given that the breath of requirements dictated by the GDPR is very broad (see Section 3.1), in the scope of this thesis we focus primarily on the enforcement of access control policies that satisfy the restrictions in personal data processing ruled out by the GDPR.

Maintainability: The policy enforcement must be detached from the application code and allow for the separation of cross-cutting concerns. Application code should be easy to manage by a team of multiple developers, and policies easy to specify and maintain by non-experts in computer engineering.

Good end-to-end performance: The incorporation of the necessary logic required for the validation and enforcement of policies should not significantly slow down the performance of web applications. In particular, our framework should deliver comparable performance to modern web frameworks.

We make several assumptions about the surrounding actors and underlying execution environment of our system. First, we consider the existence of a dedicated person – the operator – which is responsible for the specification of the privacy policy to be enforced by a given web application. This policy is expected to reflect the specific terms of the GDPR regulation that apply to the concrete organization running the web application. We assume that the policy correctly expresses the data protection measures that are expected to be implemented.

The web application developers write the code of their applications using the programming abstractions offered by our framework. This code is untrusted in the sense that the developer may accidentally introduce bugs and vulnerabilities in the application that can potentially lead to the violation of the privacy policy. However, the developer is not considered malicious. This means that the developer can be trusted to be using only the operations considered to be safe within the context of our framework; thus, any potential errors in the application code may arise exclusively from the misuse of such operations.

As for the execution environment, we trust the correctness and integrity of the browser's runtime, the security of communication channels between client and server, and the correctness and integrity of the server-side OS, web server, and DBMS. Our system is built as a trusted cross-cutting middleware to run on top of a typical web application infrastructure comprising these components.

4.2 Architecture

RuleKeeper is a web development framework for 3-tier web applications which provides a system for enforcing compliance with GDPR data protection policies. Figure 4.1 illustrates a deployment of our system that supports the execution of a web application in the context of an organization. As an example of such an organization, consider a laboratory that offers a clinical analysis service and it is responsible for processing and managing health-related information and testing biological specimen (e.g., blood or urine). This example will be adopted as our real-world use case, allowing us to perform a real-world validation of our system. We proceed to enrich this chapter with pertinent examples concerning the laboratory use case. The diagram in Figure 4.1 can then be assumed to illustrate the components of the laboratory's information system developed using the RuleKeeper web development framework.

RuleKeeper system is composed of two key components: a *middleware* and a *manager* service. The middleware consists of a set of libraries linked to the web application, which export an API that allows the interaction between the web application and the GDPR policy enforcement system. The manager service runs in a centralized management server, and it is responsible for managing and coordinating the GDPR data protection policies, which it then shares with the middleware. The middleware uses these policies to enforce GDPR compliance as the web application collects and processes personal data. We envision having additional integration with external systems to provide supplementary functionality, such

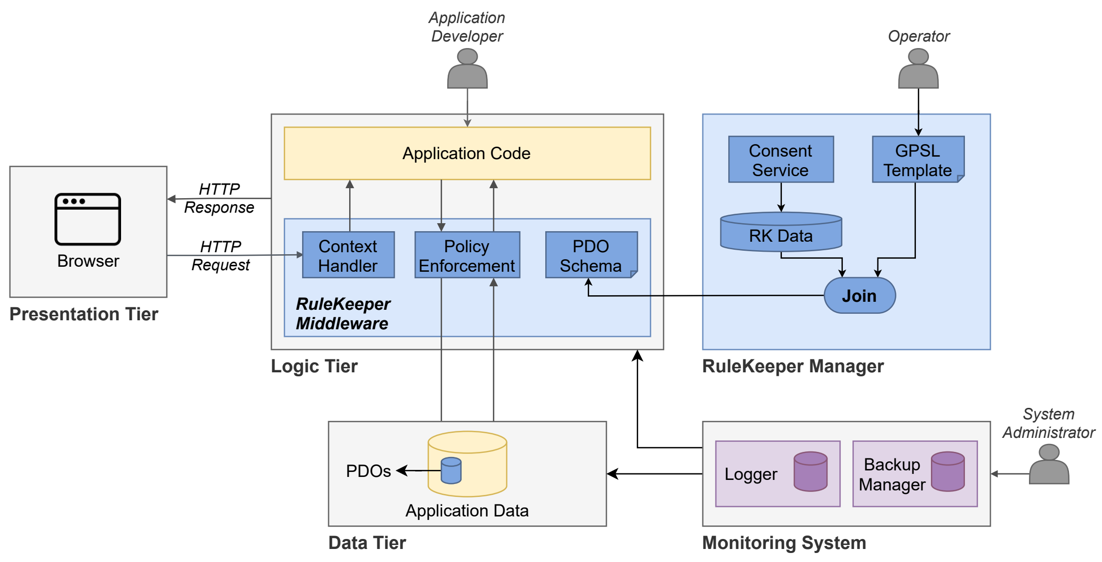


Figure 4.1: RuleKeeper architecture. Yellow boxes represent components specific to the application; blue boxes refer to RuleKeeper's components; and purple boxes pertain to external systems.

as logging or monitoring services. Considering the scope of this thesis, such services does not pertain to the core functionality of RuleKeeper.

There are two main actors that interact with RuleKeeper: the *application developer* and the *operator*. The application developer uses the middleware to develop and deploy GDPR-compliant web applications. Application development includes the specification of the data model, the implementation of the application operations, and the specification of an access control model for the principals that interact with the application. The operator is a person designated by the host organization which is responsible for managing the system and ensuring that the web application abides by the GDPR requirements. The operator receives input from the application developer concerning the web application's architecture (e.g., database model), and from the Data Protection Officer regarding the specific implementation of the GDPR for the organization, including details such as the specification of which personal data is processed by the organization.

RuleKeeper operates using **Purposeful Data Objects** (PDOs). PDO is a new abstraction that models the web application's state so as to satisfy GDPR restrictions. It includes data model properties, application code aspects, and the binding of such abstractions to the web application concerning the restrictions imposed by the GDPR, such as purposes and data minimization limitations. The system maintains a specification of this abstraction, the **PDO Schema**. This schema is built from two sources. The primary source is from a declarative policy specified by the operator in the GDPR Policy Specification Language (GPSL). GPSL is a simple domain specific language that we have developed to express GDPR policies for a given organization. The second source is some additional information that needs to be obtained at runtime, such as the data subjects' consent. The PDO abstraction allows the middleware to enforce the GDPR policies, without requiring the developer's direct involvement.

The following sections detail the purposeful data object model, the application development process, coordinated by the application developer, and the policy specification, carried out by the operator.

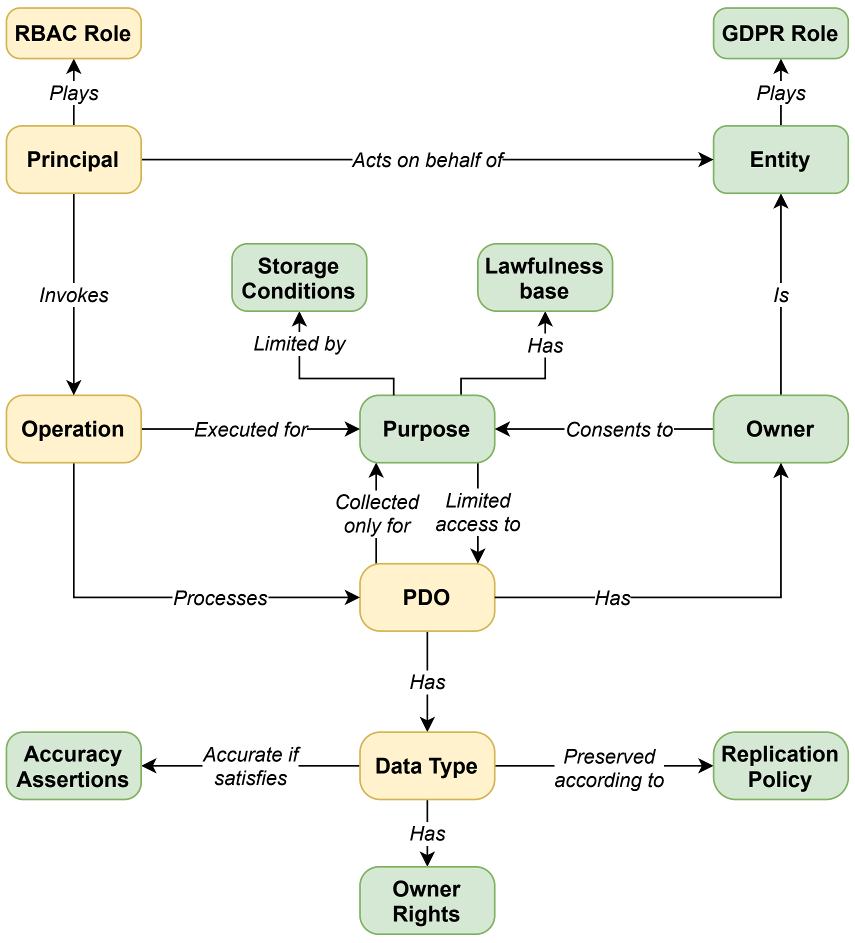


Figure 4.2: Purposeful Data Object model.

4.3 Purposeful Data Objects

Our analysis in the previous chapter concluded that the GDPR imposes a strict set of requirements when processing data. To accommodate such requirements when building information systems, we present a new abstraction, named **Purposeful Data Objects** (PDOs), which allows to model the web application's state, covering the required information for achieving GDPR policy-compliance, as represented in Figure 4.2. Some of the components presented in this model represent aspects that already exist in typical MVC web applications, and therefore are managed by the developer. These components are represented in the PDO model using yellow boxes. Green components represent the concepts introduced by the GDPR and are managed by the operator.

In an information system, there is a set of users that interact with the web application, which we name **principals**. Principals can have **roles** associated, representing their function and permissions to execute some operations in the system and their identification is done through their login. Using our clinical analysis laboratory use case, a laboratory that provides a web application to add and consult clinical information, the *principals* are the users of that application, represented by their username, and their *roles* are patients, clinical analysts and receptionists. These principals can represent **entities** in the organization's domain, involved in the personal data processing. To each entity is attributed a specific

role from the GDPR, as introduced in Section 2.1.1, describing their role in the personal data processing. For instance, the receptionists and clinical analysts would represent the *entity* laboratory, posing as the data controller and the patients would represent themselves, posing as data subjects. The laboratory patients, as data subjects, consent to one or more purposes regarding the processing of their data, and therefore are **owners** of their data.

