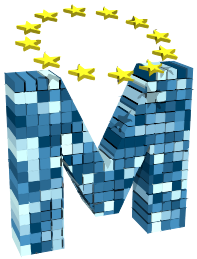

Project title: Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
Project acronym: MOSAICrOWN
Funding scheme: H2020-ICT-2018-2
Topic: ICT-13-2018-2019
Project duration: January 2019 – December 2021

D3.5

Final Report on the Data Governance Framework, with Language and Model

Editors: Sabrina De Capitani di Vimercati (UNIMI)
Reviewers: Flora Giusto (MC)
Benjamin Weggenmann (SAP SE)

Abstract

The main goal of Work Package 3 (WP3) is the definition of a data governance framework for managing data and for specifying policies in the data market scenario. This deliverable presents the policy model and language developed in WP3. We first recall the principles and desiderata at the basis of our work and describe the concepts on which the proposed model is based, that is, subject, object, operation, purpose, and condition. We then show the access and ingestion policies and how they can support the use cases of the project by enforcing transformations that protect data for their ingestion in the data market.

Type	Identifier	Dissemination	Date
Deliverable	D3.5	Public	2021.10.31

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333.

MOSAICrOWN Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	EMC Information Systems International	EISI	Ireland
3.	Mastercard Europe	MC	Belgium
4.	SAP SE	SAP SE	Germany
5.	Università degli Studi di Bergamo	UNIBG	Italy
6.	GEIE ERCIM (Host of the W3C)	W3C	France

Versions

Version	Date	Description
0.1	2021.10.05	Initial Release
0.2	2021.10.26	Second Release
1.0	2021.10.31	Final Release

List of Contributors

This document contains contributions from different MOSAICrOWN partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	Sabrina De Capitani di Vimercati (UNIMI)
Chapter 1: Introduction	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 2: Policy model	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 3: Access policy	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 4: Ingestion policy	Marco Abbadini (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Dario Facchinetti (UNIBG), Sara Foresti (UNIMI), Gianluca Oldani (UNIBG), Stefano Paraboschi (UNIBG), Matthew Rossi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 5: Use case support	Marco Abbadini (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Dario Facchinetti (UNIBG), Sara Foresti (UNIMI), Gianluca Oldani (UNIBG), Stefano Paraboschi (UNIBG), Matthew Rossi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 6: Conclusions	Sabrina De Capitani di Vimercati (UNIMI)

Contents

List of Figures

List of Tables

Executive Summary

The overall goal of the MOSAICrOWN project is to enable data sharing and collaborative analytics in multi-owner scenarios in a privacy-preserving way, ensuring proper protection of private/sensitive/confidential information. A fundamental aspect that has to be considered for achieving this goal is enabling data owners to maintain control on their data, operating independently and autonomously in the specification of regulations over their data. This is what Work Package 3 aims to realize through the design of a data governance framework that includes a data protection specification model and language. Such a model and language allow data owners to specify policies holding over their data, both in terms of access restrictions, privacy protection, and usage control. This document presents the extended version of the model and language that have been first presented in deliverable D3.3 "First version of policy specification language and model" (M18) [DS20]. In particular, this document focuses on access policies that define policy rules regulating access to datasets in the data market, and on ingestion policies that define how datasets have to be transformed before storing them in the data market. The reference scenario considered in this document can be briefly summarized as follows. The *data market provider* (DMP) manages access to data that entities, called *owners*, make available to it for external release. Such data are usually accessed by other parties, called *consumers*. The data market provider collects, processes, and makes data ready for the consumers. Datasets available in the data market can be in the original form, as they have been given by the owners, can be stored in a protected form (obtained with the application of wrapping or sanitization techniques [Par20, DG21, Böh21]), or can be obtained through the combination of data of different owners, thus obtaining aggregates. Datasets are not just released unconditionally to anybody. Rather, certain data may be released only to specific requesters or under specific conditions. As an example, there are data that can be released only to research or academic institutions, there are data whose release must happen only when the access request comes from a specific location, and so on. Work Package 3 has designed a policy model and language for the definition of such security requirements and developed a policy engine enforcing such security requirements (Deliverable D3.4 "Final tools for the governance framework" [De 21]). The policy engine mediates every request submitted to the data market provider to determine whether, and possibly at what conditions, the request can be granted.

The remainder of this deliverable is organized as follows.

