POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A framework to manage access control policies in federated Data Mesh

Tesi di Laurea Magistrale in
Computer Science and Engineering
Ingegneria Informatica

Author: **Diego Caronni**

Student ID: 928230
Advisor: Prof. Pierluigi Plebani
Co-advisor: Dott. Mattia Salnitri
Academic Year: 2022-23

Abstract

In recent years, data have become increasingly heterogeneous so, the old monolithic and centralized data platforms, such as the data warehouses and the data lakes, showed their limitations. The new paradigm of the Data Mesh data architecture, introduced by Zhamak Dehghani in 2019, tries to overcome these limitations. The core concept of the Data Mesh is considering the data as a product. Each domain team, inside a Data Mesh, is responsible for the data provided and manages it in order to make it available to other domain teams. Being a recent topic, the literature offers few works about it and some aspects must be better clarified. In our work, we want to investigate how can be possible to define and manage security policies to access the data in the federated context of the Data Mesh. We will provide models to design a security framework, with the related components to access the data and tools to run the policies. The models proposed, regulate the interactions between data consumers and data products. We will adopt Open Policy Agent (OPA) as the policy decision point (PDP) to test the policies that we will define.

Keywords: Data Mesh, Access Control, Security Policies, OPA, Rego

Abstract in lingua italiana

Negli ultimi anni, i dati sono diventati sempre più eterogenei e di conseguenza le vecchie piattaforme di contenimento dei dati, monolitiche e centralizzate, come le data warehouses e i data lakes, hanno mostrato i loro limiti. Il nuovo paradigma dell'architettura dati del Data Mesh, introdotto da Zhamak Dehghani nel 2019, prova a superare questi limiti. Il concetto centrale del Data Mesh è considerare il dato come un prodotto. Ogni team di dominio, all'interno di un Data Mesh, è responsabile dei dati forniti e li rende disponibili ad altri team di dominio. Trattandosi di un argomento recente, lo stato dell'arte offre pochi articoli a riguardo e alcuni aspetti dovrebbero essere chiariti meglio. In questo lavoro di tesi, vogliamo indagare come sia possibile definire e gestire delle politiche di sicurezza per accedere ai dati, nel contesto federato del Data Mesh. Forniremo modelli per definire i componenti coinvolti nell'accesso dei dati e gli strumenti per eseguire le politiche. I modelli proposti regolano la relazione tra consumatori e dati. Adotteremo Open Policy Agent (OPA) come decisore di politiche (PDP) per testare le politiche che definiremo.

Parole chiave: Data Mesh, Controllare l'Accesso, Politiche di Sicurezza, OPA, Rego

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

1 Introduction **1**

2 State of the art **3**
2.1 Access control 3
2.2 Security patterns 5
2.3 Declarative languages for policies 11

3 Baseline **15**
3.1 XACML architecture 15
3.2 Open Policy Agent 16
3.3 Datalog 18
3.4 Rego 19

4 Proposed approach **21**
4.1 Data Mesh components and design patterns 21
4.2 The three levels to access the data 22
4.2.1 Location of PDP and PEP 23
4.3 Scenarios 24
4.3.1 Simple data product 24
4.3.2 Aggregation of data products 26
4.4 Definition of the policies 28

5 Examples and tests **33**
5.1 Policies at the business level 33

5.2 Policies at the technical level 33
5.3 Tests 37
5.3.1 Example 1 37
5.3.2 Example 2 38
5.3.3 Example 3 38
5.3.4 Example 4 39

6 Conclusions **41**

Bibliography **43**

List of Figures **47**

List of Tables **49**

Listings **51**

Acknowledgements **53**

1 | Introduction

The term Data Mesh was first introduced by Zhamak Dehghani in 2019, this term refers to a new paradigm for data platform architectures, that tries to overcome the limitations of the old generations of data platforms. Among them, there is a first generation of proprietary enterprise data warehouses and business intelligence platforms. Then arrived the second generation of the big data ecosystems with the data lakes. Finally there is a third generation of architectures that are similar to the previous ones and concentrate on the streaming for real-time data availability with architectures such as Kappa, unifying the batch and stream processing for data transformation with frameworks such as Apache Beam, as well as exploiting cloud based managed services for storage, data pipeline execution engines and machine learning platforms.

Despite some improvements during the decades, these architectures still suffer of the following problems, as Dehghani explained in [14] and [15]:

- They are *centralized and monolithic* and they ingest data from all corners of the enterprise; this could be a problem for enterprises with rich domains, large number of sources and different set of consumers because, as more as data become ubiquitously available, the ability to harmonize it in one place under the control of one platform diminishes.
- They have a *coupled pipeline decomposition*, that lead to the creation of independent teams to parallelize the work and reach higher operational scalability and velocity. Though this model provides some level of scale, by assigning teams to different stages of the pipeline, it has a limitation that slows the delivery of features: it has high coupling between the stages of the pipeline to deliver an independent feature or value.
- They have *siloed and hyper-specialized ownership*, the data platform engineers are separated and grouped into a team based on their technical expertise of big data tooling, often absent of business and domain knowledge.

In the last years, the Data Mesh led to new principles, as stated by Dehghani in [16] and

[17]:

- The *domain ownership principle*: there are domain teams that are responsible for their data. The analytical and operational data ownership is moved to the domain teams, away from the central data team.
- The *data as a product principle*: a domain team is responsible for satisfying the needs of other domains by providing high-quality data. Data is distributed across many domains.
- The *self-serve data infrastructure platform principle*: a dedicated data platform team provides domain-agnostic functionality, tools, and systems to build, execute, and maintain the interoperability of data products for all domains.
- The *federated governance principle*: there is an interoperability of all data products through standardization, thanks to a data ecosystem with adherence to the organizational rules and industry regulations.

Among famous societies that started to introduce the Data Mesh architecture there are Zalando and Netflix as reported by [27].

We can state, from the literature, that this new paradigm for data platform architectures seems to have a lot of advantages, and tries to solve part of the problems of the previous generations. Being a recent topic, there are not a lot of works about it, and sometimes are confusing because have shortcomings and unclear aspects. One of the aspects, of great interest, that we want to investigate and explain better, concerns how to securely access the data products in the federated context of the Data Mesh. In particular we want to explore how to manage and define, in a sufficient expressive way, policies that control the access to the data products.

In this work, we want to propose an approach to access data in the Data Mesh, and to do this, we will provide the correspondent architecture and models, to clarify all the entities and components, involved in the access of the data, with their relationships. We will adopt the ABAC paradigm to perform access control because it let us to define a great variety of rules for policies, that suits our goals. Finally we will provide and test some examples of policies, written in Rego, using OPA as the PDP.

2 | State of the art

In this chapter, we investigate among works that could be useful to better understand and know the arrival point of the researches dealing with the problems presented in chapter 1. We focus on works concerning access control, the development of frameworks for security policies, both, in general, in the business sector and specifically, in the Data Mesh architecture and other publications about declarative languages for policies.

2.1. Access control

NIST, in a document published in 2006 [22], discusses the capabilities, limitations, and qualities of the access control mechanisms that are embedded for access control policy. From this paper we can learn some basic notions about access control systems; the concepts of: subject, object, operation, permission are explained. They are introduced, among the others, three main kind of access control policies with their limitations.

- Discretionary access control (DAC) leaves a certain amount of access control to the object's owner or anyone else who is authorized to control the object's access. DAC policy tends to be very flexible and is widely used in the commercial and government sectors. However, DAC is known to be weak for two reasons. First, granting read access is transitive and the user who is granted can in an easily way grants any other user access to the copy of the file of the owner. Second, DAC policy is vulnerable to Trojan horse attacks.
- Mandatory access control (MAC) means that access control policy decisions are made by a central authority, not by the individual owner of an object, and the owner cannot change access rights. It is very limited and often used in the military field.
- Role-based access control (RBAC) means that access decisions are based on the roles that individual users have as part of an organization. The use of roles to control access can be an effective means for developing and enforcing enterprise-specific security policies. The challenge of RBAC is the contention between strong security

and easier administration. For stronger security, it is better for each role to be more granular, but this, is contrast with an easier administration.

At this point we can state, that RBAC is not enough to define policy in the context of the Data Mesh architecture that is based on the concept of the data as a product; basing access only to the roles of the subjects is not sufficient; there is the necessity of a mechanism that consider also the metadata associated to the resources, and maybe other possible attributes that the subject can have, e.g. the name of the company where he/she works, so we need to use another paradigm such as the attribute-based access control (ABAC).

In [23], NIST provides a paper which focuses on attribute-based access control (ABAC). It is stated that ABAC is a logical access control methodology where authorization performs a set of operations that is determined by evaluating attributes associated with the subject, object, requested operations, and, in some cases, environment conditions against policy. ABAC controls access to objects by evaluating rules against the attributes and allows an unrestricted number of attributes to be combined to satisfy a rich set of policies. ABAC is well suited for large enterprises. An ABAC system can implement existing role-based access control policies and can support a migration from role-based to access control policies based on many different characteristics of the individual requester. However, the final considerations state that ABAC system is more complicated, and therefore more costly to implement and maintain, than other access control systems. This paper is only a discussion about ABAC, without providing examples that apply specifically in the Data Mesh architecture. We deduce by the information provided, that ABAC can be suitable to reach our goals.

To better understand the differences of the RBAC and ABAC, we examined the paper [6]. Different aspects are compared and we take in consideration only the more relevant. In terms of global agreement, RBAC does not have a global agreement, ABAC has a global agreement because of shareable user database and domain to domain connection. In terms of flexibility, RBAC is not flexible, while ABAC is very flexible because of dynamic nature in the distributed and open system. Concerning easiness, RBAC is a user-friendly access control model and rights are assigned to roles statically on the basis of static and predefined policies, ABAC is more complicated because of its global agreement, flexibility, sharing and heterogeneity of user attributes. About dynamicity, it is not supported in RBAC because in RBAC roles and permissions are given to user's domain statically; ABAC supports dynamicity because access is granted on the basis of attributes and decision is taken runtime. Concerning authorization decision, the assignment of permissions to roles are given in local domain in RBAC; in ABAC, authorization decision is performed globally as per user's credentials are given dynamically. Concerning manageability, ABAC is more difficult to manage rather than RBAC but in ABAC with respect to RBAC is simpler to change the privileges because it is not necessary to change the identity of the user or the policy terms. Finally in ABAC there is not the problem of role explosion.