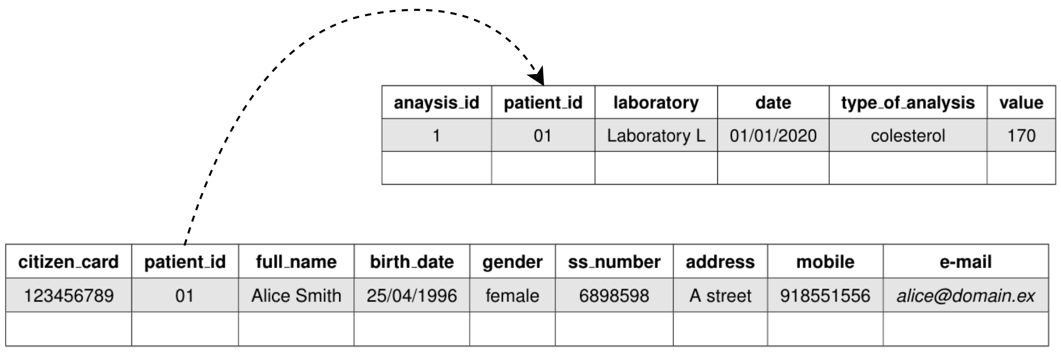
To ensure compliance with the GDPR regulations, information systems must facilitate the exercise of data subject rights. As described in Section 3.4, organizations must specify in their policy how the system should implement the **owner rights**, which includes which data is accessible when fulfilling such right. For our clinical laboratory, one could specify that data subjects could exercise the *Right to Access* automatically, allowing the access to all data owned by each data subject.

PDOs are accessed and processed through system **operations**, which are executed to fulfill a specific purpose. As per the *C01: Purpose Limitation* and *C02: Data Minimization* principles, each PDO is collected for specific purposes and each purpose can only process a **limited** and relevant set of data. Using our use case as an analogy, the PDOs would correspond to the personal and clinical information of the laboratory's patients, such as their insurance company, address and analysis results, which would be collected for the clinical analysis management purpose. The laboratory web application could then provide an operation to access each patient's data, with the purpose of clinical analysis management, and such operation could only process the clinical data associated with that patient, except for additional information produced by the laboratory.

According to principle *C03: Lawfulness of Processing*, each purpose must be associated with a **lawfulness base**, representing the valid lawful reason to process the personal data, which, in the case for out clinical laboratory, would result in associating the purpose *clinical analysis management* with a lawful reason to process such data, as the performance of a contract. Since principle *P01: Storage Limitation* states that personal data should be erased when it is no longer necessary to the purposes for which the personal data was collected, purposes must also be associated with **storage conditions** that restrict the storing of the data collected for that purpose. In our clinical analysis laboratory, one could specify that if a patient does not request a clinical analysis from that laboratory within a 5 year period, it would be determined that such patient is no longer interested in being tested in that laboratory. As so, the purpose of clinical analysis would no longer apply, and its data should be deleted from the system. Lastly, *P02: Accuracy Preservation* mandates that this data must be kept accurate, which is assessed through **accuracy assertions**. Through the use of these assertions, the laboratory could specify that a citizen card would be inaccurate if its date is expired.

4.4 Application Development

As detailed previously, the application developer is accounted for the implementation of the web application, which includes the specification of the data model, the implementation of the application operations and the specification of an access control model. This implementation is agnostic to GDPR abstractions. Next, we present the main steps of the development process of a RuleKeeper-enabled web application.



anaysis_id	patient_id	laboratory	date	type_of_analysis	value
1	01	Laboratory L	01/01/2020	colesterol	170

citizen_card	patient_id	full_name	birth_date	gender	ss_number	address	mobile	e-mail
123456789	01	Alice Smith	25/04/1996	female	6898598	A street	918551556	*alice@domain.ex*

Figure 4.3: Data model of a clinical laboratory that manages patients' personal and clinical information.

```
/* Operation that returns the personal data of patient patientId */
getPatientData(req, res) {
  const { patientId } = req.params;
  logger.info(`Received getPatientData request with id: ${patientId}.`);

  // if it does not contain information about the columns to return, returns all columns
  PatientPersonalInformation.findOne({ patient_id: patientId }, (err, user) => {
   const error = helper.handleError(err);
   if (error) {
    res.status(400).json('Error fetching patient data.');
    return;
   }
   res.status(200).json(user);
  });
 }

/* Route that instanciates the above operation. */
router.get('patients/:patientId', patientsController.getPatientData);
```

Listing 3: Implementation of an operation responsible for accessing a patient's personal information.

Define the application data model: First, the web developer must specify how data of the application domain is organized in the database, by indicating the existing tables and its corresponding columns and data types. Figure 4.3 showcases two tables of the data model that can be used to specify the clinical laboratory patients' personal and clinical data. The table depicted in the bottom of the figure contains information about each patient: the citizen card, a patient ID, full name, birth date, gender, social security number, address, mobile phone, and e-mail address. The other table contains information about the analysis records associated with each patient. As all personal data must be identified with its owner, the patient ID (`patient_id`) identifies the data subject to which the data belongs to.

Implementation of the application-specific operations: Following the specification of the application's data model, the developer must now implement the application operations' logic. Operations are the heart of the application as they are responsible for processing the application data and displaying those results to the end-user. Each operation is implemented by a piece of JavaScript code and is invoked through the submission of HTTP requests to a specific URL. Listing 3 presents the implementation of an application operation that accesses a specific patients' personal data. This operation receives as input the `patient_id` whose data the application wants to access, executes a database query that

`Get Principal()`	`get_principal(string id) ⇒ (string id, string role)` Returns information of a principal.
`Create Principal()`	`create_principal(string id, string role) ⇒ ()` Create a new principal in the system, associated with its role.
`Remove Principal()`	`remove_principal(string id) ⇒ ()` Removes a principal from the system.
`Change Role()`	`change_principal_role(string id, string role) ⇒ ()` Attributes a new role to a principal in the system.

Table 4.1: Principal's management api.

`Get Session()`	`get_session(string principal) ⇒ (context c)` Returns information of a specific session, including its role and the session expiration timestamp.
`Init Session()`	`init_session() ⇒ (context c)` Initiates a new session and creates the new context.
`End Session()`	`end_session(string principal) ⇒ ()` Ends a session, including its context.

Table 4.2: RuleKeeper's session management api.

accesses the data stored in the patient's personal information table (`PatientPersonalInformation`), belonging to that patient and returns that data, if exists. It is available in the application's REST API, mapped to the URL `patients/:patientId`. This example was implemented using Node.js and Express.

Implementation of RBAC access control model: Lastly, not all operations can be executed by all principles. The web application principals must authenticate themselves in the system, to express their role in the application, which represent their function and permissions to execute the application's operations. In the clinical laboratory, such roles include patients, receptionists and clinical analysts, all with different permissions in the application. For instance, a patient is only allowed to access its own data, whereas the receptionists are allowed to access personal information from all patients.

To achieve this, the application developer must implement an access control model to express permissions according to their role in the application. To this end, RuleKeeper provides a dedicated API that allows the developer to create new principals and manage roles and permissions. This API is presented in Table 4.1. Typically, a web application will also include the integration of a privileged principal – the "administrator" – that can invoke operations of said API to manage the principals. This API also allows to restrict the operations allowed to be executed by a principal with a specific role. In a simplified way, this access control mechanism can be implemented explicitly in the application operations, through a set of if-conditions. To be noted that there is a variety of more sophisticated mechanisms that can be used to implement access control in web applications. For simplicity reasons, RuleKeeper provides support only for standard RBAC.

To support this model, RuleKeeper natively provides an authentication mechanism enhanced with a session management API. This API allows the principals of the application to establish a session in the application, which will maintain a context c. This context stores information such as the role of the principal and the session expiration timestamp. Table 4.2 presents this session management API.

4.5 Policy Specification

Following the implementation of the web application, the operator is now responsible for writing a declarative policy that unifies the concepts of the PDO model, through a **GPSL template** provided as input to the RuleKeeper Manager. With this in mind, we present GPSL, a domain-specific policy specification language based on our PDO model. GPSL allows an operator to specify, in a non ambiguous way, the GDPR privacy requirements of the organization in such way that, when interpreted by the RuleKeeper system, the protection of personal data will be ensured to be in compliance with GDPR. Next, after introducing the abstract notation that will be used below to describe the GPSL features, we present its core language primitives and then elaborate on some future extensions to be developed in future work.

Abstract notation: GPSL can be used to specify the organization's privacy policy and the necessary mapping for providing RuleKeeper with the application-dependent context.

$$\mathbb{A} : x_1...y_1, ..., x_n...y_n$$

In this expression, $\mathbb{A}$ represents the PDO model property which the GPSL policy describes and $x_i...y_i$ represent its attributes. We proceed to specify the PDO model using this notation.

4.5.1 Core GPSL Primitives

In this section we present the core GPSL primitives that are fully supported by the current version of RuleKeeper. These primitives are used as the basic language constructs for expressing GDPR-specific access control policies. Next, we enumerate each of these primitives:

Data Types: First we need to describe which data types d_i are classified as personal data. In a web application context, the data type identifier does not hold any value, so, each data type must be mapped to the corresponding table t_i and column c_i in the database.

$$\boxed{\mathbb{D} : d_1.(t_1.c_1), ..., d_n.(t_n.c_n)}$$

Purposes: To describe the purposes involved in personal data processing, we associate each purpose p_i with its lawfulness base b_i and the maximum data d_i they are allowed to process. If the lawfulness base is the explicit consent of a data subject or the execution of a contract, it is required the consent of the data subject.

$$\boxed{\mathbb{P} : p_1.b_1.d_1, ..., p_n.b_n.d_n}$$

Operations: PDOs are accessed and processed through system operations. Each operation o_i is executed with a purpose p_i and to which is only allowed the access to a subset of datatypes D_i. In a web application context, the operation identifier does not hold any value, so, each operation must be mapped to the corresponding url u_i in the web application.

$$\boxed{\mathbb{O} : o_1.p_1.D_1.u_1, ..., o_n.p_n.D_n.u_n}$$

Entities: The entities e_i involved in the personal data processing must have roles r_i associated, denoting their role in the GDPR: *data subject*, *data controller*, *data processor* or *third party*. Entities can either be declared statically in the policy, in the case of data controllers, processors and third parties, or can be mapped to a data table t_i column c_i, if they correspond to data subjects.

$$\boxed{\mathbb{E} : e_1.r_1, ..., e_n.r_n} \qquad \boxed{\mathbb{E}_{DS} : t_i.c_i}$$