- Chapter 2 first recalls the basic principles and desiderata that have guided the work on policies, and reports a glossary of terms. Then, the chapter continues with a description of the basic concepts of the proposed model, that is, subject, object, catalog, purpose, operation, and condition. In particular, the focus is on the organization at the data market provider of the information that can be used in the policy specification and that is also managed by the policy engine.
- Chapter 3 describes the access policies that express security requirements. The chapter briefly recalls how an access request is modeled and then illustrates the policy rules sup-

ported followed by the basic constructs of the language for regulating access to data in the digital data market. The language builds upon the policy model defined in the previous chapter, supporting specification of policy rules in a simple, yet flexible and expressive way. The language provides support for conditions on subject profiles, metadata, and contextual information as well as explicit consideration of purpose of access.

- Chapter 4 describes the ingestion policies that regulate how data are transformed before their storage in the data market. The chapter also illustrates how such policies can be expressed in ODRL, a W3C policy specification language that has been also used for expressing the access policies.
- Chapter 5 illustrates how the ingestion policies described in the previous chapter can support the ingestion requirements of the MOSAICrOWN use cases.
- Chapter 6 concludes the deliverable.

1. Introduction

The goal of MOSAICrOWN is to enable data sharing and collaborative analytics in multi-owner scenarios in a privacy-preserving way, providing owners with control on the information shared and released to others. Concretely, MOSAICrOWN empowers owners with the ability to specify and to enforce in an easy and flexible way restrictions that should hold on the sharing and usage of their data. The data governance framework developed in Work Package 3 (WP3) represents the actual means allowing data owners to maintain control on their data, including regulating the controlled application, and reasoning upon the data protection techniques designed in Work Package 4 (WP4) and Work Package 5 (WP5). At the core of the governance framework is the policy model and language, presented in this deliverable, dictating the policy specifications regulating data access, usage, processing, and release. The policy model and language provide then the glue, nicely bringing together and leveraging the availability of technical solutions in the project. The MOSAICrOWN model has been designed to be extremely flexible and the adoption of Semantic Web technology for the representation of the policy [De 21] contributes significantly to the ability of a single language to represent a variety of policies.

1.1 State of the art

Data sharing and dissemination are basic features for data markets that should be supported in a controlled way so that data owners remain in control of their data. This is also in line with recent laws and regulations (e.g., the EU GDPR) that empower the subject of a data item (i.e., the individual to whom the data item refers) with rights over it. The consideration of these problems in the data market scenario introduces the need for supporting expressive and flexible policies, which impose restrictions on the use and processing of data, and efficient and effective enforcement mechanisms [DFLS21]. Such a problem is largely recognized and the growing interest in the data market platform clearly strengthens such need, calling for solutions that can regulate the use, processing, and dissemination of (potentially sensitive) data, enhancing the control of data owners on their data. Recognizing the importance and interest of these techniques, the research and industrial communities have investigated solutions for empowering users with control over their data in different sharing scenarios. Among the different proposals addressing the problem of defining policy languages and models, there are approaches aiming at: modeling regulations and supporting compliance verification for the GDPR (e.g., [ABD19, PG18]), applying the FAIR principles in the context of access control (e.g., [BNRV20]), supporting privacy of users in digital interactions (e.g., [ACK$^+$10, ADF$^+$12, BS02]), leveraging encryption for enforcing access control (e.g., [BDF$^+$18, DFJ$^+$10, ZDX$^+$20]), accounting for non fully trusted storage providers (e.g., [ZDX$^+$20]), and enriching authorization specifications (e.g., [BK19, BS03, JSSS01]), and supporting privacy-enhanced data flows in IoT processing systems (e.g., [GPW$^+$20]). Among related works, also work carried out in the context of European projects, such as PrimeLife

(primelife.ercim.eu) and SPECIAL (www.specialprivacy.eu), specifically targeting policies and privacy. Such projects considered, however, a different focus. In particular, Primelife was concerned with privacy of users accessing the system, in contrast to privacy of data accessed and processed as MOSAICrOWN. SPECIAL instead focused on the support of GDPR specifications, hence considering a limited set of requirements that may need to be considered in real-life digital data markets, which MOSAICrOWN addresses. Therefore, such works are complementary in focus and goal with respect to MOSAICrOWN.

One of the main challenges in the design of a policy model and language is to balance simplicity and easy of use (to make it appealing and acceptable for end users as well as to ensure its effective and efficient enforcement) with its expressiveness and flexibility (to ensure its suitability for capturing different requirements). For instance, logic-based languages, while appreciable for their elegance and expressiveness, may turn out to be unsuitable in practical settings for their complexity in use and enforcement. In MOSAICrOWN, we address such a challenge, providing for a simple and easy to use, yet flexible and expressive, language supporting abstractions, conditional expressions on data and subjects, and explicit consideration of purpose of access and of data transformation (which trigger the application of data wrapping and sanitization).