It is possible to combine the two paradigms [1], making an hybrid approach, combining RBAC and ABAC can provide some of the advantages of both models. RBAC, being aligned so closely to business logic, is simpler to implement than ABAC while to provide an additional layer of granularity when making authorization decisions, ABAC is more suitable. This hybrid approach determines access by combining a user's role (and its assigned permissions) with additional attributes to make access decisions. In doing this, there are also two disadvantages, this hybrid approach does not reach the same granularity as ABAC and it does not have the same simplicity as ABAC, due to those disadvantages we decide to consider only the ABAC approach in developing our policies to have more granularity. Another problem is that there are no guarantees that a policy decision point supports policies formulated with the hybrid approach.

2.2. Security patterns

Our focus now is to investigate about security patterns proposed by other researchers, we start by security patterns for the business sector in general and then we move to security patterns in the Data Mesh.

Concerning security, in the context of business environments, this article [34] provides an high level policy enforcement pattern that is supposed to satisfy the following requirements: the flexibility of scripted policy rules, the need for adaptability which is addressed by using a policy repository and the creation of plug-in modules of policy elements. It is stated that for the deployment of a security pattern, is necessary to address to the following aspects: integrating pre-built policy libraries, provide a flexible configuration, the client should be able to implement and customise business rules easily, policies should be stored in a repository and managed by a central instance. In Figure 2.1, the aforementioned security pattern.

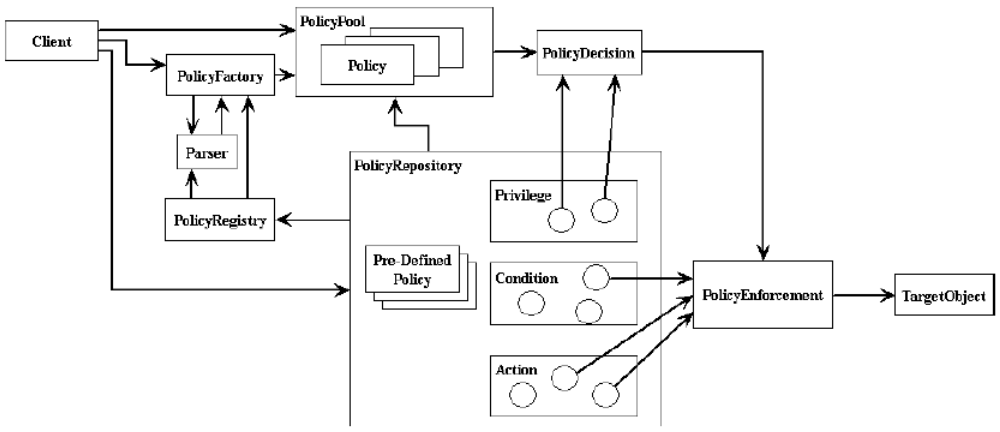


Figure 2.1: Policy enforcement pattern [34].

- The Client may request an execution on a targeted object.
- The Policy Factory creates runtime policy instances and puts them into the Policy Pool.
- The Policy Pool provides efficient runtime storage for the policy instances. If a policy that a client requests is already in the pool, it can be used directly without instantiating it again.
- The Parser parses the plain text format of policies, and then queries the Policy Registry.
- The Policy Registry is a lightweight storage for specifications of policy element interface.
- Pre-Defined Policies are rules that can be applied to the clients' request.
- Policy Elements are implemented modules used in policies, including Privilege, Condition and Action.
- The Policy Decision decides whether or not a certain request is allowed.
- The Policy Enforcement evaluates the required policy conditions and invokes the corresponding policy actions.
- The Policy Repository provides storage for the pre-defined policies, and policy elements.
- The Target Object is the object that the client requests.

The benefits of this pattern are: flexibility, customizability, adaptability and universality; it can be applied to a wide range of environments and new functions can be added. There are also some drawbacks: performance, allowing the plain text as a policy description, needs to be parsed and interpreted at runtime. This will slow down the policy execution. Other drawbacks are the higher complexity and difficulties to understand the pattern.

In this paper [30] concerning Web services platforms, is discussed a way to overcome the lack of products that support non-functional features such as security, transactionality and reliability, in a service-oriented environment. WS-Policy is a possible future standard way to specify these features and associate them with services; this paper presents a working prototype that demonstrates how the quality features can be expressed as service policies using WS-Policy. These policies are enforced by a policy framework that allows dynamic association of such non-functional features with applications on a per-interaction basis, as well as modification of these features. The focus is on two main issues: representing non-functional service behavior (granularity of policy attachment, multiple sources of policy documents, combining policies to create effective policy, negotiating runtime policy) and issues in designing the policy framework. A policy document represents constraints over the service to which such a document is attached. The aim is that at runtime the service must never violate the behavior specified via the policy assertions. It is formulated both a policy model and a policy framework design, which have a main module called Policy4J which is in charge to parse and store the policy document. In this way, every policy document is transformed in a tree form representing the correspondent boolean expression. Then, the Policy combination module is invoked during the service deployment and the service invocation to combine the supported policy and the requested policy obtaining the effective policy. Once the effective policy is computed the Policy enforcement module is in charge to process all the incoming and outgoing messages in order to verify if all of them are compliant to what the effective policy expresses. At the end of the paper the problems that emerge are: the matchmaking between services is complex since it is difficult to reach an agreement when the vocabularies used in describing policies are different, the interpretation of extraneous or missing information in the policies is unclear. These considerations will be useful for our goal, they let us understand the possible difficulties in developing a security framework.

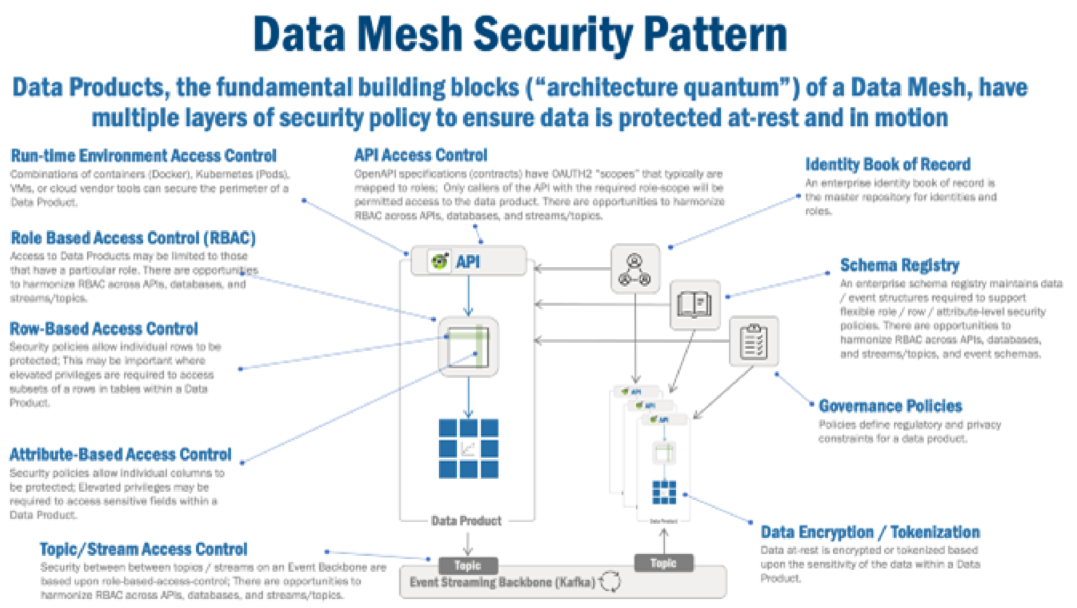
Changing side and coming in detail of works about the Data Mesh, this article, by IBM [9], states that the concept of Data Mesh lacks empirical insights. Specifically, an understanding of the motivational factors for introducing Data Mesh, the best practices, its business impact, and potential archetypes. To address this gap, the members of IBM conducted 15 interviews with industry experts and the results show that organizations have difficulties with the transition toward federated governance associated with the data mesh concept, the shift of responsibility for the development, provision, and maintenance of data products, and the comprehension of the overall concept. Among those challenges, what is relevant for our case is the one about federated data governance; according to the interviewees, the federated approach creates a number of problems for activities and responsibilities that were previously centrally owned and enforced. While they stress the importance of federated data governance to establish rules according to domain needs, interviewees highlight limitations regarding the automated execution; especially concerning security, regulatory, and privacy-related topics, the employees within the domain are unaware of which data are protected and regulated. The only solution given is that to address this challenge, is argued that organizations companies should introduce a cross-domain steering unit responsible for the enforcement of specific governance rules, especially concerning security, regulatory, and privacy-related topics. But does not provide an example or a detailed method to achieve security.

In a blog by Starburst [29] is discussed in general about the Data Mesh architecture; the relevant part is the final part where there is a comparison between pre-Mesh and the Data Mesh architecture, concerning security. What happened is a shift from centralized Data policies, to the possibility for domain teams to create policies for their products; from a centralized privacy to a federate privacy discussed and agreed between the domains; from RBAC to ABAC. In particular it is stated that RBAC fails in Data Mesh because, especially in a large organizations, for example, the domain owner of the marketing domain won't know all of the roles of employees in the operations domain. Using ABAC instead means that the domain defining the policies do not need to know about roles, groups and users in other domains, but rely on those other domains to maintain the correct mapping from roles, groups and users to the correct tags. In ABAC the role is one of the many possible attributes. This blog does not come into the details of policy formulation for Data Mesh, offers only a general view and useful tips.

The concepts stated in the previous blog [29], are better clarified by some members [21] of the universities of Tilburg (Netherlands) and Salerno (Italy), that, as stated by them, due to the scarcity of academic researches, wanted to better clarify about Data Mesh design principles, architectural components, capabilities, and organizational roles. They analyzed, and synthesized 114 industrial gray literature articles. What emerged from the researches is that the metadata of a data product may describe its owners, schemas, quality metrics, and access policies. The product can also include code to enforce various data governance policies. A data product needs to use appropriate measures to ensure data confidentiality, integrity, and availability. The measures include but are not limited to data access control, data encryption at rest and data in transit, and data access audit logs. In the Data Mesh architecture, while the security policies are primarily managed centrally, their enforcement happens in a decentralized fashion. The federated governance defines rules or guidelines that enforce consistent and correct collection, storage, access, usage, and management of data assets in an organization. For example, the governance policies at the Data Mesh level can enforce a minimum level of data security or enforce access control based on central identity management. For the Policy Enforcement, the platform should offer tools that allow local and global federated governance teams to define, store, attach, observe, and enforce governance policies. The gray literature recommends using the policy-as-code approach, where policies are defined, evaluated, and managed programmatically. Moreover, the platform also needs to provide services to support policy implementation, e.g., data anonymization service and data quality metrics calculation service. Those are useful guidelines but in practice, do not go into deep details about the languages to formulate those policies.