Consent: If the entity represents a data subject o_i, it may consent to a set of purposes P_i regarding the processing of the DPOs it owns. The concept of data subject consent does not exist in the web application implemented by the developer. It is managed by RuleKeeper's **consent service**, and does not require mapping to the application.

$$\boxed{\mathbb{C} : o_1.P_1, ..., o_n.P_n}$$

Principals: Principals represent the users u_i that interact with the system, which can have roles r_i associated and can act on behalf of some entity e_i. Each role r_i is only allowed to execute a set of operations O_i in the system.

$$\boxed{\mathbb{U} : u_1.r_1.e_1, ..., u_n.r_n.e_n} \qquad \boxed{\mathbb{R} : r_1.O_1, ..., r_n.O_n}$$

Similar to the E_i entity policy, the association principal-entity can either be declared statically in the policy, in the case of data controllers, processors and third parties, or can be mapped in the database. In the last case, we define a policy that makes an association between the column p_i that identifies principal and the column e_i that identifies the entity, in a table t_i.

$$\boxed{\mathbb{U}_{DS} : t_i.e_i.p_i}$$

Data Ownership: Each PDO must be associated with its owner. As PDOs correspond to data stored in the database tables, we associate each table t_i that contains personal data with the column o_i of that table that identifies the owner of such data.

$$\boxed{\mathbb{N} : t_1.o_1, ..., t_n.o_n}$$

The normal activity of the system will be the invocation of operations that will interact with the data store. Each operation o will be allowed to be executed by the principal p if the following conditions are satisfied:

1. exists r in *roles*(p) such that $r \in$ *granted-acess*(o)

2. for all t in *typeset*(o), exists p_t in *purposes*(t), and exists p_o in *purposes*(o), such that $p_o \in p_t$

3. for all t in *typeset*(o), exists p_t in *purposes*(t), such that $t \in$ *maximum-data*(p_t)

4. exists p in *purposes*(o) and *requires-consent*(p) then, for all d in *dataset*(o), *granted-consent*(owner(d), d, p)

The condition (1) validates the access control. The condition (2) refers to purpose limitation and the condition (3) to data minimization requirements. Condition (4) refers to to lawfulness base requirements, where an operation that acts upon the PDOs for a given purpose and that purpose requires consent of the data subjects, it can only be executed if the PDO owners have granted its consent.
The nomenclature used in these conditions is described as follows:

- *granted-access(o)*: roles authorized to perform operation o, specified by the principals' policy $\mathbb{U}$ and $\mathbb{R}$.
- *typeset(o)*: types of data processed by operation o, specified by the operations' policy $\mathbb{O}$.
- *purposes(o)*: purposes to which the operation o is executed, specified by the operations' policy $\mathbb{O}$.
- *maximum-data(p)*: the maximum data the purpose p is allowed to access, specified by the purposes' policy $\mathbb{P}$.
- *requires-consent(p)*: checks if the lawfulness base associated with the purpose p requires the data subject consent, specified by the purposes' policy $\mathbb{P}$.
- *dataset(o)*: PDOs processed by the operation o, specified by the operations' policy $\mathbb{O}$ and the data type policy $\mathbb{D}$.
- *granted-consent(o, d, p)*: checks if the owner o consented to the processing of its PDO d for purpose p, specified by the ownership policy $\mathbb{N}$, entities' policy $\mathbb{E}$ and consents' policy $\mathbb{C}$.
- *owner(d)*: data subject that owns the PDO d, specified by the ownership policy $\mathbb{N}$.

4.5.2 Extensions

To comply with the remaining principles related to other features beyond data access control, RuleKeeper also provides a specification for them. To extend RuleKeeper with such features, the system would need to supervise a set of global conditions and act upon them. The complete support for these extensions is out of scope of this work.

Storage conditions: It could be established a set of conditions that limit the storage of personal data, according to the *Storage Limitation* principle. The completion of a purpose would result from an event e_i and triggers the action a_i. This event e_i can be an application-specific action a_i^s or the expiration of a TTL t_i associated with a stamp s_i.

$$\boxed{\mathbb{S} : e_1.a_1, ..., e_n.a_n} \qquad e_i = a_i^s \vee t_i.s_i$$

These policies specify what triggers the completion of a purpose and the action that is supposed to be executed when the event is triggered. The action corresponds to a predefined routine, that could be implemented by the developer. This routine could either erase the data or send a notification to the DPO, for dedicated processing.

Accuracy assertions: A set of assertions s_i that verify the accuracy of the system could be defined, according to the *Accuracy Preservation* principle. For instance, using accuracy assertions, GPSL would allow to specify that a citizen card is inaccurate if its expiration date has passed. This is defined by comparing a data d_i with a value v_i, using a comparative method m_i: ***eq***, ***neq***, ***lt***, ***le***, ***gt*** or ***ge***. If the data is found to be inaccurate, action a_i should be triggered to fix the inaccuracy. If the fix is not performed as expected, the controller has a time t_i for fixing it.

$$\boxed{\mathbb{A} : s_1.a_1.t_1, ..., s_n.a_n.t_n} \qquad s_i = d_1.m_1.v_1$$

RuleKeeper would need to ensure that accuracy conditions are evaluated at the prescribed triggers. If one of the assertions fails, then a reactive procedure must ensue in order to restore the accuracy in the PDO store. Under normal conditions, the inaccuracy should be fixed by the defined action, with the support of the application. If it is not rectified within the predefined time, the DPO should be flagged.

Data subject rights: The exercise of data subject rights depend on the organization policy. $\mathbb{W}$ policies state which rights could be exercised automatically, without the data controller's involvement and which ones would go through the data controller for approval.

$$\boxed{\mathbb{W} : r_1.m_1.D_1, ..., r_m.m_n.D_m}$$

Each right r_i has an access mode m_i to a set of data D_i, that expresses whether its exercise requires the DPO mediation or not. RuleKeeper could provide a system that would allow the data subject's right exercise. If a right required the data controller approval, an event would be generated for the DPO to interact. If not, RuleKeeper would a predefined routine, implemented by the developer.

4.6 Policy Lifecycle and Enforcement

As presented in Figure 4.1, RuleKeeper Manager is responsible for storing and managing the GPSL template, which is managed by the organization's operator, and also for storing the RuleKeeper Data, which is the required data to evaluate the GPSL policies. The RuleKeeper Manager merges this information, generating a GPSL manifest, that is used by RuleKeeper's middleware to enforce such policies.

RuleKeeper middleware is composed of hooks that interact with RuleKeeper's PEP to convert the application request to a GPSL authorization request, evaluated by RuleKeeper' PDP.

Table 4.3 summarizes the policy enforcement roles each RuleKeeper component plays.

Component	Description	Policy Evaluation Role
Context Handler Hook	Handles the application requests' context.	PEP
Data Ownership Hook	Describes the involved data subjects.	PEP
Policy Enforcement Hook	Evaluates the GPSL data protection policies.	PEP & PDP
RuleKeeper Manager	Manages the GPSL policies and data.	PAP & PIP

Table 4.3: RuleKeeper policy enforcement roles.

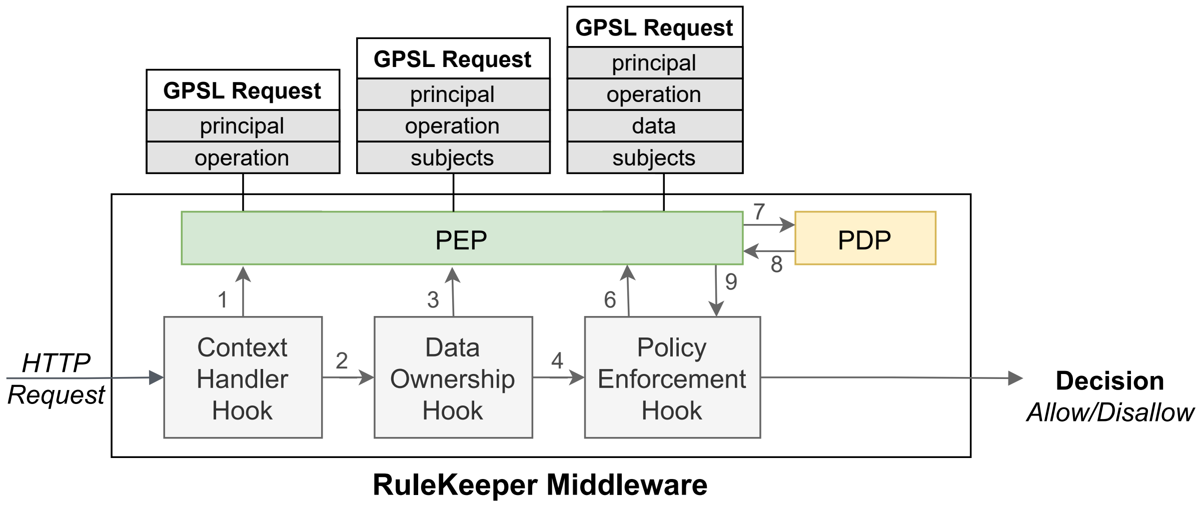


Figure 4.4: RuleKeeper's policy enforcement flow.

4.6.1 Policy Enforcement

RuleKeeper's policy enforcement mechanism comprises a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP), configured with the GPSL policies. The PEP is responsible for generating a GPSL authorization request and triggering the generation of an authorization decision in the PDP.

Figure 4.4 describes the GPSL policy enforcement flow through arrows, representing the sequential interactions with both PEP and PDP. First, the HTTP request is intercepted by the **Context Handler** hook, which sets the request context. When the controller queries the database, that request is intercepted by the **Data Ownership** hook and the **Policy Enforcement** hook. This last procedure is executed as many times as queries to the database. The Policy Enforcement hook generates a decision that reflects if the request is allowed or not.

Context Handler: The data protection policies' evaluation depends on the entity performing the operation and on the operation being executed, as different operations may have different purposes or different access levels. So, PEP generates a **GPSL Authorization Request** pre-filled with the context of the request: the principal executing the request and the operation being executed.

Data Ownership: To evaluate the GPSL policies, is required to know to which data subject the processed data belongs to. On top of that, requests to the web application may involve personal data belonging to several data subjects at the same time. PEP adds information to the GPSL Request regarding the data subjects involved in the request.

Policy Enforcement Hook: The policy enforcement hook receives the information generated by the previous hooks, and as it intercepts the database query, it turns this request into a complete GPSL authorization request by adding the requested data. It then uses the complete GPSL request to query the PDP, which returns the authorization decision generated by the GPSL data protection policies evaluation. In case of a positive authorization decision, the request is forwarded to the application controller, otherwise, the request is denied.

