Automating Policy-as-code Generation Pipeline for Data Products: an OpenAPI-driven Rego generator*

Matteo Brambilla, Matteo Falconi, Valeria Maria Fortina, Pierluigi Plebani, and Monica Vitali

Politecnico di Milano,
Piazza Leonardo da Vinci 32, 20133 Milan, Italy
[firstname.lastname]@polimi.it

Abstract. At the core of data mesh there are data products: autonomous units that expose data via REST-based programmatic interfaces, while preserving the provider sovereignty and ensuring the consumer trust. However, despite their flexibility, adopting data products raises challenges in defining and enforcing fine-grained, customizable access policies that apply to different domains, such as security and usage. This work investigates the potential of a policy-as-code approach to support policy specification and automated compliance in data product architectures. We propose an extension to the OpenAPI specification that (i) allows security and transformation policies to be represented directly within a data product's OpenAPI description, (ii) feeds an automated pipeline to generate Open Policy Agent (OPA) Rego rules.

Keywords: Data Mesh · Service Policy · Service Description.

1 Introduction

A key element in data mesh is the *data product* [10]: the architectural element that (i) provides access to a set of data through a programmatic interface (e.g., REST), (ii) provides computational resources to manage data, (iii) and enables organisations adopting the data mesh paradigm to facilitate data sharing both internally and with external entities while ensuring data sovereignty and data trustworthiness.

In this context, the customizability in the way the data are offered and exchanged between the data provider and each data consumer is a key point. In fact, a data product can be of interest to multiple data consumers, but, at the same time, different aspects such as the delivery methods, the level of visibility on the data, and access mechanisms can be tailored for each [9].

Exploiting the parallelism between services and data, this work investigates the potential of adopting a policy-as-code (PaC) [8] approach to define access-related aspects of data and automate compliance verification. To this aim, Open

* This work has been funded by the European Union (TEADAL, 101070186)

Policy Agent (OPA)[1] has been chosen as the reference authorization architecture based on Rego: a declarative language used to express policies.

The objective of this work is manyfold. Firstly, an OpenAPIextension able to express the policies related to the access to a data product. Secondly, a method to generate a set of Rego-compliant rules starting from the proposed OpenAPI extension. Thirdly, a tool able to support the proposed method that has been tested in different scenarios related to different application domains.

In the rest of the paper, the related work are presented in Sec. 2, while Sec. 3 introduces our overall approach. The provided extension of OpenAPI is described in Sec. 4, while Sec. 5 introduces the method used to generate Rego rules. Technical details about the tool supporting the policy management is described in Sec. 6. Finally, Sec. 7 outlines possible future directions.

2 Related work

If on the one hand there is a strong push to make data sharing between organizations easier [4], on the other hand, the need to preserve the value of the data requires making them available under certain conditions [5]. At a high level, there are several approaches that guide data governance, also with respect to the definition of policies that regulate sharing [1].

Concerning the technical aspects, the Data Space Blueprint [3] offers a model to enable controlled data sharing based on, among others, the Open Digital Rights Language (ODRL)[2]: a W3C standard for representing permissions, prohibitions, and obligations associated with digital content.

Also, in light of the limitations that have emerged [2], the proposed approach aims to define sharing policies specifically for a given consumer. Always linked to data structure requirements, SHACL (Shapes Constraint Language) offers an approach for expressing data quality, compliance checks, and contractual restrictions [6]. With respect to these approaches, what is proposed in this article is directly linked to OpenAPI to offer a single point of description.

3 Overall approach

The proposed approach relies on the assumption that the data assets are managed as data products by the organizations, and these data products are exposed as REST services. Given the same data product, not all consumers could operate in the same way, and constraints related to accessible data, norms and rules compliance, and technical requirements can be specified. For example, let consider a hospital that offers a data product concerning patient information to (i) an internal hospital researchers and (ii) a government agency. We assume that both actors can only access anonymized patient records and they have limitations about where they can store the data once obtained: in Spain for the researchers

[1] `https://www.openpolicyagent.org`
[2] `https://www.w3.org/TR/odrl-model/`

and in EU for the government. In addition, the researchers are allowed to access only data related to patients over 60 years old and, due to capability constraints, there is a limited number of queries the researchers can submit in a given period.