The next paper analysed, [26] aims to propose a domain model and a conceptual architecture towards the achievement of decentralized data architectures. It is proposed a domain model for the Data Mesh, in this model the security aspect is managed by a security mechanism divided into two parts, Authentication and Authorization. This security mechanism acts on all nodes in the Data Mesh and all nodes make use of the self-serve Data Platform, where they make use of storage, processing, data integration system, data visualization, software development, and machine learning tools. Data Mesh can be implemented in just one cluster, or in several, depending on existing resources and needs. To implement this are used, Apache Ranger as security mechanism, Apache Atlas for data catalog components, Apache HDFS and HIVE for node creation and storage, Apache Spark for processing, Apache Ignite for machine learning models, and Tableau for data visualization. Also including Jenkins as a testing component, Confluence for documentation, Docker as a container registry. No references about declarative languages to use in the design of the security policies.

Then, we found another interesting article [10], where the author describes a privacy policy framework for Data Mesh, that can represent and reason about complex privacy policies. It is presented a policy decision engine that supports two main uses: interfacing with user interfaces for the creation, validation, and management of privacy policies; and interfacing with systems that manage data requests and replies by coordinating privacy policy engine decisions and access to encrypted databases using various privacy enhancing technologies. Here in the figure below the aforementioned security pattern.


Data Mesh Security Pattern
Data Products, the fundamental building blocks ("architecture quantum") of a Data Mesh, have multiple layers of security policy to ensure data is protected at-rest and in motion
Run-time Environment Access Control
API Access Control
Identity Book of Record
Role Based Access Control (RBAC)
Schema Registry
Row-Based Access Control
Governance Policies
Attribute-Based Access Control
API
Data Product
Topic
Event Streaming Backbone (Kafka)
Topic
Topic/Stream Access Control
Data Encryption / Tokenization

Combinations of containers (Docker), Kubernetes (Pods), VMs, or cloud vendor tools can secure the perimeter of a Data Product.

OpenAPI specifications (contracts) have OAUTH2 "scopes" that typically are mapped to roles; Only callers of the API with the required role-scope will be permitted access to the data product. There are opportunities to harmonize RBAC across APIs, databases, and streams/topics.

An enterprise identity book of record is the master repository for identities and roles.

Access to Data Products may be limited to those that have a particular role. There are opportunities to harmonize RBAC across APIs, databases, and streams/topics.

An enterprise schema registry maintains data / event structures required to support flexible role / row / attribute-level security policies. There are opportunities to harmonize RBAC across APIs, databases, and streams/topics, and event schemas.

Security policies allow individual rows to be protected; This may be important where elevated privileges are required to access subsets of a rows in tables within a Data Product.

Policies define regulatory and privacy constraints for a data product.

Security policies allow individual columns to be protected; Elevated privileges may be required to access sensitive fields within a Data Product.

Security between between topics / streams on an Event Backbone are based upon role-based-access-control; There are opportunities to harmonize RBAC across APIs, databases, and streams/topics.

Data at-rest is encrypted or tokenized based upon the sensitivity of the data within a Data Product.

Figure 2.2: Data Mesh security pattern [10].

The pattern in Figure 2.2 recognizes at the perimeter of Data Products, the following security policies:

- Run-time Environment Security: these policies are heavily dependent upon the runtime environment for the Data Product.
- API Access Control: OpenAPI specifications, allows security schemas and "scopes" (use with OAUTH2) to be defined which dictate credentials to access a Data Product.

Within Data Products, this pattern recognizes:

- RBAC: access to Data Products may be limited to those that have a particular role or group.
- Row-Based Access Control: security policies allow individual rows to be protected.
- ABAC: access based on the attributes of the entities involved.
- Data Encryption and Tokenization: based upon the sensitivity of the data within a Data Product.

There are several supporting capabilities for this pattern:

- An Enterprise Identity Book of Record: the master repository for identity and maintains relationships to assigned groups and roles.

- A Schema / Specification Registry: maintains “scopes” and data / event structures required to support flexible security policies for APIs and events.
- Governance Controls: policies define regulatory and privacy constraints for a data product

This article is quite accurate but does not provide a formulation of a policy or a language for policies to be used in Data Mesh.

Sometimes Data Mesh can be harmful for organizations, especially for the small ones; researches led by some members of the IEEE, proposed a mask-mediator-wrapper architecture that cover the most relevant features of the Data Mesh architecture [19] without damaging the organization. The concept of the Data Mesh is still in the early stages of development and many efforts to implement and use it are expected to have negative consequences for organizations due to a lack of technological guidelines and best practices. So to mitigate the risk of negative outcomes this paper proposes the use of the mask-mediator-wrapper architecture as a Data Mesh driver, providing the basic functionalities that a Data Mesh requires. The advantages of using this wrapper are: low-risk adoption trial, without loss of money and a rapid prototyping of the Data Mesh in an organization. In this paper there is also a small section that stated that the wrapper guarantees: security related to authentication, authorization, confidentiality and data integrity. While security in terms of logical operations (e.g. sharing confidential system information or sending unsafe data) can be assured through a common library for each component type as well as defining a standard set of inter-component exchanges. But the problem is that this paper does not give a clear methodology or examples to achieve this.

2.3. Declarative languages for policies

In the last part of this chapter, we investigate about possible declarative languages that have been already used to define policies.

A first example is Ponder [13] which provides a common means of specifying security policies that map onto various access control implementation mechanisms, firewalls, operating systems and databases. Ponder can also be used for security management activities such as registration of users or logging and auditing events for dealing with access to critical resources or security violations. Key concepts of the language include roles to group policies relating to a position in an organisation, relationships to define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an organisational unit such as a department. The language is flexible, extensible and adaptable to a wide range of management requirements. Ponder is very powerful but we want to have more expressiveness, so we do not use it, but other languages in our work.

In the paper [31] is presented a language for policies based on RuleML, this can be defined as a declarative specification of guidelines, rules of conduct, organization, and behaviour of entities in a given environment. In the paper is presented an extended version of RuleML to handle various policy descriptions embedding rules and constraints marked-up in the RuleML language for Web Services. The proposal goes beyond the description of policies and their execution, but also defines a framework for policy interchange. This language offers great expressiveness but its level of description is too much high for our objectives, it has an expressive power, similar to the natural language, and we want to stay to a more technical level of description.

The researches of the next paper [20] have defined the notion of security policy in terms of rewrite systems. The authors provided a formal definition of declared interference. With these declared interference it is possible to take into account more real-life situations like password verification and communication of encrypted data through a public channel. The paper provides a tool to define very subtle security policies and presents an analysis algorithm for the notion of declared interference. I only consider this paper to better understand the methodology to follow to choose and develop a declarative language and suit it for the case of the Data Mesh.

Despite the next paper [25], is not properly linked to security in Data Mesh, can be relevant to understand the importance of having a sufficiently expressive language for policies. In this case, are policies for web privacy protection. The problems that the authors exposed in the field of web privacy protection are quite similar to the problems of the enforcement of policies in the Data Mesh. Here the Platform for Privacy Preferences (P3P) is a W3C framework for web privacy management. It provides a standard vocabulary that websites can use to describe their privacy practices, but this is rarely used because the languages available to describe user privacy preferences are not sufficiently expressive, P3P policies published by websites are not trusted by users and finally P3P framework does not provide a coherent view of available privacy protection mechanisms to the user. To overcome these problems, the authors use a more expressive policy language based on deontic concepts to describe users privacy-related policies, constraints and preferences. It is introduced a new trust model for websites and its use in user privacy preferences. Finally they are presented sample policies to demonstrate the relevance of their work.

Paper [18] offers an overview of capability-safe languages; components can access a resource only if they possess a capability for that resource. As a result, this can prevent an untrusted component from accessing a sensitive resource by ensuring that the component never acquires the corresponding capability. In order to reason about which components may use a sensitive resource it is necessary to reason about how capabilities propagate through a system. So they are introduced two independently useful semantic security policies to regulate capabilities and describe language-based mechanisms that enforce them. Access control policies restrict which components may use a capability and are enforced using higher-order contracts. Integrity policies restrict which components may influence (directly or indirectly) the use of a capability and are enforced using an information-flow type system. Finally, it is described how programmers can dynamically and soundly combine components that enforce access control or integrity policies with components that enforce different policies or even no policy at all.

The last relevant article [28] describes a privacy policy framework that can represent and reason about complex privacy policies. By using a Common Data Model together with a formal shareability theory, this framework enables the specification of expressive policies in a concise way without burdening the user with technical details of the underlying formalism. There is a policy decision engine which supports two main uses: interfacing with user interfaces for the creation, validation, and management of privacy policies; and interfacing with systems that manage data requests and replies by coordinating privacy policy engine decisions and access to (encrypted) databases using various privacy enhancing technologies.

We have done this analysis of works, to understand the arrival point of the researches in the Data Mesh and formulation of security policies. We can state that basing on our knowledge, till now, the documentation about security in Data Mesh is poor, because the topic is of recent appearance; some aspects are unclear, there are not suitable models to provide a security pattern and a language to define policy in Data Mesh must be provided or chosen among the languages that exists.

3 | Baseline

In this chapter are presented concepts, components and tools that have been useful, in the next chapters, to achieve the goals of this work. In particular, here, we present what is involved in the mechanism of the access of the resources and consequently in the definition of policies to access the data. First, we introduce the XACML architecture, with its components: PEP, PDP, PIP and PAP, that defines a framework to manage access control on a resource. Then is presented OPA, that simplifies the previous architecture and will be used as the PDP. Finally we present Datalog and Rego, the second is an extension of the first and is used to define the policies that OPA applies in order to grant or deny the request sent by the PEP.

3.1. XACML architecture

We do not go in details of the XACML architecture, we only provide a description of its components, that is interesting for us. Figure 3.1 shows these components and their interactions.

- *PEP*: policy enforcement point, which protects a resource and allows access to it only if the evaluation of the request sent to the PDP is positive.
- *PDP*: policy decision point, which takes inside various parameters such as policies and information (users, resources, attributes, metadata), then allows or denies the request from the PEP, and communicates the response to the PEP.
- *PIP*: policy information point, which provides to the PDP useful information to authorize or not the requests.
- *PAP*: policy administration point, which manages policies and provides the relevant policies regarding the access required by the PEP.