4.6.2 Policy Lifecycle

As presented in Figure 4.1, RuleKeeper manages the data protection policies and stores the required data for enforcing such policies in RuleKeeper Tables. It is composed of a GPSL template, managed by the organization's DPO to depict the organization's privacy policy and of tables that store the required data to enforce such policy. RuleKeeper Manager then merges the GPSL template, filled by the DPO with the RuleKeeper's data into a GPSL manifest.

Policy Configuration: When RuleKeeper is integrated and initialized in a web application, it establishes a connection with the RuleKeeper Manager. RuleKeeper Manager then configures the middleware with the GPSL manifest, which allows the middleware to evaluate the policies locally, without the involvement of the Manager.

Policy Management: The privacy policy management is the operator's responsibility. As so, Rule-Keeper Manager provides an interface that allows the operator to view and update the GPSL template. It lets the operator to map the organization's privacy policy with its requirements, such as purposes, storage conditions, amongst others, including the application-context mapping.

Policy Data: RuleKeeper must store the dynamic data involved in the policy evaluation in RuleKeeper Manager tables, that contain organized and relevant data from the application logic. This data comprises the information on the data subject's consent and is managed by RuleKeeper's Manager **consent service**. This service provides an api that is available for the data subjects to submit their consent.

Policy And Data Updates: Both the GPSL template and the policy data can be updated by the operator and the data subject, respectively. Whenever the RuleKeeper Manager receives either update, it generates a new GPSL manifest with the new information and sends it to every RuleKeeper middleware connected.

Summary

This chapter detailed the design of RuleKeeper, a novel web framework that allows the development of GDPR-compliant web applications. The design of RuleKeeper includes GPSL, a declarative policy specification language that allows to specify, in a non-ambiguous way, the organizations' privacy policies. GPSL is based on the Purposeful Data Objects abstraction, responsible for modeling the web application's state, according to the organization's privacy policy, covering the required information for achieving GDPR-compliance. It supports the GDPR principles and challenges described in Chapter 3. The next chapter describes the implementation of the RuleKeeper's prototype.

Chapter 5

Implementation

This chapter addresses the implementation details of the RuleKeeper prototype. Section 5.1 presents an overview of RuleKeeper integration in web applications. The policy specification and enforcement is presented in Section 5.2 and Section 5.3 describes how policies are managed by RuleKeeper.

5.1 Implementation Overview

We have implemented a RuleKeeper prototype for MERN web applications, as represented in Figure 5.1. The RuleKeeper prototype comprises a middleware based on Node.js and Express, to be integrated in MERN web applications, and a management server, implemented as a MERN web server as well. We used Mongoose to help us model the MongoDB application data. These components communicate through web sockets, implemented with Socket.IO. We implemented RuleKeeper middleware as an external package, to incorporate it as seamlessly as possible, requiring minimal effort from the developer.

5.1.1 MERN Web Framework

In web development, most developers opt to build their web applications using frameworks whether for time-saving, scalability, integration, robustness, and many other advantages. Although in the early days, developers specialized in a specific area, such as frontend or backend, due to market trends, a lot of them are becoming *full-stack* developers, meaning that they work in both frontend and backend at the same time. Full-stack developers employ a combination of different technologies and frameworks concerning the frontend, backend, and database layers to build a complete web application.

From an historical point of view, full-stack frameworks started to become popular, starting with the *LAMP stack* and culminating with frameworks such as *MEAN/MERN stack* and *Ruby on Rails*. For the 7th consecutive year, JavaScript is considered the most commonly used programming language and as a result of its popularity, frameworks such as React.js, Angular, Express.js, Node.js, and others are experiencing exponential growth [49, 50]. Due to its popularity and advantages in terms of performance, maintainability and ease to use, we chose to implement a RuleKeeper prototype for MERN applications.

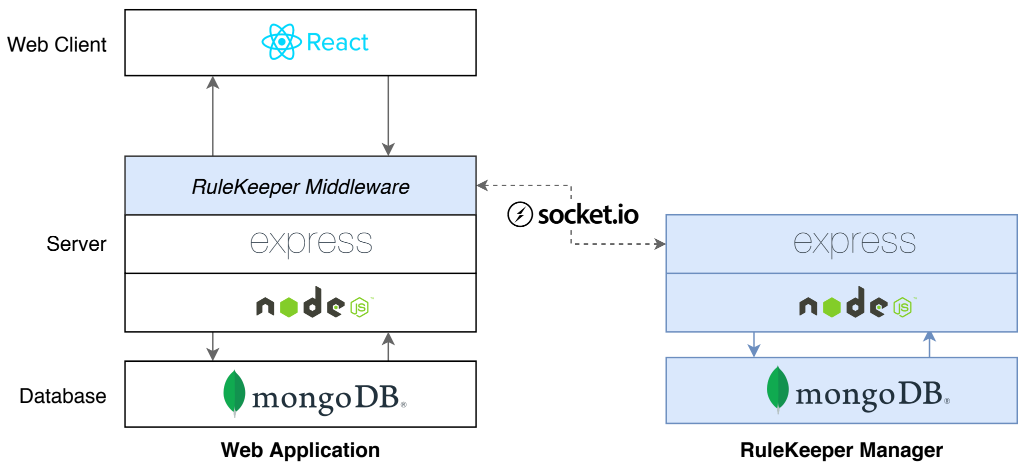


Figure 5.1: Components of RuleKeeper prototype (blue boxes).

5.1.2 System Setup and Operation

To initialize the system, RuleKeeper must be launched at the web application web server. The web server must import the RuleKeeper middleware libraries and initialize it in the Express application (typically, the well-known *app.js*). The RuleKeeper Manager must be configured with the RuleKeeper data tables, mapped accordingly. Once RuleKeeper Manager is launched, a communication channel is established between the middleware and the management server. The RuleKeeper Manager server is global to the organization, and only needs to be initialized once. The middleware only needs to be configured once as well, but can have as many running instances as web servers.

As presented in Figure 4.1, the middleware is responsible for enforcing the local policies and the manager is responsible for managing such policies. The policy enforcement and policy management mechanisms are detailed in Sections 5.2 and Section 5.3, respectively. For an efficient and accurate policy evaluation, the RuleKeeper Manager continuously updates the connected middleware instances with new policies and/or data.

5.1.3 RuleKeeper Internal Communication

The internal communication of RuleKeeper, between the middleware and the manager, is performed using **Socket.IO**. Socket.IO is a Node.js library that enables real-time, bidirectional and event-based communication. When possible, it tries to establish a WebSocket connection, and if that is not possible, falls back on HTTP long polling. It implements a heartbeat mechanism that allows both the management server and middleware to know when the other one is nor responding anymore. The RuleKeeper Manager is responsible for opening this channel, attached to the existing **HTTPS** server. This automatically upgrades the underlying communication to SSL.

The purpose of this continuous communication is to allow the Manager to configure new booted instances of the middleware and to continuously update the existing ones.

```
    app.use(function (req, res, next) {
        // middleware code (hooks)
        next();
    });
```

Listing 4: Intercepting HTTP Requests with Express middleware.

```
    schema.pre(m_function, function(next) {
        // middleware code (hooks)
        next();
    });
```

Listing 5: Intercepting MongoDB Requests with Mongoose middleware.

5.1.4 Intercepting Requests

RuleKeeper's policy enforcement is based on the interception of both the HTTP request and the subsequent database queries. To intercept the HTTP Request, we used the Express built-in application-level middleware, which has access to the request object **req**, the response object **res**, and the **next** middleware function in the application's request-response cycle, as presented in Listing 4. It is executed everytime the application receives a request, and we can add as much middleware as we need.

Intercepting the database queries uses the same approach, but employing Mongoose middleware. This middleware intercepts the defined mongoose functions from a defined schema, as presented in Listing 5. To avoid having to define hooks for all mongoose schemas, requiring the developer to configure RuleKeeper middleware with the application databases, we used mongoose global plugins to register these hooks for every schema. We implemented pre mongoose hooks for intercepting read queries and post mongoose hooks for intercepting create/update/delete queries. We also implemented a global hook responsible for setting the context of the query, which intercepts all queries, including document, query and model functions. To allow mongoose middleware to access the express middleware context, we used the npm package *request-context*.

5.2 Policy Specification and Enforcement

The RuleKeeper framework must allow the specification of data protection policies in accordance with the GDPR and guarantee the enforcement of such policies. It is required that the policy enforcement is detached from the web application, easily maintained by the organization's team of developers, attaining the minimum interference with the application logic. It should also not significantly slow down the web application performance. The GPSL policies should also be easy to specify and maintain by non-experts in computer engineering. To achieve these goals, we specified our GPSL policies using the Rego language and used Open Policy Agent as our policy engine.

5.2.1 Policy Specification

We specified the GPSL policies using the Rego language, described in Section 2.3.3. Rego policies are very simple, maintainable and built for a world where JSON documents are pervasive. As GPSL policies are declarative and based on if-conditions, we were able to specify them entirely using Rego.

GPSL policies are evaluated every time the web application queries the database and, in a nutshell, verify the conditions presented in Section 4.5, corresponding each condition to a Rego rule. Listing 6

```
# Allow if complies with the data minimization principle
data_is_minimal {

  # Check purposes of the invoked operation
  purposes := get_operation_purposes(input.operation)

  # Get input data that is personal
  personal_data := {x | x = input.data[_]} & {x | x = data.personaldata[_]}

  # Get maximum data of all purposes of the operation (intersection)
  min_data := {y.data | y = data.minimum_data[_]; y.purpose == purposes[i]}

  # Check if input data is a subset of minimal data
  allowed_purposes := {d | d = min_data[_];
                         count({i | i = d[_]; i == personal_data[_]}) == count(personal_data)}

  count(allowed_purposes) == count(purposes)
}
```

Listing 6: Rego implementation of the GPSL data minimization condition.

presents an example of the Rego implementation of the GPSL data minimization condition.

5.2.2 Policy Engine

We implemented the policy engine using Open Policy Agent, described in Section 2.4.2, an open source and lightweight policy engine that enforces Rego policies. We integrated the OPA Policy Engine with the WebAssembly module to evaluate the policies. According to the analysis of different integration choices conducted by OPA itself [51], the integration via WebAssembly provides the fastest evaluation. WebAssembly [52] (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. It is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

The development of the WebAssembly integration with OPA is still in progress but the current work was more than enough to develop this prototype, since they already provide the JavaScript SDK. The RuleKeeper Manager compiles the Rego policies into Wasm instructions and sends them to the middleware. These instructions are embedded and can be evaluated by a WebAssembly runtime, with different inputs and external data, using the *@open-policy-agent/opa-wasm* package.