1.2 MOSAICrOWN innovation

The work on the policy language produced several advancements over the state of the art.

- *Requirement support*. The language supports different requirements that data owners might wish to specify, and have enforced, on data ingested, stored, or processed in the data market.
- *Transformation*. The language supports the specification of data transformations, including wrapping and sanitization, to provide protection of data with direct control in the language.
- *Granularity*. Policy rules can be specified at different granularity levels, with consideration of abstractions.
- *Metadata support*. The language supports data access and usage conditions depending on metadata associated with data and subjects.
- *Deployment of existing technology*. The policy engine enforcing the policies uses standard solutions for providing effective deployment and interoperability with existing solutions.

In the remainder of this deliverable, we describe the MOSAICrOWN policy model and language that is used to represent the access policies that must be applied when access requests are made to the data stored in the data market (Chapters 2-3). This control can be executed on multiple data sources, with distinct models (relational data sources and RDF repositories) and is the responsibility of the policy engine described in deliverable D3.4 [De 21]. We then show how the MOSAICrOWN policy model and language can also be used to represent the ingestion policies, that is, how the data can be imported into the data market (Chapters 4-5).

2. Policy model

This chapter provides a description of the basic concepts of the policy model. We first present the principles and desiderata followed in the design of the policy model and language, and then examine the different elements of the model that characterize our proposal (i.e., subject, object, operation, purpose, and condition).

2.1 Basic principles and desiderata

Before illustrating the policy model, we briefly recall the basic principles and desiderata that the language should satisfy and that guided our work.

- *Flexible management of heterogeneous datasets.* The data market should manage data of different kinds, ranging from structured to unstructured data, and accessible at different granularity levels.

  Datasets in the data market can correspond to datasets in their original format, datasets protected through the application of wrapping and/or sanitization techniques, and datasets corresponding to the results of analytics.

- *Fine-grained protection.* Data protection techniques can be applied on datasets at different levels of granularity. The protection techniques can be applied during the different phases of the data life-cycle, ranging from ingestion to analytics.

- *Fine-grained access control.* Access control should support different granularity levels with respect to datasets and subjects.

- *Configurable protection.* The owners of the datasets should be able to specify how their datasets should be protected through wrapping or sanitization techniques. The owner can specify the kind of techniques as well as the corresponding privacy parameters regulating their working.

- *Purpose-based access control.* Access control should support access and usage restrictions based on purpose.

- *Abstractions.* The policy model and language should support the specification of access restrictions based on typical abstractions (e.g., groups) defined over the domain of users.

  Data owners moving their datasets to the data market should be able to specify whether their datasets can be shared with a particular consumer or set of consumers.

- *Metadata support.* The policy model and language should support the specification of access restrictions based on conditions on metadata describing (meta)properties of the datasets such as the level of sensitivity of (portions of) datasets, and the kind of datasets.

In addition to these principles, our work has considered further desiderata.

- *Human- and machine-understandable language.* The protection requirements should be specified using a format both human- and machine-understandable that should be simple and expressive. It should be also easy to verify the compliance with respect to defined protection requirements.
- *Extensible model and language.* The model and language should be easily extensible for considering new domains, vocabularies, and new requirements.
- *Scalable and efficient implementation.* The model and language should be practically usable in real-word scenarios and applications where the scalability and efficiency in policy specification and enforcement is fundamental.
- *Practical applicability and compatibility with existing technologies.* The model and language should enjoy practical applicability as well as compatibility and integrability with the current approaches and technologies used in the interaction with data market providers, thus ensuring direct deployment.

To avoid ambiguity in the remainder of this document, Table 2.1 reports a simple glossary of terms that are assumed known in this document.

2.2 Subjects

The data market provider recognizes only subjects registered at the data market. Each subject s is assigned an identifier, denoted $id(s)$, that allows the data market provider to refer to the subject. Besides their identifiers, subjects registered at the data market provider usually have other properties associated with them (e.g., an owner may have properties such as name, address, and occupation). To capture and reason about these properties, we assume each subject is associated with a *profile*. Intuitively, profiles describe properties of subjects. To be as general as possible, we view profiles as semi-structured documents (e.g., profiles can be implemented through XML or RDF like documents). The profile associated with a subject defines the name and value of some properties, also called *attributes*, that characterize the subject. Figure 2.1 illustrates an example of profile for two subjects, `Anna` and `Billy`. At the abstract level, we use the classical dot notation to refer to a specific property, that is, $id(s).attribute\_name$ denotes the value of attribute *attribute_name* in the profile of subject s. For instance, `Anna.email` denotes the `email` address stored in the profile of `Anna`.