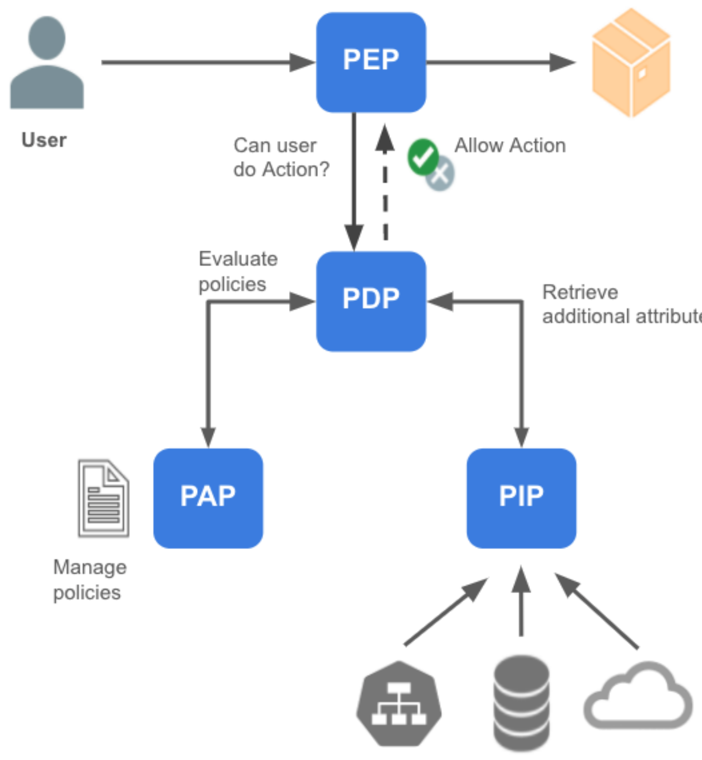


Figure 3.1: XACML architecture [8].

The typical XACML schema behavior is as follows [33]:

- A user wants to access a resource; to do this, he must communicate to the PEP his identity, the resource he wants to access and the operation he wants to do.
- The PEP, having received the request, temporarily freezes it and consults the PDP. The request is contextualized with the information coming from the PIP.
- The PDP, in turn, requests the corresponding security policy from the PAP.
- The PDP, now having all the elements available (the request, the context and the security policy) will decide whether or not to allow access and will communicate the result to the PEP.
- The PEP, having received the decision of the PDP, will authorize (or block) the user's access to the resource.

3.2. Open Policy Agent

The Open Policy Agent (OPA) [5] is an open source, general-purpose policy engine that unifies policy enforcement across the stack. OPA can be used to enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more. OPA decouples policy decision-making from policy enforcement. When a software needs to make policy decisions it queries OPA and supplies structured data (e.g., JSON) as input. OPA generates policy decisions by evaluating the query input against policies and data. The policies in OPA are written in Rego, they are domain-agnostic so it is possible to describe almost any kind of invariant in the policies.

Without OPA, it is needed to implement policy management for a software from scratch; required components such as the policy language (syntax and semantics) and the evaluation engine need to be carefully designed, implemented, tested, documented, and then maintained to ensure correct behaviour and a positive user experience for customers. On top of that it is necessary to consider security, tooling, management, and more.

OPA policies make decisions based on hierarchical structured data. Sometimes we refer to this data as a document, set of attributes, piece of context, or even just “JSON”. Data can be loaded into OPA from outside world in various ways: JSON web tokens, overload input, bundle API, push or pull of the data. All data loaded into OPA from the outside world are called base documents while the values generated by rules are called virtual documents. In Figure 3.2 we have a graphical representation of how OPA works.

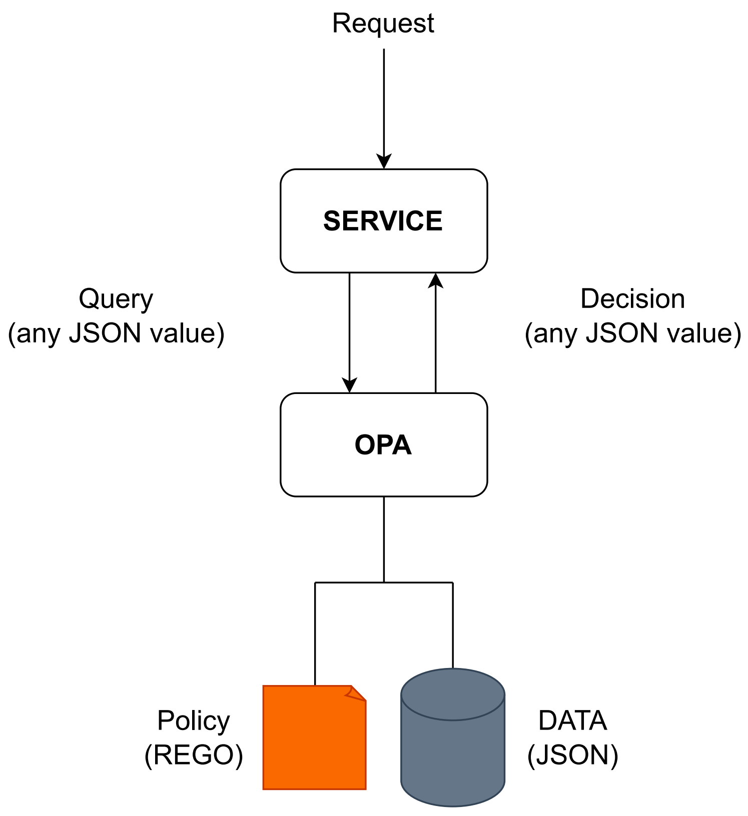


Figure 3.2: Policy decoupling in OPA [5].

With Cloud Native, XACML has become less used, and OPA represents a valid simplification of the XACML schema of Figure 3.1. The concepts of PEP and PDP remain with OPA [1], OPA can be used as a PDP. Our policies, that are the Rego rules in OPA, represent the PAP component. The PEP requests an authorization decision from the PDP. The easiest and most accessible method for requesting an authorization decision is to send an authorization request (or query) to the PDP. This is the suggested method of integrating PEPs with a PDP, because it is a familiar pattern for exposing functionality to multiple services in an application. The only function of the PEP in this pattern is to forward the information that the authorization request needs and then, effectively giving access to the resource in case of positive response from the PDP. Concerning the PIP, if OPA is used as a PDP, it can accept arbitrary JSON data as overload input that is passed as part of the request. The data inside of the organization, can be managed by an S3 storage such as MinIO [4].

3.3. Datalog

Datalog is a declarative logic programming language, introduced in 1984 [11] [7]; it is syntactically a subset of Prolog, but generally uses a bottom-up rather than a top-down evaluation model. In Datalog with respect to Prolog, there are no symbols of function and there is a non-procedural model of evaluation. Datalog is a series of rules, each rule is composed by a head (LHS) on the left of "**:-**" and a body (RHS) on the right of "**:-**" ; a rule has the following structure:

$$\mathrm{P} \text{ :- } \mathrm{P}_1,\ \mathrm{P}_2,\ \mathrm{P}_3,\ \mathrm{P}_4,\ \ldots,\ \mathrm{P}_{\mathrm{N}}.$$

Each P_{i} is called a fact and it is an instance of a predicate composed by:

- Its name.
- A list of arguments between round brackets ( ) :
  - Constants.
  - Variables.
  - Symbol do not care (_) which cannot appear in the head.

To guarantee safety, all the variables in the LHS must appear in the RHS. LHS is true if RHS is true.

In a rule, the notation: $\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_{\mathrm{n}}$ means intersection $\cap$.

The union of rules $\mathrm{P} = \mathrm{R} \cup \mathrm{S}$ is expressed by the following notation:

$$P(X,Y) \text{ :- } R(X,Y).$$

$$P(X,Y) \text{ :- } S(X,Y).$$

The difference of rules P = R - S, is expressed by the following notation:

$$P(X,Y) \text{ :- } R(X,Y), \neg\, S(X,Y).$$

Recursive queries are expressed by the following notation:

$$P(X,Y) \text{ :- } R(X,Y).$$

$$P(X,Y) \text{ :- } S(X,Z), R(Z,Y).$$

3.4. Rego

From the official documentation [5], is stated that Rego was inspired by Datalog. It extends Datalog to support structured document models such as JSON.

Rego queries are assertions on data stored in OPA, these queries can be used to define policies that enumerate instances of data that violate the expected state of the system. The policies written in Rego are easy to read and write. Rego focuses on providing powerful support for referencing nested documents and ensuring that queries are correct and unambiguous. Rego is declarative so policy authors can focus on what queries should return rather than how queries should be executed. These queries are simpler and more concise than the equivalent in an imperative language.

We resume in Table 3.1 the commands useful to define some examples of policies in the next chapters.

Command	Description
future.keywords.if	True if the condition is satisfied
input.	Values taken from input.json
data.	Values taken from data.json
default	Default value of a rule
true/false/null	Pre-defined values

Table 3.1: Useful Rego commands.

Here below a simple example where Rego is used to make access decisions about which users are allowed to access information in a fictional payroll microservice. The corresponding policy in natural language is: Employees can read their own salary and the salary of anyone who reports to them [1].

```
default allow = false

allow = true {
input.method == "GET"
input.path = ["getSalary", user]
input.user == user}

allow = true {
input.method == "GET"
input.path = ["getSalary", user]
managers := data.managers[input.user][_]
contains(managers, user) }
```

Listing 3.1: Example of policy in Rego [1].

4 | Proposed approach

In this chapter we propose an approach, giving definitions and graphical models with the related explanations, with the aim of defining a framework to access the data in federated Data Mesh; by doing this, we try to solve the problems presented in chapter 1. First, we introduce the components of the Data Mesh taken by the existing literature, then we can proceed by defining the interactions between the domain teams and the data products, giving the possible scenarios, and finally we provide the models to define the policies.

4.1. Data Mesh components and design patterns

Recalling this topic from chapter 1, and adding some other details, we can state that the Data Mesh architecture is a decentralized data architecture where the data is a product and the data producers act as data product owners. Historically, a centralized infrastructure team would maintain data ownership across domains; now there is still a centralized data governance team but only to enforce common standards and procedures around the data. Domain teams become responsible for their ETL data pipelines under the Data Mesh architecture [24].

The following components are defined [12]:

- The *data product* is a published dataset that can be accessed by other domains, similar to an API. It is described with metadata and the *domain team* is responsible for the operations on the data product during its entire lifecycle.
- The *data contract* is a formal agreement between two parties to use a data product. It specifies the guarantees about a provided dataset and expectations concerning data product usage.
- The *data platform*, which needs functions to ingest, store, query, and visualize data as a self-service. An advanced data platform for Data Mesh provides additional domain-agnostic data product capabilities for creating, monitoring, discovering, and accessing data products, for example, by using a data catalog. The platform can also support policy automation.