5.2.3 Policy Evaluation

As described in Section 4.6.1, the policy enforcement comprises tree different hooks. First, the HTTP request is intercepted by the Context Handler hook. When the controller queries the database, that request is intercepted by the Data Ownership hook and the Policy Enforcement hook. This last procedure is executed as many times as queries to the database. This section details the implementation of the presented hooks, ordered by the execution sequence.

Context Handler

The Context Handler hook is the first to be executed and is responsible for generating the GPSL Authorization Request pre-filled with the context of the request: the principal executing the request and the operation being executed. Web applications have no default method of identifying the principal of the request, excluding volatile information such as the IP address, and many have their own implementation of authentication systems. So, RuleKeeper implements its own authentication using *json web tokens* which is used to identify the principal in the context of the web application.

Regarding the type of operation, since the web application's concept of operation is the URL being requested, RuleKeeper identifies the request's operation as the requested URL. We used the *express-list-endpoints* npm package to help us match the correct express route. As presented in Section 5.1.4, the middleware has access to the express **request object**, which is global to all application's request-response cycle. So, we use this object to store the GPSL Authorization Request, in JSON. This hook was implemented with 36 lines of Express middleware code.

Data Ownership

The Data Ownership hook is the first to be executed when the application controller queries the database. As the personal data involved in the requests it is not marked with the corresponding data subject by default, this hook is responsible for associating the data with its owner(s). To achieve that, all tables in the database that contains personal data must have a column with a key that identifies the data subject that owns such data, and that information must be provided to RuleKeeper, so it can access the correct data to identify the data subject. This requirement is already established in the developer's API, described in Section 4.4. This hook does not evaluate the organization's policy, it only pre-validates and pre-processes the data for a correct evaluation, according to the following rules:

- If the entity is a **data subject**, the processing is only allowed if the queried personal data belongs to it. This hook verifies if the principal owns the data and if it does not, the request is terminated.
- If the entity is a **controller**, **processor** or **third party**, it is required to verify the consent of all the data subjects involved in the request. RuleKeeper adds a list of those data subjects to the GPSL Authorization request.
- If the entity does not have associated a GDPR role, RuleKeeper terminates the request.

This hook was implemented using 123 lines of Mongoose middleware code.

Policy Enforcement

The last hook to be executed is the Policy Enforcement hook, responsible for evaluating the GPSL data protection policies. This hook intercepts the query before it is executed in the database in case of *WRITE* queries, to prevent the query from being executed without permission. *READ* queries are intercepted after the query is executed, to be able to get with precision the data that is trying to be accessed. First, the data involved in the request is added to the the GPSL Authorization Request. Then, it queries the

Data Subject	Consented Purposes
alice	[clinical analysis, marketing]

Table 5.1: RuleKeeper Manager consent table.

WebAssembly runtime module with the GPSL Authorization Request as input. If the decision is *false*, it uses the **res** object to terminate the HTTP request. This hook was implemented using 110 lines of Mongoose middleware code.

5.3 Policy Management

As presented in Section 4.6.2, RuleKeeper Manager manages the data protection policies and stores the required data for enforcing such policies in RuleKeeper Tables. Although the RuleKeeper Manager is an independent server and, as a matter of fact, could be built using an array of different technologies, we used the same tools as MERN. This provided consistency with the middleware and allowed to implement the web socket communication as seamlessly as possible. RuleKeeper Manager was implemented with 624 lines of JavaScript code.

The relevant data stored by RuleKeeper Manager for enforcing the GPSL policies consists of a single table that stores the data subject's consent, represented in Figure 5.1. At the same time, it stores the GPSL Rego policies and, by using the OPA client tools for WebAssembly (*@open-policy-agent/opa-wasm* package) it generates the *.wasm files containing the wasm instructions that correspond to the Rego policies. It also stores a copy of the data in a JSON format, to allow the configuration and update of the middleware instances.

When a RuleKeeper middleware instance connects to the Manager, it sends the middleware a copy of the wasm policies and the json data. To support the update of a Rego policy in runtime, RuleKeeper provides an operation, that receives the new *.rego file, re-compiles it into the wasm instructions and replaces the old *.wasm file. Whenever there's an update to either the data or the policies, RuleKeeper Manager bundles that data in a JSON file and sends it to all connected RuleKeeper middleware instances, through a Socket.IO event.

Summary

This chapter described the implementation of the three components of RuleKeeper: the policy specification language, the policy enforcement mechanism and the policy management. This prototype comprises a middleware based on Node.js and Express, to be integrated in MERN web applications and a management server, implemented as MERN server. We used Rego and Open Policy Agent to specify and evaluate the GPSL policies. For the management of policies, we implemented a management server, responsible for managing the GPSL policies and storing the required data in dedicated tables. The next chapter presents the experimental evaluation of this prototype.

Chapter 6

Evaluation

This chapter presents the experiments that were performed to evaluate RuleKeeper. The main goal of our evaluation is to demonstrate that RuleKeeper is able to successfully enforce GDPR-policy compliance through the enforcement of GPSL policies, while providing good application maintainability and delivering good performance results. Section 6.1 describes the case study that supported a real-world validation of our system. Then, Section 6.2 details the measurements of the performance of RuleKeeper. Finally, Section 6.3 presents the level of effort required for a developer to use RuleKeeper.

6.1 Case Study

Not only the regulation is extensive and considerably poor on specifics, but its implementation also depends on the organizations' domain. Therefore, to evaluate the relevance and usability of RuleKeeper, we applied it to a concrete case study, allowing us to perform a real-world validation of our system. We collaborated with **LEB** - *Laboratórios Elisabete Barreto*[1], a clinical laboratory with the aim of developing a prototype intranet service for supporting its internal administrative processes. Since GDPR imposes many restrictions in systems that process personal data, the implementation of a prototype that supports the processing of health-related data (a less permissive personal data category) is very pertinent to the evaluation of GDPR-compliance.

6.1.1 GDPR Support for health-related organizations

Whenever an organization seeks to be compliant with the regulation, the data controller must learn, interpret, and apply the regulation principles to their specific case. To simplify that process, some associations study and develop frameworks that determine the best approach for certain types of organizations. Thus, organizations can follow a certified framework, developed and optimized by a specialized association, without having to implement it from scratch, making the process less error-prone.

APAC, a Portuguese association for clinical analysts who collaborates with national and international health institutions, developed a turnkey GDPR framework that highlights a collection of principles and

[1] https://www.leb-analises.com

Purpose	Lawfulness Base	PIA	Processes
Clinical Analysis Management	Performance of a contract	Yes	Pre-analytic process, Analytic process, Post-analytic process
Human Resources Management	Consent	Yes	Application management, Employees management, Self-employed workers management
Marketing	Performance of a contract	Yes	Marketing communications
Video Surveillance	Legitimate interests of the company	Yes	Video surveillance
Administrative Management	Legitimate interests of the company	No	

Table 6.1: Personal data processing activities defined by APAC.

establishes a set of technical and organizational measures that meets the requirements we introduced in Chapter 3. It also provides a set of guidelines regarding the enforcement of the regulation, adopted by LEB. The presented guidelines state that it is mandatory to preserve records of processing activities and privacy impact assessments.

Records of Processing Activities: Each data controller must preserve all records of processing activities that manipulate personal data under its responsibility. This record consists in a compilation of all processing performed on personal data, grouped by purpose. The information on each purpose of processing includes the lawfulness base, the categories of personal data involved, the data retention period, and whether or not a privacy impact assessment was performed.

Privacy Impact Assessments: The data controller must conduct a privacy impact assessment on all processing activities regarding personal data protection before initiating its processing, since they can implicate a high risk for individuals' rights and freedoms. Contemplating most laboratories, APAC determined four purposes subject to a privacy impact assessment, and associated each one with their functional description.

Table 6.1 presents the information regarding the purposes involved in all processing activities carried out by the laboratories, their lawfulness base, if they were selected for a privacy impact assessment and if so, the processes involved in each purpose, defined by APAC.

6.1.2 Internal Process Analysis

To design and develop a web application that supports the internal administrative processes of the LEB organization, we performed a detailed and extensive analysis of the impact assessment reports regarding personal data processing for clinical analysis laboratories (PIA's), developed by APAC, and the LEB's internal administrative processes, conducted by LEB. We selected the four LEB processes that process personal data and therefore need to be compliant with the regulations: (1) the pre-analytic process, (2) the analytic process, (3) the post-analytic process and (4) the human resources management process.

The **pre-analytic** process is responsible for the patient registration and the specimen preparation and processing, involving the operations *verification of medical requisition*, *analysis form filling*, *estimate of price and time of the results*, *result delivery receipt print*, *analysis form print*, *specimen collection*

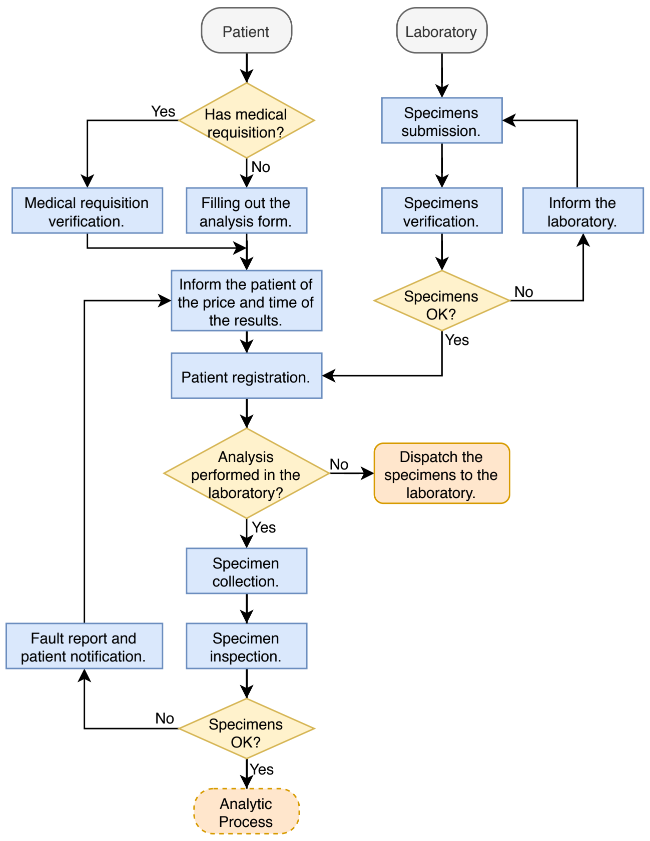

Patient
Laboratory
Has medical requisition?
Yes
No
Medical requisition verification.
Filling out the analysis form.
Specimens submission.
Specimens verification.
Inform the laboratory.
Inform the patient of the price and time of the results.
Specimens OK?
No
Yes
Patient registration.
Analysis performed in the laboratory?
No
Dispatch the specimens to the laboratory.
Yes
Specimen collection.
Specimen inspection.
Fault report and patient notification.
Specimens OK?
No
Yes
Analytic Process

Figure 6.1: Flowchart representing the LEB pre-analytic process.