Most of the time, these policies are formalized in documents by non-technical stakeholders, but the actual enforcement of these rules and the implementation of the agreements are handled separately by technical personnel.

To facilitate the definition of policies in a machine-readable format that can serve as a foundation for the automated configuration of enforcement mechanisms, we propose an extension of OpenAPI to specify the non-functional aspects, as agreed between data provider and data user. Being written in a machine-readable format, the document containing the agreement between the data provider and the data consumer for a given data product will be used to produce rules in Rego format. These rules will be used to configure an OPA PDP (Policy Decision Point) that is in charge of enforcing the defined policies.

4 OpenAPI extension

Figure 1 shows the metamodel of the proposed OpenAPI extension composed of: (i) a policy attachment mechanism to connect the policies to the different elements of an OpenAPI description, and (ii) a policy model adopted to specify the rules to be enforced when the data is accessed and transferred.

Policy Attachment. We distinguish three hierarchical levels of policies. *Global policies* apply system-wide, affecting every path and method of the API. *Resource-specific polices* are associated with a specific API path, and refine constraints for particular resources. Finally, *Operational policies* are defined per operation, allowing a finer-grained resource control. To this aim, we introduce the `x-policies` element[3], which can be attached at three levels: `Paths` level to indicate Global policies; specific `Path` for Resource-specific policies; and `Operation` for Operational policies. These three levels of granularity are combined using a logical conjunction (AND), a request is authorized only if all applicable policies are satisfied. Within each policy level, multiple policy definitions can coexist. Policies at the same level are combined using logical disjunction (OR), as alternatives: satisfying at least one of them satisfies the requirements for that level.

Policy Model. A policy is defined as a group of alternative policies composed of *clauses*. Clauses can be combined with an AND or OR *operator*, requiring all or at least one to be satisfied. At this stage, we focus on two main classes of policies, i.e., the *access policies* and *transformation policies*.

Access policy: Defines the constraints that must be satisfied when accessing a resource. It can be expressed using *Enumerated Values* (for categorical constraints) or *Interval Values* (for numerical constraints). The value is defined by three attributes: min, max, and unit. The clauses in an access policy are:

[3] We leverage OpenAPI's native support for custom fields via the *x-* prefix.