2.2.1 Subject groups

Abstractions can be defined within the domains of subjects. Intuitively, abstractions allow to group together subjects with common characteristics and to refer to the whole group with a name. Groups can be nested (i.e., groups can be defined as members of other groups) and need not be disjoint (i.e., a subject can belong to more that one group). At a very high level, groups can distinguish the different categories of subjects that need to interact with the data market provider. Groups define a partial order that can be depicted as an acyclic graph whose nodes are the groups, and an arc between node n_1 and node n_2 indicates a *direct* (i.e., explicitly defined) membership of n_1 in n_2, going from bottom to top. Figure 2.2 illustrates an example of subject hierarchy, where groups

Term	Definition
Attribute	Property that characterizes an object or subject
Consumer	Organization interested in accessing data offered through the data market platform
Data category hierarchy	Categories of datasets with a containment relationship forming a hierarchy rooted at `Any`
Data market catalog	A catalog that provides a description of all objects available in the data market
Data market provider (DMP)	Organization that receives data from data owners and makes them available to other parties accessing the data market platform
Data owner	An entity that produces objects that are made available through the data market platform
Dataset	Data stored on the data market platform access to which must be controlled
Materialized object	An object physically stored in the data market that can be in plaintext or in protected form
Metadata	Information associated with a dataset or an attribute of a dataset, describing a property of the dataset, attribute, or dataset's content
Object	Dataset or metadata managed by a data market provider
Operation	An action that a subject can perform over the objects in the data market
Operation hierarchy	Sets of operations with a containment relationship forming a hierarchy rooted at `Any`
Purpose	Reason for which an object can (or will be) used
Purpose hierarchy	Abstractions defined over the set of purposes forming a hierarchy rooted at `Any`
Sanitization technique	Non-reversible protection technique producing a transformed version of an object
Subject	Data owner or consumer of the data market
Subject hierarchy	Groups of users with a containment relationship forming a hierarchy rooted at `Any`
Transformation	Wrapping technique or sanitization technique used for protecting datasets
Virtual object	An object listed in the data market catalog that has to be computed on-the-fly
Wrapping technique	Reversible protection technique producing a transformed version of an object

Table 2.1: Glossary of terms

distinguish different categories of subjects. We assume each hierarchy to be rooted, meaning there is one element to which all elements in the hierarchy belong. This assumption is not limiting (a dummy group to which all elements in a domain belong can be assumed) and is common in

Anna

Attribute	Value
ID	78934
name	Anna
dob	1990-07-22
group	Marketing
address	Ficus street, Auckland
region	Auckland
citizenship	NZ
email	anna@mybank.nz
telephone	0273874629
registrationDate	2021-07-01

Billy

Attribute	Value
ID	78234
name	Billy
dob	2000-03-10
group	HumanResource
address	Picea street, Milan
region	Lombardy
citizenship	Italian
email	billy@mytree.com
telephone	0373567914
registrationDate	2020-06-23

Figure 2.1: An example of two subject profiles

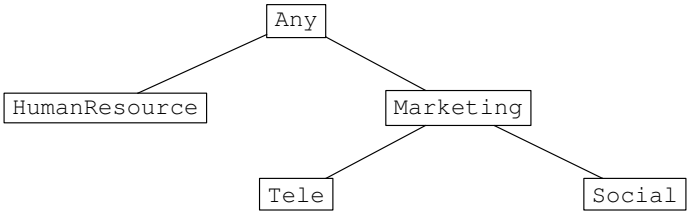

Figure 2.2: An example of subject hierarchy

many systems, where, for example, a group Public is usually considered to which all the subjects belong. In the remainder of this document, we assume group `Any` to be the root of the subject hierarchy. The subject hierarchy in Figure 2.2 is then characterized by the root `Any` that includes groups `HumanResource` and `Marketing`. Group `Marketing` is further specialized in `Tele` and `Social`.