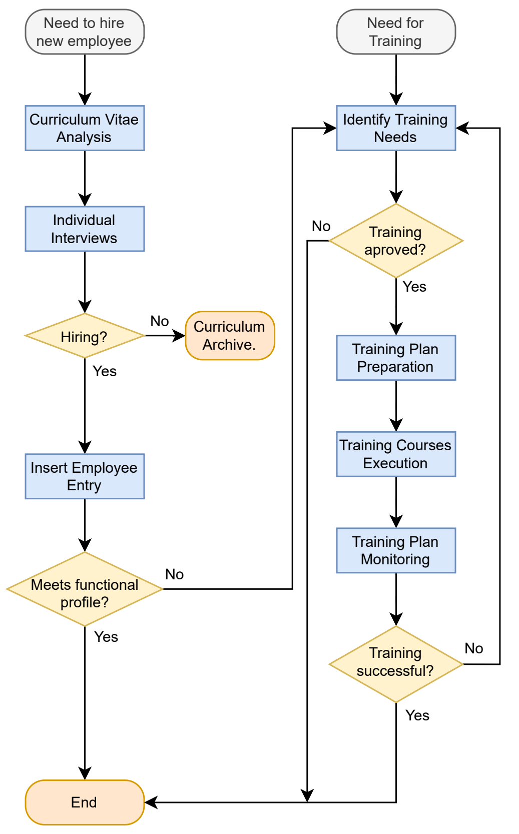


Figure 6.2: Flowchart representing the LEB human resources management process.

instructions print, *patient registration in the system*, *submission of specimen by laboratory*, *specimen collection*, *specimen verification* and *patient or laboratory notification of faulty specimen*. It involves the actors *patient*, *receptionist*, *clinical analyst*, and *external laboratory*. The **analytic** process is responsible for the specimen analysis and validation, involving the operations *specimen testing*, *specimen results validation*, *analysis results entry* and *notify the need of a new collection*. It involves the actors *patient*, *clinical analyst* and *technical manager*. The **post-analytic** process is responsible for preparing and emitting the analysis results, involving the operations *analysis result report emission*, *external results registration*, *contact external laboratory*, *biopathological validation*, *result delivery to the patient* and *notify the need of a new collection*. It involves the actors *patient*, *technical manager*, *clinical analyst* and *external laboratory*. The **human resources management** process is responsible for hiring and training new employees, involving the operations *curriculum vitae analysis*, *interviewing candidate*, *insert employee in the system*, *curriculum vitae archive*, *identification of training needs*, *design of a training plan*, *execution of a training plan*, *monitoring of a training plan* and *evaluation of training effectiveness*. It involves the actors *management*, *quality department*, *employee* and *candidate*. As all of these processes process personal data, APAC assigned a purpose to each one of them. For the clinical processes, the purpose is *Clinical Analysis*, and for the human resources management process, the purpose is *Human Resources*. These processes individually detailed and simplified in Figures 6.1, 6.2, 6.3 and 6.4. Decision blocks are represented in yellow, process blocks are represented in blue and terminal blocks are represented in orange.

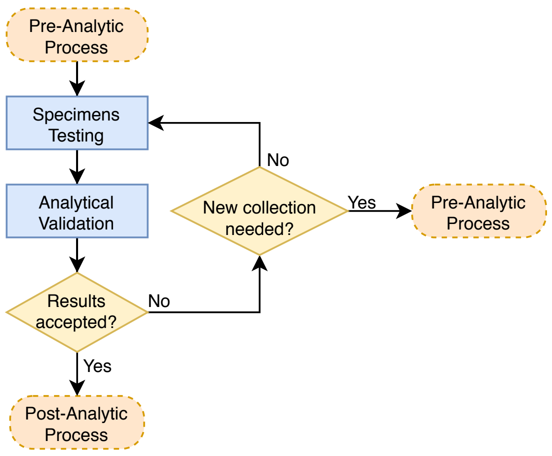


Figure 6.3: Flowchart representing the LEB analytic process.

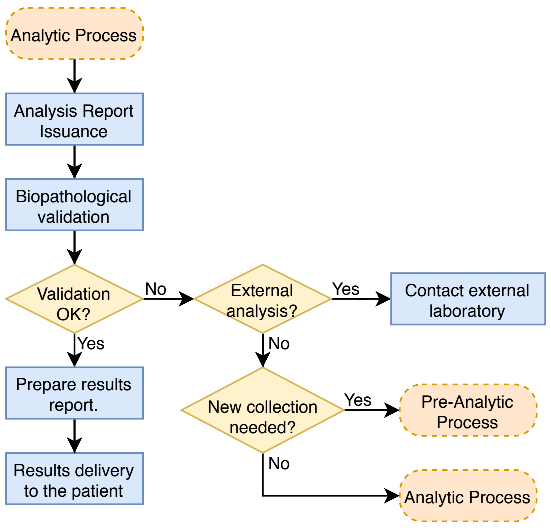


Figure 6.4: Flowchart representing the LEB post-analytic process.

6.1.3 Prototype Implementation

Taking into consideration our analysis of the LEB internal administrative processes and their implications regarding personal data, signaled by the privacy impact assessments, we implemented a prototype intranet service, based on the MERN stack. Our prototype supports three types of users: patients, receptionists, and system administrators. It simulates the actions between such users and the service, supporting the mentioned processes, specifically the following operations: (i) patient registration by receptionists, (ii) patient data handling by both receptionists and patients and (iii) user management by system administrators. We specified the LEB privacy policies, using the Rego implementation of GPSL.

Web Application: Our prototype was built using the MERN stack with an in-memory database. Given the extent of the internal processes and the fact that most of them process the same data, our prototype is simplified and only supports a set of core operations. As our goal is to extend this prototype with Rule-Keeper, we focused on the processes and actions that involve the patients' personal data processing. We implemented eight controllers as described in Table 6.2.

Policy Specification: To inform the data subjects on the processing made regarding their personal data, LEB composed a privacy policy and a document named *Information on the Processing of Personal Data*. Both documents disclose how LEB processes personal data and how it applies data protection principles, meeting **Arts. 12** (*Transparent Information*), **13**, and **14** (*Information to be provided*) of the **GDPR** [3]. To demonstrate that GPSL fully supports the policies detailed by LEB's privacy policy, we matched the LEB's privacy policy requirements with the GPSL policies, as depicted in Table 6.3.

6.1.4 Portability Effort

The integration of RuleKeeper in our clinical analysis laboratory prototype consisted in integrating the RuleKeeper code in the LEB prototype web application and describing the organization's privacy policy in GPSL. To integrate RuleKeeper in the LEB web application, it was required to import and initialize

Controller	Description	Data	Lines of Code
Register Patient	The receptionist can register new patients in the system.	Patient data (*citizencard, patient_id, authenticated, full_name, birth_date, gender, ss_number, photo, address, mobile, e_mail)*, User data: (*username, password, role, entity*)	50
Get Patient Information	The receptionist can fetch a patient's data. A patient can fetch its own data.	Patient data (*citizencard, patient_id, authenticated, full_name, birth_date, gender, ss_number, photo, address, mobile, e_mail)*	14
Get Information from all Patients	The receptionist can fetch data from all patients in order to query some parameters.	Patient data (*citizencard, patient_id, authenticated, full_name, birth_date, gender, ss_number, photo, address, mobile, e_mail)*	11
Update Patient Information	The receptionist can update a patient's data. A patient can update some aspects of its own data.	Patient data (*citizencard, patient_id, authenticated, full_name, birth_date, gender, ss_number, photo, address, mobile, e_mail)*	12
Register User	A system administrator can register a new user in the system.	User data: (*username, password, role, entity*)	37
Delete User	A system administrator can delete a user in the system.	User data: (*username, password, role, entity*)	8
Update User	A system administrator can update a user in the system.	User data: (*username, password, role, entity*)	10

Table 6.2: Controllers (operations) implemented by the LEB prototype intranet service.

RuleKeeper middleware. This involved adding 3 lines of code to the LEB web application. Then, RuleKeeper Manager requires a setup, involving setting up the RuleKeeper tables and mapping them to the data model. Last and more challenging of all, we tried to describe LEB's privacy policy in the GPSL's Rego implementation. This task came out very complex and tricky since the LEB privacy policy was lacking important GPSL requirements.

6.2 Performance Evaluation

In this section, we aim to assess RuleKeeper's quality and performance impact in existing web applications. We evaluate RuleKeeper's performance as part of the prototype implementation of our clinical analysis laboratory use case web application.

6.2.1 Methodology and Metrics

This performance evaluation aims at measuring the performance overhead of RuleKeeper in terms of latency and throughput. To measure RuleKeeper's latency, we measured the total execution time of the system without RuleKeeper and the total execution time of the system with RuleKeeper. The throughput measurement was performed by saturating the system with and without importing RuleKeeper and calculating the maximum sustainable rate for each case. To test requests that may result in different outcomes, we performed these tests using 3 different controllers. For that, we describe 3 request types, characterized by parameters that may influence RuleKeeper's performance and associate each one of them with a controller implemented in the use case application, described in Table 6.4. Each controller is characterized by the entity performing the operation and the number of data subjects involved.