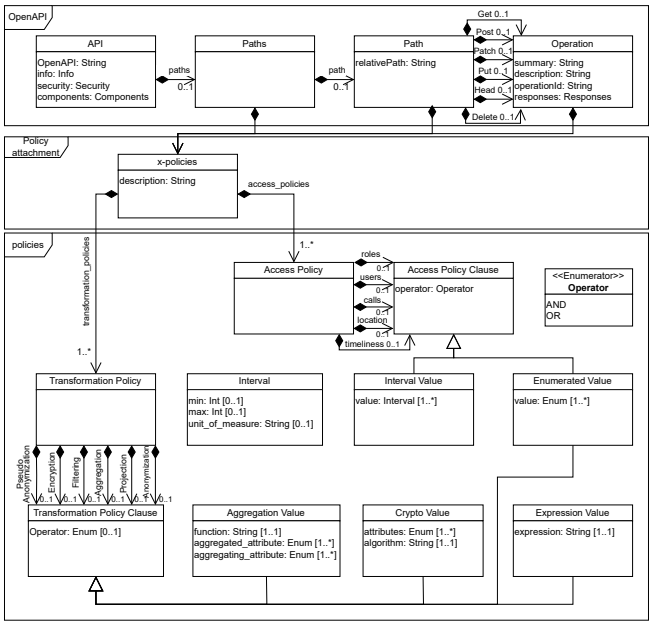

Fig. 1: Class Diagram of OpenAPI Extension

- Roles (categorical): the list of roles a user must belong to for access.
- Users (categorical): the list of users that have access granted.
- Calls (numerical): the number of times a resource can be called. The unit field represents the unit of measure (e.g., calls per month (cpm)), the min field represents the min number of calls that can be made in a given unit of measure, and the max field represents the max number of calls. Listing 1 shows the case where the max number of calls is 10 per day and 50 per week.
- Location (categorical): access is limited depending on the user's location. The value is taken from a set of admissible values (e.g., cities or countries).
- Timeliness (numerical): access is restricted based on the temporal distance between the creation of the data and the time of the call, by specifying the minimum distance (min), the maximum distance (max), or the interval value (represented by both min and max).

Transformation Policy: Defines the transformation that the data shared must be subject to. In addition to the categorical constraints, clause-specific formats are introduced to express them. The clauses defined for this type of policy are:

- Filter: the filtering condition (Listing 2) defined with a boolean *expression*.
- Projection: the list of data attributes to be retained in the output.
- Aggregation: for data aggregation, a policy defines a function, such as average, and an aggregated attribute that identifies which attribute to aggregate.

```yaml
calls:
  value:
    - max: 10
      unit: cpd
    - max: 50
      unit: cpw
  operator: AND
```

Listing 1: Call Clause

```yaml
filter:
  operator: AND
  policies:
    - expression: age <= 18 OR
                  age >=65
    - country: IT
```

Listing 2: Filter Clause

```yaml
aggregation:
  operation: AND/OR
  policies:
    - function: average
      aggregated_attribute: age
      aggregating_attributes:
        - country
        - pathology
```

Listing 3: Aggregation Clause

Additionally, an array of aggregating attributes indicates the attributes used to group the data (see Listing 3).
– Encryption/Anonymization/Pseudo-Anonymization: enforces encryption or (pseudo-)anonymization using a given algorithm, on selected attributes, and are associated with the `Crypto Value` class.

Referring to the example about patient data introduced in Section 3, Listing 4 and Listing 5 provide a formalization of the policies for, namely, the researchers and the governement which aim to consume the data product.

5 Rego rules generation

Given a REST interface described using the proposed OpenAPI extension, we describe here a method to generate enforceable policies written in the domain-specific language Rego, defined by the OPA framework.

Policy evaluation. To understand how the Rego rules are generated, it is important to understand how they are evaluated. Given req the user request coming from the user, this can be accepted or not if there is at least one $rule$ that is satisfied. Thus:

$$eval(req, rule_i) : [true, false] = \bigvee_i (match(req, rule_i))$$

If $eval$ is true, then the request can be processed; otherwise, it must be rejected. To be more specific, a request is defined as:

$$req = \langle method, path, \{param_i\} \rangle$$

Here, $method$ is the HTTP verb, $path$ is the resource, and $param_i$ includes additional metadata like query parameters, body, or middleware parameters (e.g., user info from IAM).

Rules are expressed in Disjunctive Normal Form (DNF) for efficient evaluation: any conjunction of literals corresponds to a set of clauses (a configuration) that must all hold at the same time [7]. With this form, the request can at most be compliant with a single policy; we can formalize a rule as:

$$rule = \langle method, path, \vee_i (conf_i) \rangle = \langle path, \vee_i (\wedge_j (clause_{i,j})) \rangle$$

```yaml
paths:
  x-policies:
    description: [...]
    access-policies:
      - role:
        value:
          - researchers
        - storage:
          value:
            - Spain
        - call:
          value:
            - max: 1000
            - unit: cpm
/AllPersonAsync:
  get:
    x-policies:
      description: [...]
      transformation-policies:
        - anonymization:
          policies:
            - algorithm: Hash
            - attributes:
              - "#/components/schemas
              /Person/Name"
              - "#/components/schemas
              /Person/Surname"
        - filter:
          policies:
            - expression: age > 60
/drug_exposures:
    ...
```