2.3 Objects

Objects are the entities to which accesses can be requested. The data market provider distinguishes two kinds of objects: *datasets* and *metadata*. In the following, we describe datasets and metadata in more detail, and briefly recall the concept of transformations, for creating a protected version of datasets, and the concept of catalog, for keeping track of the datasets managed by a data market provider.

2.3.1 Datasets

Datasets contain information to which access is being regulated. Our model supports both access to a whole dataset (i.e., an access to a dataset is either allowed or denied) as well as access to a finer granularity. We consider *structured* datasets that are characterized by a unique identifier and a set of *attributes* modeling properties of the datasets. Like for subjects, given a dataset d, $id(d)$ denotes the unique identifier of dataset d, and $id(d).attribute\_name$ denotes the value of attribute *attribute_name* of dataset d. Note that datasets can be stored in their original format (source dataset) or can be stored in a protected form, meaning that the data are transformed by the owner (or by the data market provider) before moving them to the data market platform. For instance,

CardHolder

cid	name	surname	dateofbirth	email	phone	creditscore
27822	Alice	Rossi	1990-01-05	alice@example.org	+39022534816	650
36538	Bob	Smith	1955-02-27	bob@example.org	+14156551825	800
56141	Carol	Brown	200-08-23	cb@example.org	+64027369595	720
...	...	...	...	...	...	...

InsurancePlan

id	name	surname	dob	gender	country	type	coverage
27822	Alice	Rossi	1990-01-05	female	NZ	basic	life
91885	Dave	Moore	1942-12-25	male	AU	advanced	health
59154	Eva	Clark	1978-05-05	female	NZ	basic	vehicle
...	...	...	...	...	...	...	...

(a) datasets: CardHolder and InsurancePlan

META (CardHolder)
- **category** : Financial
- **purpose** : Commercial
- **creator** : Til
- **retention** : ten years

META (InsurancePlan)
- **category** : Financial
- **level** : 1
- **creator** : BeSafe
- **retention** : ten years

(b) metadata

Figure 2.3: An example of datasets (a) and corresponding metadata (b)

Figure 2.3(a) illustrates two datasets, CardHolder and InsurancePlan storing information about the holders of a credit card and the insurance plans subscribed by users. The information stored in these datasets is organized as a set of records having a fixed set of attributes. In particular, dataset CardHolder has attributes cid, name, surname, dateofbirth, email, phone, and creditscore modeling the identifier, name, surname, data of birth, email address, telephone number, and credit score, respectively of a cardholder. Dataset InsurancePlan has attributes id, name, surname, dob, gender, country, type, coverage modeling the identifier, name, surname, date of birth, gender, and country of a policyholder, respectively, together with the type of insurance and of coverage.

Datasets can also be organized in a hierarchical structure, defining sets of datasets, called *categories*, that can be collectively referred together with a given name. A category corresponds to a set of datasets referring to the same context. Like for subjects, categories can be nested and do not need to be disjoint. The definition of categories of datasets introduces a hierarchy over them, called *data category hierarchy*. Figure 2.4 illustrates an example of such a hierarchy, according to which there are two main categories of datasets, Public and Company. The Company category is further specialized in Financial and Personnel. Again, we assume the hierarchy to be rooted. In the remainder of this document, Any will be assumed to be the root of the data category hierarchy.

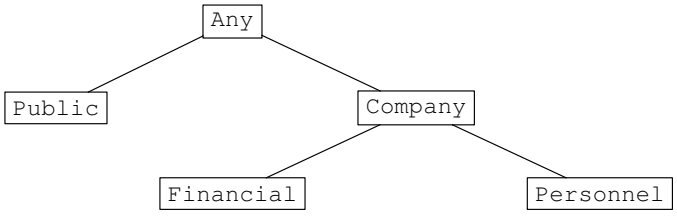

Figure 2.4: An example of data category hierarchy

2.3.2 Metadata

Metadata represent data about data. They can be in the form of textual or semistructured documents. At a practical level, we distinguish between two different kinds of metadata:

- metadata associated with whole datasets;
- metadata associated with single properties of structured datasets.

We assume that a bijective function META() makes the association between a dataset and its metadata. For instance, given dataset `CardHolder`, function META(`CardHolder`) refers to the metadata on the left-hand side of Figure 2.3(b) that include information about the category of the dataset, the purpose for which the dataset can be used, the creator of the dataset, and the retention policy applied to the content of the dataset. The same function META() can also be used for associating a property of a dataset with its metadata. For instance, META(`CardHolder.name`) refers to the metadata associated with attribute `name` of dataset `CardHolder`.