LEB Privacy Requirement	GPSL Policies
“Personal data will be exclusively processed for the clinical analysis activity exercise, by Elisabeth Barreto Laboratories”	Entities $\mathbb{E}$ policy: ‘Laboratory LEB’.‘data controller’
“The purposes of the data collected are: Management of information regarding clinical analysis services, Human resource Management, Marketing (if applicable) and Video surveillance, (if applicable)”, “*The legal basis for the processing of your personal data is the contract established at the time you ask us to perform clinical analysis.*”	Purposes $\mathbb{P}$ policy: ‘clinical analysis’.‘execution of a contract’, ‘human resources management’, ‘marketing’, ‘video surveillance’
On the first visit to the laboratory, the patient is asked for the citizen card to open the identification form and the following data can be collected: full name, date of birth, sex, beneficiary number, TIN, photo, address, mobile phone/telephone, email and relevant clinical observations.	Data Type $\mathbb{D}$ policy : (full name, date of birth, sex, beneficiary number, TIN, photo, address, mobile phone, clinical information) Consent $\mathbb{C}$ policy : ‘patient A’.‘clinical analysis’
The collected data is processed and stored computerized (...) during the minimum period necessary for use according to the purpose for which they were collected	This data is insufficient to specify a storage limitation $\mathbb{S}$ policy.
The data collected and held by LEB — Elisabeth Barreto Laboratories may be transmitted (...) to the following entities: Health Insurance and Subsystems; Health professionals; Subcontractors who will process the data on behalf of LEB — Elisabeth Barreto Laboratories and according to the purposes determined by it.	Entities $\mathbb{E}$ policy: ‘health insurance A’.‘third party’, ‘subcontractor A’.‘data processor’.
To request the exercise of any data subject right, must be sent an e-mail to *dpo@leb-analises.com*	Owner Rights $\mathbb{W}$ policy: ‘right to access’.‘indirect access mode’

Table 6.3: LEB privacy requirements matched with GPSL policies.

#	Controller	Path	Entity	Data Subjects
1	Get Patient Information	*GET /patients/:patientId*	Data Subject	1
2	Get Patient Information	*GET /patients/:patientId*	Controller	1
3	Get Information from all Patients	*GET /patients/all*	Controller	100

Table 6.4: Controllers used for performance measurements.

Our testbed is composed of five 64-bit Ubuntu 18.04.5 LTS virtual machines (VMs) provisioned with 20GB of RAM and eight virtual Intel Xeon E5506 2.13GHz CPUs, located in different physical machines but connected over a local network. VM1 runs the use case application, extended or not with Rule-Keeper, depending on the test, and VM4 runs an in-memory MongoDB database with the use case application data. VM2 executes an instance of the RuleKeeper Manager prototype and VM3 runs an in-memory MongoDB database with the RuleKeeper Manager data. Finally, VM5 is used to run the client-side experiments with up to 50 client instances, running simultaneously to saturate RuleKeeper.

6.2.2 Execution Time Overheads

We simulated traffic by generating 10.000 sample requests sequentially for each one of the 3 controllers and measuring the time required to return each response. For each controller, we measured the total

Controller	Controller Time (ms)		RuleKeeper Time (ms)		Overhead
	mean	99th	mean	99th	
#1	9.16	16.31	10.16	16.73	1.00 ms (9.84%)
#2	9.37	19.60	10.68	30.82	1.31 ms (12.27%)
#3	62.73	145.14	69.47	159.82	6.47ms (9.70%)

Table 6.5: RuleKeeper overheads, sampled from the generated traffic.

execution time of the controller without importing RuleKeeper and the total execution time of the controller with RuleKeeper, reporting the arithmetic mean and 99th percentile. The execution time does not include web page loading and rendering. Table 6.5 and Figure 6.5 show the execution time overheads. The yellow bars represent the controller execution time without RuleKeeper and blue bars represent the controller execution time with RuleKeeper. On average, RuleKeeper overheads are low and likely unnoticeable to web application users: 8-12%. As expected, these are small client-perceived overheads.

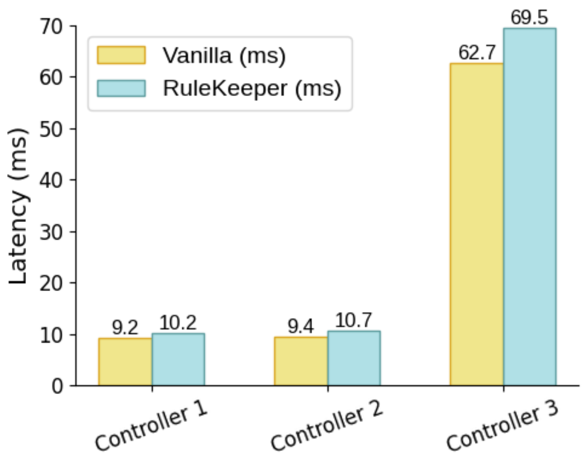


Figure 6.5: RuleKeeper overhead for 3 controllers in the use case application. Labels give the execution times.

6.2.3 Operation Scalability

To evaluate RuleKeeper throughput under high load, we used wrk2 to generate sample requests of the 3 controllers and measured the maximum sustainable rate, i.e the maximum number of requests our server could respond to, before it could no longer answer incoming requests within a reasonable amount of time. We report the average operations/second, as observed in Figure 6.6, where listed areas represent the web application using RuleKeeper. In average, RuleKeeper responds to 20-30% fewer requests, regardless of the number of data subjects involved in the operation, but with worse throughput when the entity performing the action is not the data subject. Regarding the context of RuleKeeper applications, the provided throughput is still very acceptable to web application requests.

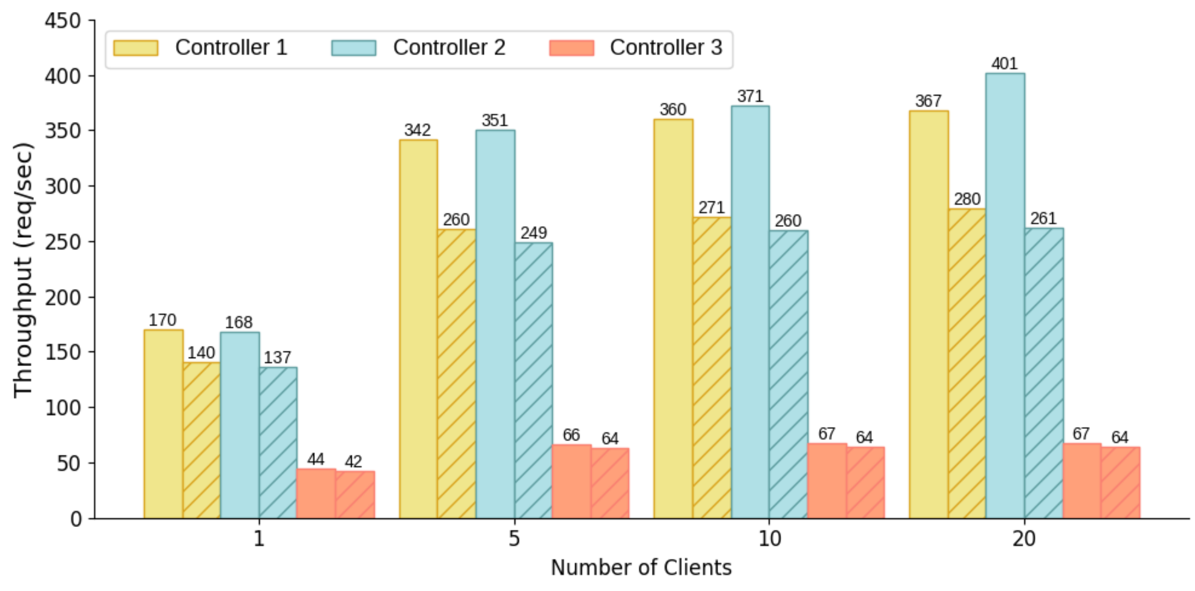


Figure 6.6: Throughput measured using the pre-defined LEB use case controllers.

6.3 Programming Effort

RuleKeeper prototype implementation is composed by different components: (i) the *RuleKeeper Middleware* to be imported by the 3-tier web applications, and (ii) the *RuleKeeper Manager* server. To determine the level of effort required for a developer to use RuleKeeper, we measured the lines of code needed to import RuleKeeper middleware into the web application, the requirements to setup RuleKeeper Manager and the additional logic to update the RuleKeeper Manager data.

Lines of Code: To use the RuleKeeper middleware in a Node/Express 3-tier application, the developer only needs to add 3 lines of code. As the RuleKeeper middleware was developed as an npm package, to import it in a node/express application, the developer only needs to add one line. Then, it needs to add two lines: one line responsible for adding mongoose middleware - placed before importing the mongoose models - and another line for adding express route middleware - placed before defining the routes and after custom middleware.

RuleKeeper Manager Requirements: To use RuleKeeper, it is mandatory to setup and boot the RuleKeeper Manager and the RuleKeeper Manager database. To setup and boot the RuleKeeper Manager server, the developer needs to create the MongoDB database with an empty collection: *consent*. Then, the developer needs to import the code and define the mongodb database url.

Additional Data: The data model specified by the developer must have, for each table that contains personal data, a column that identifies the data owner to support the data ownership mechanism.

Summary

This chapter described the implemented case study and the experimental evaluation of RuleKeeper. In collaboration with **LEB**-*Laboratórios Elisabete Barreto*, we developed a prototype intranet service for supporting its internal administrative processes, with special attention to their implications regarding personal data.

This prototype allowed us to evaluate the relevance and usability of RuleKeeper by performing a real-world validation of our system. To assess RuleKeeper's quality and performance impact on existing web applications, we measured the performance overhead of RuleKeeper in terms of latency and throughput, using our use case web application. RuleKeeper has shown to only add small client-perceived overheads, a very acceptable result regarding the web application context .

The next chapter concludes this document by summarizing the achievements of this thesis and introducing future work.

Chapter 7

Conclusions and Future Work

Nowadays, people tend to share their data publicly and at an unprecedented scale to use many of the services provided by web applications. This free and uncontrolled flow of data has increased the scale of collection and sharing of personal data. To protect citizens, the European Union issued the General Data Protection Regulation (GDPR), which comprises a collection of rules and guidelines that aim to ensure the deployment of extensive security mechanisms for the protection of users' data and privacy.

However, building information systems as off now has been at odds with the enforcement of such GDPR-compliant mechanisms. Most modern systems tend to be optimized for performance, cost, and reliability, leaving security as a secondary goal. Additionally, due to its numerous advantages, most developers are opting to build their applications using web frameworks, which do not provide any compliance with the GDPR. This challenges developers to built their GDPR-aware web applications without disrupting such frameworks. As a result, not only the web users remain prone to numerous risks, including the exposure of sensitive data, but the organizations themselves may incur high fees in the case of non- compliance with the GDPR.