Listing 4: Extended OpenAPI for the researchers

```yaml
paths:
  x-policies:
    description: [...]
    access-policies:
      - role:
        value:
          - government
      - storage:
        value:
          - EU
/AllPersonAsync:
  get:
    x-policies:
      description: [...]
        - anonymization:
          policies:
            - algorithm: Hash
            - attributes:
              - "#/components/schemas
              /Person/Name"
              - "#/components/schemas
              /Person/Surname"
/drug_exposures:
    ...
```

Listing 5: Extended OpenAPI for the government agency

To adopt this evaluation model, rules in DNF must be derived from the extended OpenAPI description as discussed in the next.

Policy DNF generation. To derive the DNF of policies in OpenAPI, we first formalize policies attached to an element in OpenAPI(e.g., `x-policies`) as: $AP = \underset{r}{\vee}(\underset{s}{\wedge}(C_{r,s}))$, where r is the number of alternative in `x-policies` and s the number of clauses in the given alternative Here, r indexes policy alternatives, and s indexes clauses within each alternative. This format applies to access or transformation policies and is consistent at all OpenAPI levels: (i) $GP = \vee_{GP.r}(\underset{GP.s}{\wedge}(C^{GP}_{r,s}))$: the set of global policies attached to `paths` element defined as a set of alternative sets of clauses; (ii) element, $RP(p) = \underset{RP(p).r}{\vee}(\underset{RP(p).s}{\wedge}(C^{RP(p)}_{r,s}))$: the resource specific policies attached to a given `path` p; (iii) $OP(p,m) = \underset{OP(p,m).r}{\vee}(\underset{OP(p,m).s}{\wedge}(C^{OP(p,m)}_{r,s}))$: the policies attached to the method m of a path p.

Given a path p_x and one of its method $m_{x,y}$, the attached policies *pol* are:

$$pol(p_x, m_{x,y}) = \wedge(GP, RP(p_x), OP(m_{x,y}) =$$

$$= \wedge \left(\underset{GP.r}{\vee} \left(\underset{GP.s}{\wedge} C^{GP}_{r,s} \right), \underset{RP(p).r}{\vee} \left(\underset{RP(p).s}{\wedge} C^{RP(p)}_{r,s} \right), \underset{OP(p,m).r}{\vee} \left(\underset{OP(p,m).s}{\wedge} C^{OP(p,m)}_{r,s} \right) \right)$$

Exploiting the distributive law, we can obtain the DNF version:

$$\underset{GP.r}{\vee} \left(\underset{GP.s}{\wedge} C^{GP}_{r,s} \right) \wedge \underset{RP(p).r}{\vee} \left(\underset{RP(p).s}{\wedge} C^{RP(p)}_{r,s} \right) \wedge \underset{OP(p,m).r}{\vee} \left(\underset{OP(p,m).s}{\wedge} C^{OP(p,m)}_{r,s} \right) =$$

$$= \underset{GP.r,GP.s,RP(p).r,RP(p).s,OP(p,m).r,OP(p,m).s}{\vee} \left(\wedge C^{GP}_{r,s}, \wedge C^{RP(p)}_{r,s} \wedge C^{OP(p,m)}_{r,s} \right)$$

Rego rules generation Building upon the DNF formalization, the approach to generate the corresponding Rego rules operates as follows:

1. *Clause Translation:* Each policy clause is translated into a Rego boolean expression, encoding its predicate logic and referencing relevant request data (e.g., user info, parameters, or contextual data).
2. *Handling Logical Operators:* The DNF structure ensures that logical ORs (policy alternatives) result in separate rule bodies, while logical ANDs (conjunctions of conditions) are combined within a single rule body.
3. *Hierarchical Specialization:* Clause expressions from each policy level (global, resource, operational) are recursively combined. Authorization is granted only if all relevant clauses across applicable hierarchies are satisfied. More specific policies (e.g., for a particular path) refine or override broader ones.
4. *Rule Generation:* Each combination of clause alternatives forms a distinct `allow` rule body. A request is authorized if at least one `allow` rule evaluates to true, mirroring the logical OR structure of DNF within Rego.