- The *federated governance group* is a guild consisting of the representatives of all teams taking part in the Data Mesh. They agree on global policies, defining the rules on how the domain teams must build their data products.
- The *enabling team* consists of specialists with extensive knowledge on data analytics, data engineering, and the self-serve data platform. They help other domain teams.
- The *mesh* emerges when domain teams use other team's data products. Data from multiple domain teams can be aggregated to build comprehensive reports and new data products.

Data Mesh can have different design patterns, the original proposed by Dehghani is the fine-grained fully federated mesh [32]. Data in this topology is owned, managed and shared by each individual domain. Each data product is approached as an architectural quantum, this means that you may instantiate many and different small data product architectures for serving and pulling data across domains. Organization could have difficulties in the adoption of this design pattern, so alternative patterns were proposed. Fine-grained and fully governed mesh: domains distribute and route data via a central logic entity. Hybrid federated mesh: federation on the consuming side with a bit of centralization on the source-system aligned side. Value chain-aligned mesh: aligning domains around a full chain of business's activities. Coarse grained aligned mesh: alignment around business units, regions and capabilities. Coarse grained and governed mesh: the same as before while governing cross boundary distribution of data.

4.2. The three levels to access the data

After providing the components of the Data Mesh, we define three levels for the management of the access of the data:

- The *Business level* is the level where the policies to access and transforming the data are defined in natural language, such as English. These policies have not a technical form and are general and valid for all the domain teams, they can be formulated by the federated governance group. On the contrary, the other two levels are implemented inside each domain team.
- The *Technical level* is the level where the policies, formulated before, are defined in Rego. The rules for the policies are formulated basing on the directives expressed in the previous level and on relevant facts, e.g., attributes and metadata, transmitted to OPA that acts as the PDP; this component receives a request in JSON by the PEP and, by applying the policies, returns to it a decision in JSON on whether a

particular access is permitted or not. We assume that the PDP trusts the PEP, for the validation of the identity of the data consumer, who can be internal or external to the organization.

- The *Enforcement level* is the level where the deployment effectively happens. Information about what are the various PEPs are provided. A PEP is responsible for granting the effective access to the resource basing on the evaluation of the PDP. The PEP only needs to have inside a logic to interpret the request of a user and formulate the corresponding request in JSON to query the PDP, then it will interpret the decision in JSON provided by the PDP. Depending on the outcome of the PDP, the PEP grants or denies access to the resource. It is possible that an organization could require to another organization other information that does not have and are necessary to perform operations on the data product e.g., the attributes of an external user that makes the request. At this level it is necessary to define how and where the PEPs of the organization require the previous information. We suppose that the data of an external organization are passed in input as part of the query of the PEP to the PDP.

4.2.1. Location of PDP and PEP

The two main components involved in the access of a data product are the PDP and the PEP. As we can see in Figure 4.1, we suppose that each organization is served by a central PDP, and each data product, provided by a domain team in the organization, has its own PEP inside the organization.

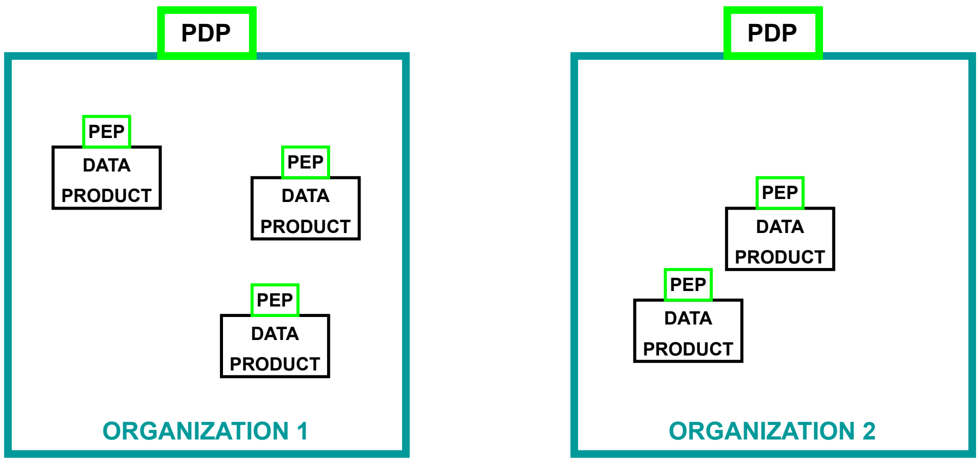


Figure 4.1: Location of PDP and PEP.

4.3. Scenarios

The simplest scenario is when the required data product is not an aggregation of other data products. Otherwise, if the requested data product is an aggregation of other secondary data products, we can have three additional scenarios. The first is when the data product is an aggregation of other secondary data products and they are all inside the same organization, the second is when the secondary data products are all outside the organization, the third is when a data product aggregates other data products both inside and outside the organization.

4.3.1. Simple data product

In this case, the PEP of the data product communicates with the central PDP of the organization, querying it when a data consumer requires the data product. The request can be done by an internal or external member to the organization.

The model in Figure 4.2, regards a request coming from a data consumer who works inside the organization; if the data is encrypted, in addition of having the necessary requisites, the data consumer must also own the correct public key, as part of his attributes, to access it. Being the data consumer inside the organization, all the necessary attributes and metadata to make the evaluation are easy to retrieve and pass to OPA because are inside the organization.

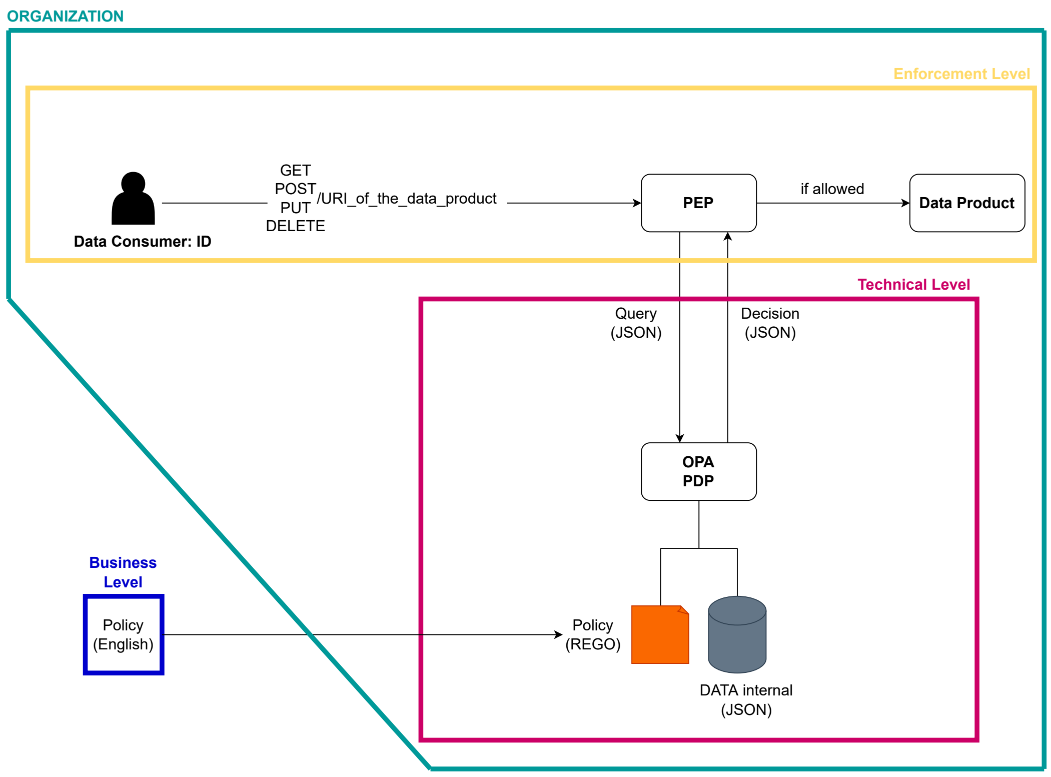


Figure 4.2: Data consumer inside the organization.

The model, in Figure 4.3, regards a request of a data consumer outside the organization. In addition to what said before, the PEP will pass to OPA some external data inside the query, specifically the attributes of the data consumer which are outside the organization, then OPA, combining internal and external data with the policies in Rego, provides the decision. In general, being the data consumer external to the organization, the data product could be anonymized or pseudoanonymized before granting access to it, to avoid the data consumer to retrieve sensitive information about it.

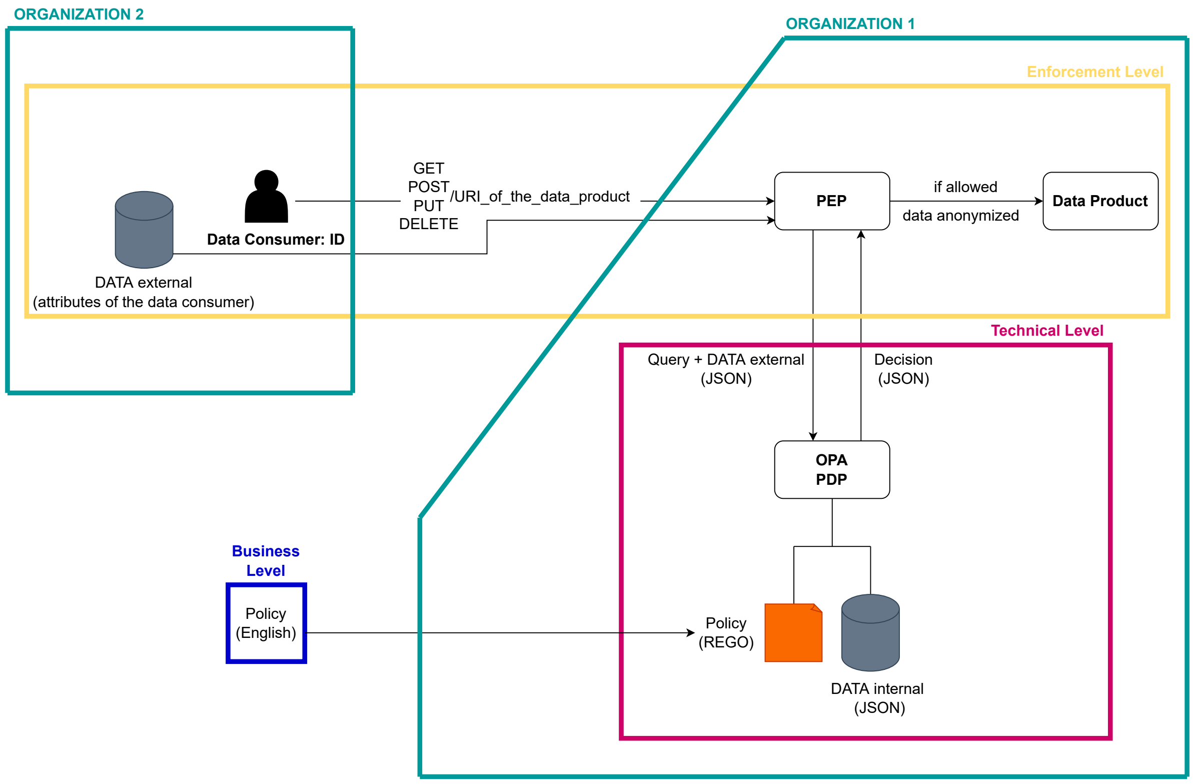


Figure 4.3: Data consumer outside the organization.