In this thesis, we provided a detailed GDPR analysis to fully understand the challenges of GDPR compliance in information systems and presented a set of principles that organizations must follow when managing their information systems. Considering this set of principles and challenges, we designed and implemented RuleKeeper, a novel web application framework tailored to provide data security and privacy protections according to GDPR-compliant policies. RuleKeeper includes a declarative policy specification language, GPSL, that allows to specify, in a non-ambiguous way, the organization's privacy policies, based on a novel abstraction named *Purposeful Data Objects*. The PDO abstraction allows the application state modeling, covering the required information to achieve GDPR-compliance and supporting our GDPR analysis results.

We collaborated with **LEB** - *Laboratórios Elisabete Barreto* to perform a real-world validation of our system, including the expressiveness of GPSL policies. The experimental evaluation conducted over RuleKeeper shows that the integration of RuleKeeper in existing web applications only adds small client-perceived overheads, while providing good application maintainability.

In the near future, we want to fully implement the RuleKeeper prototype, supporting all the designed mechanisms, including a system to exercise the data subjects' rights and a monitoring system. We also believe that the current implementation could be optimized, enhancing RuleKeeper's scalability and that the evaluation could be extended by performing micro-benchmark measurements and expressing a range of privacy policies using GPSL. It would be interesting to develop a user interface for the organization's privacy policies, which could be used by Data Protection Officers to test the GPSL policy expressiveness. Other possibilities for future work include the incorporation of automatic systems in RuleKeeper, such as transactional support for consent withdrawal, comprising the backtracking and reversion of all effects contemplating the data owned by a given data subject, and accuracy consistency support, including the specification of such conditions and their automatic validation. Lastly, an important direction for future work comprises a deeper study on how to enforce the presented GDPR requirements in legacy systems.

Bibliography

[1] The Member States. Charter of Fundamental Rights of the European Union. *Official Journal of the European Union*, C 326, October 2012.

[2] The Member States. Consolidated version of the Treaty on the Functioning of the European Union. *Official Journal of the European Union*, C 326, October 2012.

[3] The European Parliament and the Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L 119, May 2016.

[4] OWASP. Top 10 - 2017: A3 sensitive data exposure. https://www.owasp.org/index.php/Top_10-2017_A3-Sensitive_Data_Exposure Accessed: 2020-01-04.

[5] J. Mohan, M. Wasserman, and V. Chidambaram. Analyzing GDPR compliance through the lens of privacy policy. *CoRR*, abs/1906.12038, 2019.

[6] T. Bertram, E. Bursztein, S. Caro, H. Chao, R. C. Feman, P. Fleischer, A. Gustafsson, J. Hemerly, C. Hibbert, L. Invernizzi, L. K. Donnelly, J. Ketover, J. Laefer, P. Nicholas, Y. Niu, H. Obhi, D. Price, A. Strait, K. Thomas, and A. Verney. Five years of the right to be forgotten. In *Proceedings of the Conference on Computer and Communications Security*, 2019.

[7] C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz. (Un) informed consent: Studying GDPR consent notices in the field. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 973–990, 2019.

[8] A. Shah, V. Banakar, S. Shastri, M. Wasserman, and V. Chidambaram. Analyzing the impact of gdpr on storage systems. In *Proceedings of the 11th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'19. USENIX Association, 2019.

[9] S. Shastri, M. Wasserman, and V. Chidambaram. The seven sins of personal-data processing systems under gdpr. In *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'19. USENIX Association, 2019.

[10] M. Schwarzkopf, E. Kohler, M. F. Kaashoek, and R. Morris. Position: Gdpr compliance by construction. In *Poly/DMAH@VLDB*, 2019.

[11] OASIS. eXtensible Access Control Markup Language (XACML) version 3.0, 2013. `http://docs.oasisopen.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf` Accessed: 2020-01-04.

[12] I. Akyildiz and X. Wang. A survey on wireless mesh networks. *Communications Magazine, IEEE*, 43(9):S23–S30, Sept. 2005. ISSN 0163-6804. doi: 10.1109/MCOM.2005.1509968.

[13] OASIS. Abbreviated language for authorization version 1.0, 2015. `https://www.oasis-open.org/committees/download.php/55228/alfa-for-xacml-v1.0-wd01.doc` Accessed: 2020-01-04.

[14] OASIS. Open standards. open source. `https://www.oasis-open.org/` Accessed: 2020-01-04.

[15] C. Ardagna, L. Bussard, S. De, C. Vimercati, G. Neven, S. Paraboschi, E. Pedrini, F.-S. Preiss, D. Raggett, P. Samarati, S. Trabelsi, and M. Verdicchio. Primelife policy language. January 2009.

[16] M. Azraoui, K. Elkhiyaoui, M. Önen, K. Bernsmed, A. S. De Oliveira, and J. Sendor. A-PPL: an accountability policy language. In *Data privacy management, autonomous spontaneous security, and security assurance*, pages 319–326. 2014.

[17] D. Butin, M. Chicote, and D. Le Métayer. Log design for accountability. In *IEEE Security and Privacy Workshops*, pages 1–7, May 2013.

[18] S. Kasem-Madani and M. Meier. Security and privacy policy languages: A survey, categorization and gap identification. *CoRR*, abs/1512.00201, 2015.

[19] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal), 2003. Version submitted to the W3C. Available at `http://www.w3.org/Submission/2003/SUBMEPAL-20031110`.

[20] A. H. Anderson. A comparison of two privacy policy languages: Epal and xacml. In *Proceedings of the 3rd ACM Workshop on Secure Web Services*, SWS '06, page 53–60. Association for Computing Machinery, 2006.

[21] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[22] M. Abadi and B. T. Loo. Towards a declarative language and system for secure networking. In *Proceedings of the 3rd USENIX International Workshop on Networking Meets Databases*, NETB'07. USENIX Association, 2007.

[23] A. Vahldiek-Oberwagner, E. Elnikety, A. Mehta, D. Garg, P. Druschel, R. Rodrigues, J. Gehrke, and A. Post. Guardat: Enforcing data policies at the storage layer. In *Proceedings of the European Conference on Computer Systems*, EuroSys '150, pages 13:1–13:16. Association for Computing Machinery, 2015.

[24] E. Elnikety, A. Mehta, A. Vahldiek-Oberwagner, D. Garg, and P. Druschel. Thoth: Comprehensive policy compliance in data retrieval systems. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 637–654. USENIX Association, 2016.

[25] M. Becker, C. Fournet, and A. Gordon. Security policy assertion language (secpal) specification, version 1.0, February 2007.

[26] M. Becker, A. Malkis, and L. Bussard. A framework for privacy preferences and data-handling policies. December 2019.

[27] C. N. C. Foundation. Rego Policy Language, 2019. `https://www.openpolicyagent.org/docs/latest/policy-language` Accessed: 2020-12-07.

[28] S. Chong, K. Vikram, and A. C. Myers. Sif: Enforcing confidentiality and integrity in web applications. In *Proceedings of USENIX Security Symposium on USENIX Security Symposium*, pages 1–16, 2007.

[29] K. Pullicino. Jif: Language-based information-flow security in java. *CoRR*, abs/1412.8639, 2014.

[30] J. Liu, M. D. George, K. Vikram, X. Qi, L. Waye, and A. C. Myers. Fabric: A platform for secure distributed computation and storage. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, pages 321–334, 2009. doi: 10.1145/1629575.1629606.

[31] A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek. Improving application security with data flow assertions. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, pages 291–304, 2009.

[32] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. C. Mitchell, and A. Russo. Hails: Protecting data privacy in untrusted web applications. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, pages 47–60, 2012.

[33] F. Wang, R. Ko, and J. Mickens. Riverbed: Enforcing user-defined privacy constraints in distributed web services. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, pages 615–629, 2019.

[34] A. Mehta, E. Elnikety, K. Harvey, D. Garg, and P. Druschel. Qapla: Policy compliance for database-backed systems. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1463–1479, 2017.

[35] A. Bichhawat, M. Fredrikson, J. Yang, and A. Trehan. Contextual and granular policy enforcement in database-backed applications. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, page 432–444, 2020.

[36] C. N. C. Foundation. Open Policy Agent (OPA), 2019. `https://www.openpolicyagent.org/` Accessed: 2020-12-07.

[37] J. Valacich and C. Schneider. *Information Systems Today: Managing the Digital World*, chapter 1. Prentice Hall Press, 4th edition, 2009.

[38] K. C. Laudon and C. G. Traver. *Management Information Systems*. Prentice Hall Press, 12th edition, 2011.

[39] A. Leff and J. T. Rayfield. Web-application development using the Model/View/Controller design pattern. In *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, pages 118–127, Sep. 2001. doi: 10.1109/EDOC.2001.950428.

[40] MongoDB. The database for modern applications. `https://www.mongodb.com` Accessed: 2020-01-04.

[41] Express.js. Fast, unopinionated, minimalist web framework for Node.js. `https://expressjs.com` Accessed: 2020-01-04.

[42] Angular.js. Superheroic javascript MVW framework. `https://angularjs.org` Accessed: 2020-01-04.

[43] Node.js. Javascript runtime built on Chrome's V8 javascript engine. `https://nodejs.org/` Accessed: 2020-01-04.

[44] React.js. A javascript library for building user interfaces. `https://reactjs.org` Accessed: 2020-01-04.

[45] E. D. P. Board. Administrative criminal proceedings of the austrian data protection authority against Österreichische post ag. `https://edpb.europa.eu/news/national-news/2019/administrative-criminal-proceedings-austrian-data-protection-authority_en` Accessed: 2020-12-13.

[46] C. C. N. de l'Informatique et des Libertés. The cnil's restricted committee imposes a financial penalty of 50 million euros against google llc. `https://www.cnil.fr/en/cnils-restricted-committee-imposes-financial-penalty-50-million-euros-against-google-llc` Accessed: 2020-12-13.

[47] G. Cloud. Data deletion on google cloud platform. `https://cloud.google.com/security/deletion` Accessed: 2020-12-13.

[48] T. N. Y. Times. Uber hid 2016 breach, paying hackers to delete stolen data. `https://www.nytimes.com/2017/11/21/technology/uber-hack.html` Accessed: 2020-12-13.

[49] Stack Overflow. Stack overflow developer survey results 2019. `https://insights.stackoverflow.com/survey/2019` Accessed: 2020-01-04.

[50] GitHub. Octoverse git hub top languages. `https://octoverse.github.com/#top-languages` Accessed: 2020-01-04.

[51] O. P. Agent. Integrating opa. https://www.openpolicyagent.org/docs/latest/integration/ Accessed: 2020-12-19.

[52] Webassembly. https://webassembly.org/ Accessed: 2020-12-19.