Overall, this DNF-based rule generation robustly enforces complex, hierarchical authorization requirements, while maintaining both clarity and modularity in Rego code. Importantly, it decouples policy content from logical structure, supporting future extensions and maintainability.

6 Validation

The validation of our approach includes two types of activities concerning (i) the ability of the policy model to express constraints in different settings, and (ii) the ability to generate the Rego rules starting from OpenAPI descriptions.

About the first point, we conducted a validation across six real-life scenarios covering different domains. This case study refers to different domains: healthcare (whose details are already provided in Section 4), manufacturing, mobility, smart cities, and agriculture. These are OpenAPIs related to real services that companies have provided as the interfaces used to expose data for sharing purposes. In this case, we have empirically proven that the provided extension was able to address all the requirements of the domain experts.

The second validation activity has produced a tool[4] that ingests an extended OpenAPI specification and parses its hierarchical policy structure, which

[4] `https://github.com/HE-TEADAL-PROJECT/opa-policy-manager`

comprises global, intermediate (path-specific), and operational (method-specific) policies. This structure is populated by a parsing module, which extracts configuration parameters and organizes them according to the hierarchical model. The core generator module then transforms this representation into a set of Rego rules. Each policy type is mapped to its corresponding Rego predicate, and rules are composed according to the logical structure described in Section 5, ensuring correct enforcement of policy alternatives and conjunctions across all levels.

7 Conclusion

This has proposed an OpenAPI extension to explicitly include policy constructs governing data product access. This extension was used to develop a pipeline that automatically generates a set of Rego-compliant rules to populate the PDP. The approach, supported by a tool, is validated through a series of tests involving multiple domains.

Future work will focus on extending the clauses to further aspects, such as sustainability. Moreover, LLM-based mechanisms for the generation of extended OpenAPIs will be studied, starting from a description in natural language.

References

1. Abraham, R., Schneider, J., vom Brocke, J.: Data governance: A conceptual framework, structured review, and research agenda. Int'l J. of Information Management **49**, 424–438 (2019). https://doi.org/10.1016/j.ijinfomgt.2019.07.008
2. Cimmino, A., Cano-Benito, J., García-Castro, R.: Practical challenges of odrl and potential courses of action. In: Companion Proc. of the ACM Web Conference 2023. p. 1428–1431 (2023). https://doi.org/10.1145/3543873.3587628
3. Data Spaces Support Centre (DSSC): Core concepts. `https://dssc.eu/space/Glossary/176554052/2.+Core+Concepts`
4. European Commission: Data Governance Act. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32022R0868` (2022)
5. Jussen, I., Schweihoff, J., Dahms, V., Möller, F., Otto, B.: Data sharing fundamentals: characteristics and definition. In: Proc. of the HICSS 2023. Maui, Hawaii, USA (2023). https://doi.org/10.24251/HICSS.2023.452
6. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL) - W3C Recommendation, `https://www.w3.org/TR/shacl/`
7. Mukhi, N.K., Plebani, P.: Supporting policy-driven behaviors in web services: experiences and issues. In: Proc. of the 2nd ICSOC. p. 322–328. ICSOC '04, ACM, New York, NY, USA (2004). https://doi.org/10.1145/1035167.1035214
8. Wider, A., Jarmul, K., Akhtar, A.: Towards automating federated data governance. In: 2024 IEEE ICWS. pp. 10–19 (2024). https://doi.org/10.1109/ICWS62655.2024.00019
9. Wider, A., Verma, S., Akhtar, A.: Decentralized data governance as part of a data mesh platform: Concepts and approaches. In: 2023 IEEE ICWS. pp. 746–754 (2023). https://doi.org/10.1109/ICWS60048.2023.00101
10. Yang, J., Tang, Y., Beheshti, A.: Design Methodology for Service-Based Data Product Sharing and Trading, pp. 221–235. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-73203-5_17