4.3.2. Aggregation of data products

In case of a data product which is an aggregation of secondary data products, the considerations about the data consumer internal/external are still valid but now, the primary, queried, data product becomes itself a sort of data consumer of the secondary data products.

In Figure 4.4 the secondary data products are inside the organization. The PEP of the primary, requested, data product interacts with the central PDP of the organization and the PEPs of the other two secondary data products. The PEP receives the decisions to access the secondary data products from the central PDP, if the PDP grants access only to one of the two secondary data products, then the PEP of such data product allows to access it even if the other PEP cannot give access to its data product. In this way, the data consumer can obtain a partial reconstruction of the primary data product.

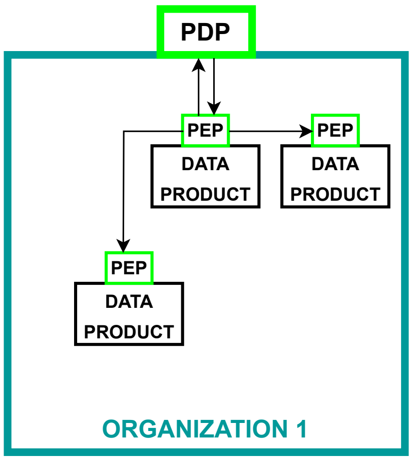


Figure 4.4: Aggregated data products inside the organization.

In Figure 4.5 we have a data product which is an aggregation of data products outside the organization, so the access to the secondary data products is provided not only by the PDP of the organization but by the interaction of this with the PDP of the organization that contains the two secondary data products. If the access to one of these two data products is forbidden to the data consumer of the primary data product, he cannot be able to access both the two secondary data products even if he has the requisites to access one of it.

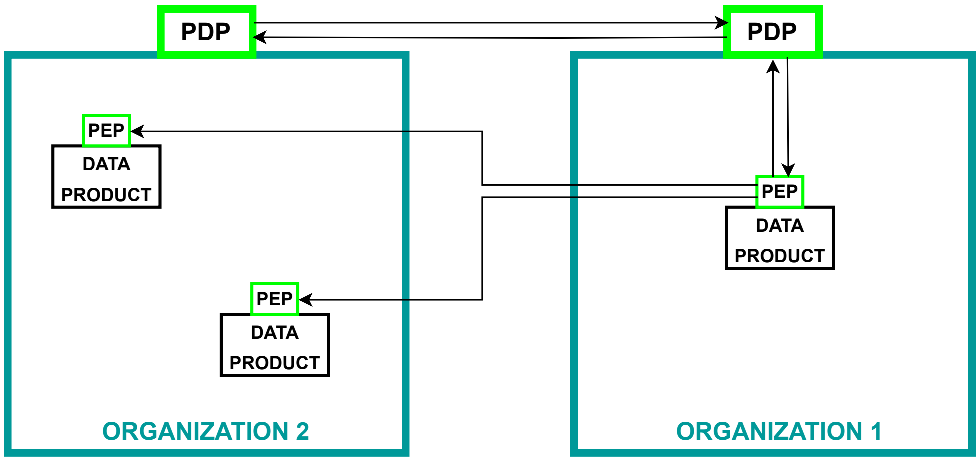


Figure 4.5: Aggregated data products outside the organization.

In case of secondary data products both inside and outside the organization, Figure 4.6, there could be the possibility that the data consumer can access only some of it. We took the case of two secondary data products, if the data consumer can access only the secondary data product inside the organization but not the external one, he can however retrieve this secondary data product inside the organization. On the contrary, if he can access the secondary data product external to the organization but not the one inside the organization, he cannot be able to access anything even if he has the requisites to access the external one.

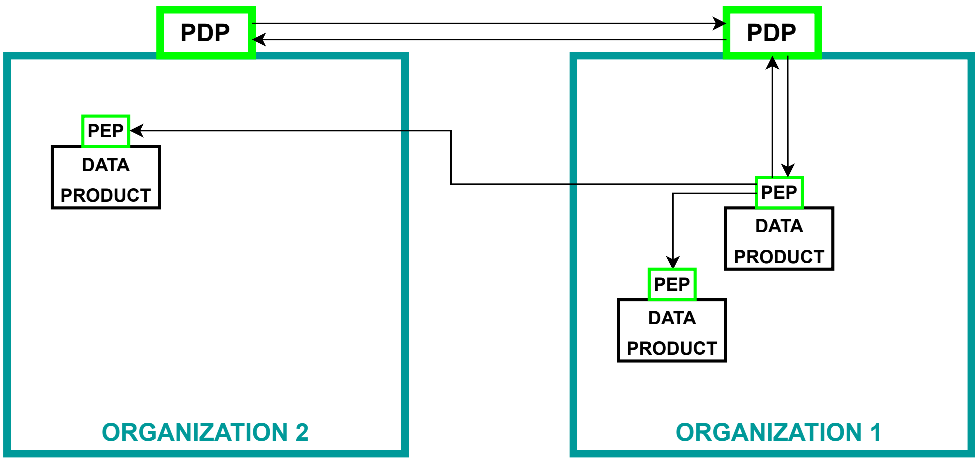


Figure 4.6: Aggregated data products inside and outside the organization.

4.4. Definition of the policies

The following E-R models give us an overview on the relationships of the policies with the members and the resources of a Data Mesh architecture.

In Figure 4.7, we design the *Consumer/Provider model* that shows the entities involved in the access of a data product. A data provider provides one or more data products; the data product can be accessed according to the directives imposed by the policy agreement that the data consumer makes with the data provider. As said in the previous chapters, the policies adopt the ABAC paradigm, where the constraints to access the data product are made according to the value of the metadata of the data product and the attributes of the data consumer and the data provider.

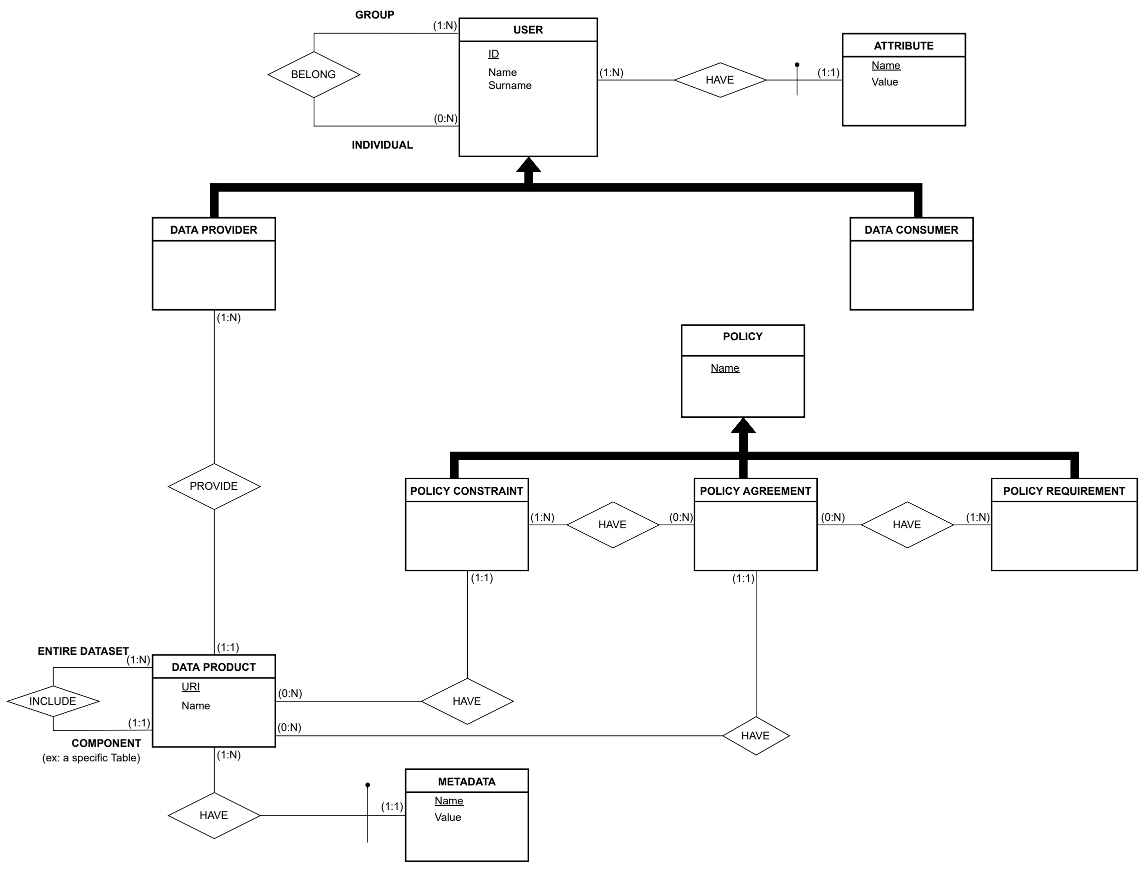


Figure 4.7: Consumer/Provider model.

In detail, the entities are the following:

- The general entity of USER with two child entities: the DATA CONSUMER and the DATA PROVIDER; a user can belong to a group or be an individual person.
- The ATTRIBUTE that is an attribute associated to a user, each user can have one or more attributes, each attribute has a corresponding value.
- The DATA PRODUCT, whose definition was given before in this chapter, it is queried by the data consumer, who wants to access it to compute some operations, and is provided by the data provider.
- The METADATA of the data product are features of the data product and they have a value.
- The POLICY which have three child entities:
  - POLICY CONSTRAINT, the constraints on the data product.
  - POLICY REQUIREMENT, the requirements on the data product.