For metadata browsing as well as for the evaluation of conditions that may determine whether a given access to datasets can be allowed, it is useful to evaluate the content of metadata. The model then supports fine-grained access to metadata documents at the level of a single property. Properties within a metadata document are referred by means of the classical dot notation. For instance, notation META(`CardHolder`).`category` refers to the category of dataset `CardHolder`. Analogously, META(`CardHolder.name`).`type` refers to the metadata `type` associated with attribute `name` of dataset `CardHolder`.

2.3.3 Transformations and catalog

MOSAICrOWN distinguishes between two main classes of techniques for protecting objects: *sanitization* and *wrapping*. The application of such techniques can be controlled at the policy level, with the policy language supporting authorizations restricting subjects to access protected versions of the objects (in contrast to the original objects) [DS20]. In particular, wrapping and sanitization techniques can be applied *offline* or *on-the-fly*. Offline means that a protected version of an object is produced and stored (materialized) in the data market and such a version is the one available for access. On-the-fly means that a protected version of an object is produced at the time of access (virtual). Since the data market may have different (materialized or virtual) protected versions of the same object, to avoid ambiguity during the evaluation of an access request, a consumer must always specify the unique identifier associated with the object of interest. To facilitate the discovery of such an object, we assume that all objects in the data market can be searched via a *data market catalog*. Such a catalog stores information about the materialized objects (i.e., the objects physically stored in the data market) as well as information about *virtual objects* that can be

Figure 2.5: An example of catalog showing two source datasets and their protected versions

obtained from the on-the-fly application of a transformation (sanitization or wrapping technique) on materialized objects. Note that only the objects explicitly listed in the catalog are the ones available for access. The catalog can support the discovery of objects through the specification of metadata properties that characterize the objects. For instance, a consumer could search all objects that have been created by Til, where `creator` is a metadata property associated with the objects in the data market. The detailed description of the catalog schema, of the discovery function, and of how the information shown in the catalog should be eventually obfuscated to avoid the leakage of possible sensitive information, is outside the scope of this document. For our purpose, we assume that the catalog provides a list of objects together with their unique identifier, schema, and additional information. Figure 2.5 shows an example of the possible content of the data market catalog. In this example, for simplicity of exposition, the identifier of the objects corresponds to their name. Furthermore, for each object the catalog shows some metadata associated with it. Figure 2.5 illustrates four datasets: `CardHolder` and `InsurancePlan` are two source datasets, and `VirtualCardHolder` and `MaterializedInsurancePlan` are the protected versions of these two source datasets. In particular, the `VirtualCardHolder` dataset is a dataset for which there is only a description in the catalog of how it can be obtained from the `CardHolder` dataset. Such dataset can be obtained by encrypting on-the-fly attributes `name`, `surname`, and `dateofbirth`. Dataset `MaterializedInsurancePlan` is a materialized version of the

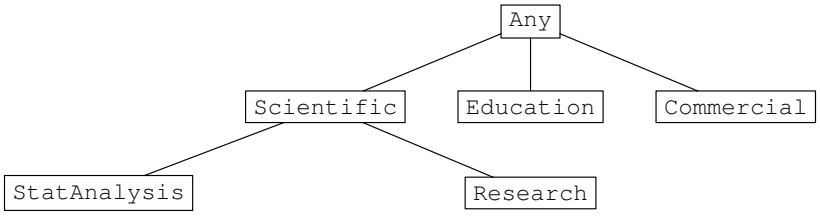

Figure 2.6: An example of purpose hierarchy

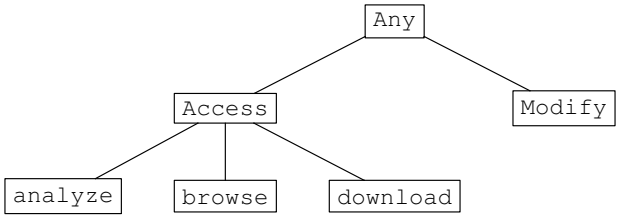

Figure 2.7: An example of operation hierarchy

`InsurancePlan` dataset that is obtained through the application of k-anonymity, with $k = 5$, on attributes `dob` and `gender`. The resulting dataset includes only attributes `dob`, `gender`, `type`, and `coverage`.