– POLICY AGREEMENT that is the policy resulting by the match making between the previous two.

The E-R model showed in Figure 4.8, is the *model of the definition of the policy at the technical level*, that provides the entities involved in the definition of the policies, written in Rego, that grant or deny the access to the data product. We can say that this model is a representation of the entity POLICY AGREEMENT of the previous model. Some of these entities are the same as the previous model but, the difference is that here refer to theoretical entities that appears in the definition of policies and not to concrete physical entities as before.

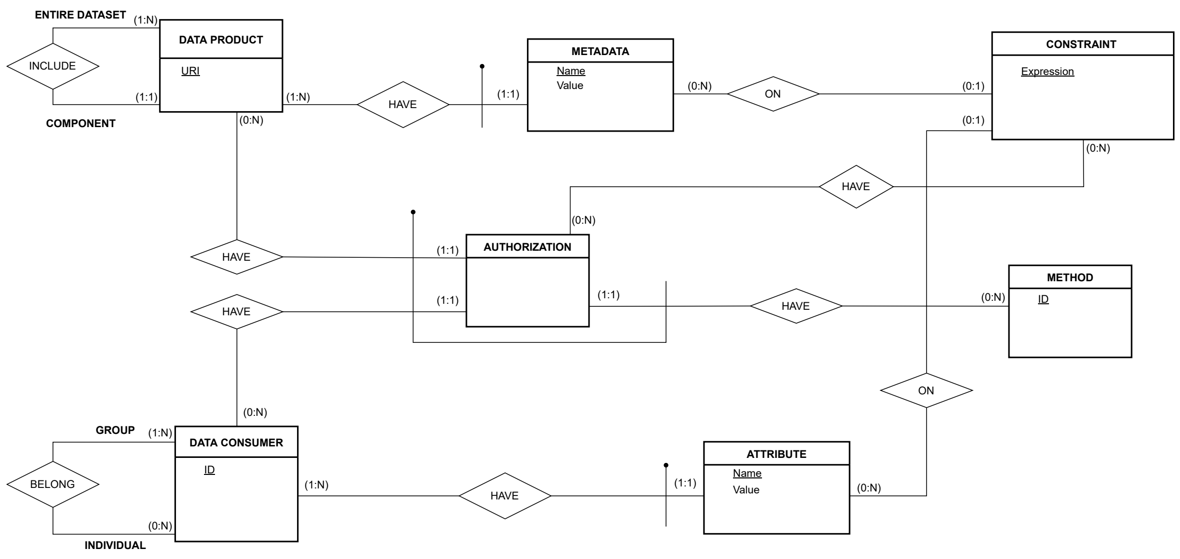


Figure 4.8: Model of the definition of the policy at the technical level.

To define a policy, we need the following entities:

- The DATA PRODUCT that is the resource to which the actions applied, it can have more than one metadata.
- The METADATA is the entity that define a feature of the data product with the corresponding value.
- The DATA CONSUMER is the entity who wants to do an action, he can have more than one attributes.
- The ATTRIBUTE is the entity that define an attribute of the consumer with the corresponding value.
- The CONSTRAINT entity is a constraint on the value of a metadata or of an

attribute, the values are compared to each other or to fixed literal or numerical values.

- The METHOD refers to both an access mode or an operation that is done on a data product.
- The AUTHORIZATION which is a weak entity and does not exist without a method, a data product and a data consumer. Basing on the constraints (zero or more), lets a certain user or group of users to perform some actions on a certain data product.

5 | Examples and tests

In this chapter we provide examples of policies of the business level that are transformed in the correspondent policies of the technical level. For simplicity, we consider only not aggregated data products. The policies are formulated following the entities and relationships of the E-R model of Figure 4.8 and consequently, with respect of the ABAC paradigm. They will be tested by running OPA as the PDP.

5.1. Policies at the business level

We start by defining policies at the business level written in English. The policies regard authorizations in hospital organizations. Here are some examples:

- The doctors can read and write their data.
- The research doctors can read every data even if it is external to the hospital where they work.
- The data generated by a research doctor can be read by other workers in the hospital where the research doctor works.
- The data provided by a research doctor is encrypted.
- The data is anonymized if the request is done by an external member of the organization.

5.2. Policies at the technical level

We reformulate the previous policies, at the technical level, in Rego. The file *rules.rego* defines the rules for the constraints and the authorizations by taking inside the values contained in the *data.json* file and in the *input.json* file.

```
package rules
import future.keywords.if

import input.data_consumer
import input.method
import input.data_product

#import the attributes of the data_consumer if is external
import input.attributes

#constraints

user_is_doctor if data.attributes[input.data_consumer].role == "doctor"
user_is_doctor if input.attributes[input.data_consumer].role == "doctor"

user_is_researcher if data.attributes[input.data_consumer].researcher == true
user_is_researcher if input.attributes[input.data_consumer].researcher == true

user_is_owner if input.data_consumer == data.metadata[input.data_product].owner

resource_provided_by_researcher if data.attributes[data.metadata[input.data_product].owner].researcher == true

same_company if data.attributes[input.data_consumer].company == data.metadata[input.data_product].company

#authorizations

default allow := false
default encrypted := false
default anonymized := false

#The doctors can read and write their data.
allow if {
  user_is_doctor
  user_is_owner
  input.method == "GET"
}

allow if {
  user_is_doctor
  user_is_owner
  input.method == "POST"
}

```

```
#The research doctors can read every data.
allow if {
  user_is_doctor
  user_is_researcher
  input.method == "GET"
}

#The data generated by a research doctor can be read by other workers in
    the hospital where the research doctor works.
allow if {
  resource_provided_by_researcher
  same_company
  input.method == "GET"
}

#The data provided by a research doctor is encrypted.
encrypted if {
  data.attributes[data.metadata[input.data_product].owner].role == "
   doctor"
  data.attributes[data.metadata[input.data_product].owner].researcher ==
    true
}

#The data is anonymized if the request is done by an external member of
   the organization.
anonymized if {
  not same_company
}
```

Listing 5.1: rules.rego

In the *data.json* file are defined some facts that can be useful to formulate the policies. The information contained in this file are internal to the organization. The facts are the list of the workers, identified by their ID, with their attributes and the list of the data products, identified by their URI, with their metadata; we give them some values. It is possible to define additional attributes and metadata. Among the attributes that a user could have, there are: his years of experience and the years working for the organization. Among the metadata of a data product, there could be: the format of the data, the type of the data, the original URI of the data, the date of the last edit and the source of that data.

```
{
    "attributes": {
        "a2b67": {"company": "c1", "role": "doctor", "researcher": true,
    "public key": "p.key_a2b67"},
        "98uio": {"company": "c1", "role": "doctor", "researcher": false
, "public key": "p.key_98uio"}
        },
    "metadata": {
        "https://www.db1.com/resource1/": {"name": "resource1", "db": "
db1", "owner": "a2b67", "company": "c1"},
      "https://www.db1.com/resource2/": {"name": "resource2" , "db": "
db1", "owner": "98uio", "company": "c1"},
      "https://www.db1.com/resource3/": {"name": "resource3", "db": "db1
", "owner": "98uio", "company": "c1"},
      "https://www.db3.com/resource6/": {"name": "resource6", "db": "db3
", "owner": "a2b67", "company": "c1"}
        }
}
```

Listing 5.2: data.json

In our framework, the query is made by the PEP to the OPA-PDP, then the PEP waits for the answer. The query sent to OPA is in JSON format and contains: the data consumer who makes the request, the method to access the data product and the data product that is required, if the data consumer is external, he must also provide his attributes and the PEP, will transmit them in the query formulation. Changing the values of the query, the *allow*, defined in the *rules.rego* file, becomes true or remains false, if remains false the user cannot access the resource with that method. If an external user for some reasons does not provide his attributes, the PEP cannot be able to pass them to the PDP and *allow* remains false even if the user would have the right requisites to access the data. Changing the input modifies also the values of *encrypted* and *anonymized*. If *encrypted* is true, the data consumer must have the right public key in order to make the operation on the data. If *anonymized* is true, the data is stripped of any identifiable information, this is an additional security measure against data consumers external to the organization.

```
{
"input":{
    "data_consumer": id_of_the_data_consumer,
    "attributes": { (attributes) },
    "method": the_method (GET/POST/PUT/DELETE),
    "data_product": uri_of_the_data_product
     }
}
```

Listing 5.3: input.json

5.3. Tests

Here, OPA is run as a server with the information contained in the file *data.json* that are the data inside the organization. We use the following command: **./opa run data.json rules.rego -s**. Then we can provide some queries in JSON to test the rules defined before.

5.3.1. Example 1

A doctor internal in the organization, makes the request of GET or POST to a data of which he is the owner and *allow* becomes true. *anonymized* remains false because the request is internal in the organization while *encrypted* becomes true because, in that case, the owner is a research doctor otherwise would remain false.

Query:

```
{
    "input": {
        "data_consumer": "a2b67",
        "method": "GET",
        "data_product": "https://www.db1.com/resource1/"
    }
}
```

Listing 5.4: Query 1.

Decision:

```
{
    "result": {
        "allow": true,
        "anonymized": false,
        "encrypted": true,
        "resource_provided_by_researcher": true,
        "same_company": true,
        "user_is_doctor": true,
        "user_is_owner": true,
        "user_is_researcher": true
    }
}
```

Listing 5.5: Decision 1.

5.3.2. Example 2

A doctor, who is not the owner of a resource, cannot do the POST of another resource even if is a research doctor and is internal to the organization. So in this case, *allow* remains false.

Query:

```
{
    "input": {
        "data_consumer": "a2b67",
        "method": "POST",
        "data_product": "https://www.db1.com/resource2/"
    }
}
```

Listing 5.6: Query 2.

Decision:

```
{
    "result": {
        "allow": false,
        "anonymized": false,
        "encrypted": false,
        "same_company": true,
        "user_is_doctor": true,
        "user_is_researcher": true
    }
}
```

Listing 5.7: Decision 2.

5.3.3. Example 3

A data consumer, external to the organization, who is not a research doctor, cannot do the GET of a resource of the organization and consequently, *allow* remains false.

Query:

```
{
    "input": {
        "data_consumer": "56tywe",
        "attributes": {
            "56tywe": {
                "company": "c2",
                "role": "nurse",
                "researcher": null,
                "public key": "p.key_56tywe"
            }
        },
        "method": "GET",
        "data_product": "https://www.db1.com/resource1/"
    }
}
```

Listing 5.8: Query 3.

Decision:

```
{
    "result": {
        "allow": false,
        "anonymized": true,
        "encrypted": true,
        "resource_provided_by_researcher": true
    }
}
```

Listing 5.9: Decision 3.

5.3.4. Example 4

A research doctor, external to the organization, providing his attributes, can do the GET of a resource. The resource will be *anonymized* because the request comes from outside. Here *encrypted* is true because, in this case, the resource belongs to a research doctor. If the data consumer wants to do a POST, *allow* remains false.

Query:

```
{
    "input": {
        "data_consumer": "24hj9",
        "attributes": {
            "24hj9": {
                "company": "c2",
                "role": "doctor",
                "researcher": true,
                "public key": "p.key_24hj9"
            }
        },
        "method": "GET",
        "data_product": "https://www.db1.com/resource1/"
    }
}
```

Listing 5.10: Query 4.

Decision:

```
{
    "result": {
        "allow": true,
        "anonymized": true,
        "encrypted": true,
        "resource_provided_by_researcher": true,
        "user_is_doctor": true,
        "user_is_researcher": true
    }
}
```

Listing 5.11: Decision 4.

If for some reasons, an external data consumer, or his organization, does not provide his attributes, *allow* remains always false and he cannot access the resource even if, he would have the right requisites to access it.

6 | Conclusions

Given the results obtained, we conclude our investigation. In this work, we have provided a framework that manages the interactions of domain teams, organizations, data consumers and data products. We have defined the structure of the security policies used in the Data Mesh architecture to regulate the access to the data, choosing a declarative language like Rego to define it, and using OPA as the policy engine; we have provided some examples and carry out some tests.

The future development of this work, could be the possibility to extend the investigation to a wider context, the data sovereignty, in particular whether the presented framework must be modified to be conformed to data sovereignty. Data sovereignty refers to the idea that a country or jurisdiction has the authority and right to govern and control the data generated within its borders. This means that the government has the power to regulate the collection, storage, processing, and distribution of data that originates within its territory. In 2016, the EU Parliament, already approved some data sovereignty measures within a General Data Protection Regulation (GDPR) [3]. This regulatory package homogenizes data protection policies for all European Union members. Nowadays there are still some key challenges of data sovereignty. The cross-border data flows: data sovereignty can make it more difficult to transfer data across borders; this can result in increased costs and complexity for businesses that operate globally with flows of data across different jurisdictions. The data localization requirements: some countries may require that certain types of data be stored and processed within their jurisdiction, which can be challenging for businesses that operate in multiple regions. The possibility of cybersecurity risks: data sovereignty can increase cybersecurity risks, particularly if data is stored in a single location or jurisdiction. Finally, data sovereignty can create challenges for international data sharing agreements, particularly if countries have different requirements for data protection and storage. To overcome these problems, there is a recent project called Gaia-X [2], the goal of this project is to provide a secure and federated data infrastructure that stands for European values, digital sovereignty of the data owners, interoperability of different platforms. Within this ecosystem, it will be possible to provide, share, and use data within a trustworthy environment. At the European level, Gaia-X invests money to create data spaces. A data space is a virtual and interoperable system between different cloud service providers, which allows its users to exchange data when needed. Data spaces can be organized by supply chain, sector or scope.

Bibliography

[1] Aws prescriptive guidance multi-tenant saas authorization and api access control. URL https://docs.aws.amazon.com/prescriptive-guidance/latest/saas-multitenant-api-access-authorization/welcome.html.

[2] What is gaia-x. URL https://gaia-x.eu/what-is-gaia-x/.

[3] Gdpr - regolamento 2016/679 - garante privacy. URL https://www.garanteprivacy.it/regolamentoue.

[4] Minio. URL https://min.io/.

[5] Open policy agent. URL https://www.openpolicyagent.org/docs/latest/#overview.

[6] M. U. Aftab, Z. Qin, Zakria, S. Ali, Pirah, and J. Khan. The evaluation and comparative analysis of role based access control and attribute based access control model. In *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 35–39, 2018. doi: 10.1109/ICCWAMTIP.2018.8632578.

[7] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems concepts, languages & architectures*, pages 77–80. The McGraw-Hill Companies, 1999.

[8] A. Berben. Enterprise-level policy enforcement with opa (open policy agent) and gloo edge. URL https://www.solo.io/blog/opa-open-policy-agent-gloo-edge/.

[9] J. Bode, N. Kühl, D. Kreuzberger, and S. Hirschl. Data mesh: Motivational factors, challenges, and best practices. *arXiv preprint arXiv:2302.01713*, 2023.

[10] E. Broda. Data mesh/data product security pattern, 8 2022. URL https://towardsdatascience.com/data-mesh-data-product-security-pattern-c5b93a27e82e.

[11] S. Ceri, G. Gottlob, and L. Tanca. *Logic programming and databases*, pages 1–14. Springer Science & Business Media, 2012.

[12] J. Christ, L. Visengeriyeva, and S. Harrer. Data mesh from an engineering perspective. URL `https://www.datamesh-architecture.com/#why`.

[13] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks: International Workshop, POLICY 2001 Bristol, UK, January 29–31, 2001 Proceedings*, pages 18–38. Springer, 2001.

[14] Z. Dehghani. How to move beyond a monolithic data lake to a distributed data mesh, 2019. URL `https://martinfowler.com/articles/data-monolith-to-mesh.html`.

[15] Z. Dehghani. Data mesh paradigm shift in data platform architecture., 2020. URL `https://www.youtube.com/watch?v=52MCFe4v0UU`.

[16] Z. Dehghani. Data mesh principles and logical architecture, 2020. URL `https://martinfowler.com/articles/data-mesh-principles.html`.

[17] Z. Dehghani. *Data Mesh.* O'Reilly Media, Inc., 2022. ISBN 9781492092391.

[18] C. Dimoulas, S. Moore, A. Askarov, and S. Chong. Declarative policies for capability control. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 3–17. IEEE, 2014.

[19] J. Dončević, K. Fertalj, M. Brčić, and M. Kovač. Mask-mediator-wrapper architecture as a data mesh driver. *arXiv preprint arXiv:2209.04661*, 2022.

[20] R. Echahed and F. Prost. Security policy in a declarative style. In *Proceedings of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 153–163, 2005.

[21] A. Goedegebuure, I. Kumara, S. Driessen, D. Di Nucci, G. Monsieur, W.-j. v. d. Heuvel, and D. A. Tamburri. Data mesh: a systematic gray literature review. *arXiv preprint arXiv:2304.01062*, 2023.

[22] V. C. Hu, D. Ferraiolo, D. R. Kuhn, et al. *Assessment of access control systems.* US Department of Commerce, National Institute of Standards and Technology . . . , 2006.

[23] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162):1–54, 2013.

[24] IBM. What is a data mesh? URL `https://www.ibm.com/topics/data-mesh`.

[25] P. Kolari, L. Ding, G. Shashidhara, A. Joshi, T. Finin, and L. Kagal. Enhancing web privacy protection through declarative policies. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 57–66. IEEE, 2005.

[26] I. A. Machado, C. Costa, and M. Y. Santos. Data mesh: concepts and principles of a paradigm shift in data architectures. *Procedia Computer Science*, 196:263–271, 2022.

[27] I. A. Machado, C. Costa, and M. Y. Santos. Data mesh: concepts and principles of a paradigm shift in data architectures. *Procedia Computer Science*, 196:263–271, 2022.

[28] K. Martiny, D. Elenius, and G. Denker. Protecting privacy with a declarative policy framework. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 227–234. IEEE, 2018.

[29] A. Mott and C. Ng. Practical security and policy-based governance in a data mesh, 2022. URL `https://www.starburst.io/blog/practical-security-and-policy-based-governance-in-a-data-mesh/`.

[30] N. K. Mukhi and P. Plebani. Supporting policy-driven behaviors in web services: experiences and issues. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 322–328, 2004.

[31] S. Ross-Talbot, S. Tabet, S. Chakravarthy, and G. Brown. A generalized ruleml-based declarative policy specification language for web services. In *W3C Workshop on Constraints and Capabilities for Web Services*, 2004.

[32] P. Strengholt. Data mesh: Topologies and domain granularity, 5 2022. URL `https://towardsdatascience.com/data-mesh-topologies-and-domain-granularity-65290a4ebb90`.

[33] Wikipedia. Xacml, 2022. URL `https://it.wikipedia.org/wiki/XACML`.

[34] Y. Zhou, Q. Zhao, and M. Perry. Policy enforcement pattern. In *Proceedings of the Conference on Pattern Languages of Programs*, pages 1–14. Citeseer, 2002.

List of Figures

2.1 Policy enforcement pattern [34]. 6
2.2 Data Mesh security pattern [10]. 10

3.1 XACML architecture [8]. 16
3.2 Policy decoupling in OPA [5]. 17

4.1 Location of PDP and PEP. 23
4.2 Data consumer inside the organization. 25
4.3 Data consumer outside the organization. 26
4.4 Aggregated data products inside the organization. 27
4.5 Aggregated data products outside the organization. 27
4.6 Aggregated data products inside and outside the organization. 28
4.7 Consumer/Provider model. 29
4.8 Model of the definition of the policy at the technical level. 30

List of Tables

3.1 Useful Rego commands. 19

Listings

3.1 Example of policy in Rego [1]. 20
5.1 rules.rego 33
5.2 data.json 35
5.3 input.json 36
5.4 Query 1. 37
5.5 Decision 1. 37
5.6 Query 2. 38
5.7 Decision 2. 38
5.8 Query 3. 38
5.9 Decision 3. 39
5.10 Query 4. 39
5.11 Decision 4. 40

Acknowledgements

In primo luogo, vorrei ringraziare il mio relatore, Pierluigi Plebani e il mio correlatore, Mattia Salnitri, per la grande disponibilità e competenza oltre al prezioso aiuto nella redazione di questa tesi.

Il più grande ringraziamento va ai miei genitori, che mi hanno dato la possibilità di studiare e raggiungere questo traguardo sostenendomi e incoraggiandomi sempre; mi sento fortunato ad avere genitori come loro.

Un ringraziamento speciale va a Giulia, per il suo sostegno morale e per tutti i momenti felici trascorsi insieme e che hanno reso meno pesante la conclusione del mio percorso universitario.

Infine, ringrazio tutti i miei amici, sia quelli di vecchia data che quelli più recenti, per i momenti spensierati passati insieme.