2.4 Purpose

Purpose is an important concept that is mentioned in recent regulations (e.g., the General Data Protection Regulation). It represents the purpose of the processing (e.g., subjects can have the need of restricting the access to their data for research purposes only). Our model captures this concept and supports the definition of abstractions over it. Different purposes can be grouped together and can be represented by a more general purpose (e.g., `StatAnalysis` and `Research` can be represented by the `Scientific` purpose). This is equivalent to say that purposes can be organized according to a hierarchy, which can be depicted as an acyclic graph with a root element. Figure 2.6 illustrates an example of purpose hierarchy rooted at `Any`. In this example, `StatAnalysis` and `Research` are a specialization of `Scientific`, and `Scientific`, `Education`, and `Commercial` are a specialization of purpose `Any`.

2.5 Operations

Operations to be considered may vary depending on the specific dataset. Generally, the kinds of operations supported should include: `download` (a subject can download an object and can perform off-line analysis), `analyze` (a subject can invoke a set of pre-defined operations that perform calculations on selected objects), `browse` (a subject can browse a dataset without downloading it). Abstractions can also be defined on actions, specializing actions or grouping them in sets. Figure 2.7 illustrates an example of purpose hierarchy rooted at `Any`, where the three kinds of actions mentioned above are grouped in a set called `Access` and thus referred to as one.

2.6 Conditions

Our model and language supports the definition of conditions associated with policy rules. Such conditions must be satisfied to consider the policy rules with which they are associated. In particular, we consider two kinds of conditions:

- conditions on subjects and objects;
- conditions on contextual information such a location and time of an access request.

The evaluation of conditions of subject profiles, metadata as well as conditions on context implies the existence and management of subject's profiles and contextual information that the data market provider can access. In particular, as also discussed in deliverable D3.4 [De 21], the evaluation of context information (e.g., time or location of an access request) and subject profile cannot be directly performed by the policy engine, which is the component in charge of checking whether an access request is compliant with the existing policies. The policy engine must interact with other components. The check process then returns either a true or a false value depending on whether the conditions are satisfied. The specification of context conditions is modeled through the definition of specific predicates. For instance, ORIGIN(www.mosaicrown.eu) is a predicate that is evaluated to true whenever the access request originates from domain www.mosaicrown.eu. Analogously, predicate WORKINGHOURS() evaluates to true whenever the access request is generated during the "working hours". In this case, the definition of the working hours is local to the data market platform. The policy engine can instead directly evaluate conditions on the content of objects [De 21].

3. Access policy

This chapter illustrates the access policy language. Access policies define policy rules concerning access to objects in the data market. Policy rules need to be checked when a subject submits an access request. When a subject request is submitted to the data market platform, the data market first evaluates such request against the access policy rules applicable to it. In this chapter, we first recall the format of an access request (Section 3.1), show the different components of the policy rules (Section 3.2), and illustrate how policy rules are enforced (Section 3.3).

3.1 Access request

The data market provider receives requests from subjects for processing objects. A subject request is a quadruple of the form:

$$\langle \textit{subject}, \textit{object}, \textit{operation}, \textit{purpose} \rangle$$

- *subject* is the subject that makes the request;
- *object* is the object on which *subject* wants to perform *operation*;
- *operation* is the operation that is being requested;
- *purpose* represents the reason for which object *object* is being requested.

Subject. A *subject* can be either the identifier of an entity registered with the data market platform or can be an anonymous entity. In the first case, the *subject* component is an identifier from which it is possible to retrieve the corresponding subject profile stored by the data market provider. In the second case, the *subject* component corresponds to the reserved keyword `anonymous`.

Object. The *object* component of a request can be the identifier of a dataset or a *view* over a (structured) dataset. A view over a dataset d characterized by a set $a_1, \ldots, a_m$ of attributes is a subset of the set $a_1, \ldots, a_m$. For instance, with respect to dataset `CardHolder` shown in Figure 2.3(a), `CardHolder.{cid, name, surname}` is a view over `CardHolder`.

Operation. The *operation* component of a request corresponds to the operation that the *subject* requires to perform over *object*. Specific operations may vary depending on functionality provided on specific kinds of datasets. In the following, we assume that an operation accesses or manages an object as it is. Examples of this class of operations are `download`, `read`, and `delete`.

request	::=	⟨subject, object, operation, purpose⟩
subject	::=	subject-id | `anonymous`
object	::=	object-id[.{list}]
operation	::=	identifier
purpose	::=	identifier
subject-id	::=	identifier
object-id	::=	identifier
list	::=	identifier | list [, identifier]
identifier	::=	letter | identifier [character]
character	::=	letter | digit

Figure 3.1: BNF syntax of access requests

Purpose. The *purpose* component of a request is the reason for which objects are being requested and will be used. The purpose allows us to model the fact that the decision of whether some objects may or may not be released may also depend on the use the subject intends to do with the object being requested, possibly declared by the subject at the time of the request. The purpose component is an element of the purpose hierarchy, meaning that it may correspond to a ground purpose or to an abstraction.

The following are examples of subject requests.

- ⟨`anonymous`, `read`, `InsurancePlan`, `StatAnalysis`⟩: an anonymous subject requires to `read` dataset `InsurancePlan` for `StatAnalysis` purposes. The dataset includes information about the insurance plans of users.
- ⟨`Anna`, `read`, `CardHolder.{name, surname}`, `Commercial`⟩: `Anna` requires to `read` attributes {`name`, `surname`} of the `CardHolder` dataset for `Commercial` purposes. The dataset includes information about the cardholders.

Figure 3.1 illustrates the BNF syntax of the access request.

3.2 Policy rules

We describe our policy language that is based on the concepts discussed in the previous chapter. Declarative access control specifications are generally based on rules stating which subject can exercise which operation on which object. Our proposal supports policy rules having the form:

⟨*subject*, *object*, *operation*, *purpose*, *condition*, *sign*⟩

- *subject* identifies the set of subjects to which the policy rule refers;
- *object* identifies the set of objects to which the policy rule refers;
- *operation* is the operation (or a class of operations) to which the policy rule refers;
- *purpose* is the purpose (or a purpose abstraction) to which the policy rule refers;

- *condition* is a boolean expression of conditions that every access request to which the policy rule applies must satisfy;
- *sign* is the sign of the policy rule ($+$ or $-$).

Our policy rules include a sign ($+$ or $-$) that indicates if the rule specifies a permission or a denial. Informally, a policy rule $\langle$*subject*, *object*, *operation*, *purpose*, *condition*, $+\rangle$ states which *subject* can perform which *operation* on which *object* under which *condition*, and for which *purpose*. Analogously, a policy rule $\langle$*subject*, *object*, *operation*, *purpose*, *condition*, $-\rangle$ states which *subject* cannot perform which *operation* on which *object* under which *condition*, and for which *purpose*. The reasons for considering negative policy rules are twofold. First, they enable support of protection requirements that demand some accesses to be prohibited. Second, they enable support for exceptions to positive rules, making specification of authorizations more natural and convenient.

Different policy rules can apply to the same access request, and a conflict may arise when there exists at least one positive rule and one negative rule for the same access request. In this case, an access request is granted if there is at least a policy rule that allows the request, and no rule prohibits it (see Section 3.3 for a discussion on policy enforcement). We now describe the different components of a policy rule in more detail.

3.2.1 Subject

The *subject* component refers to a specific subject or group. *Subject* can also refer to a set of subjects depending on whether they satisfy or not certain conditions that are evaluated on their profile. Also, to make it possible to refer to the subject of the access request being evaluated without need of introducing variables in the language, we introduce the keyword **subject** that indicates the identifier of the subject making the request. In other words, such a keyword is substituted with the actual parameters of the access request in the evaluation at access control time. Note that the value is "undefined" in the case no actual value has been declared (e.g., when the access request is submitted by an anonymous subject).

We assume profiles to be referenced with the unique identifier of the corresponding subject. Single properties within subjects profiles are referenced using the dot notation. For instance, `Anna.address` indicates the address of subject `Anna`. Here, `Anna` is the identifier of the subject (and therefore the identifier for the corresponding profile), and `address` is the address property. As another example, **subject**`.registrationDate` indicates the property `registrationDate` within the profile of the subject whose request is being processed.
The *subject* component has the form:

(*subject-id*, [*condition*])

where:

- *subject-id* is the identifier of a subject or group of subjects;
- *condition* is a boolean formula of terms that are evaluated over the properties of the subject making an access request.

The following are examples of the *subject* component.

- (`Any`, _) denotes all subjects.
- (`Private`, _) denotes all subjects who belong to group `Private`.
- (`Any`, **subject**.`citizenship`=Italian AND **subject**.dob<1990-07-01) denotes all Italian citizens who are born before July 1, 1990.

3.2.2 Object

The *object* component identifies the object or the set of objects to which the policy rule applies. Like for subjects, we can specify conditions on the objects to which the policy rule applies. Such conditions may refer to the content of the objects, metadata associated with the objects, or the metadata associated with the properties of the objects. To make it possible to refer to the object/properties to which the request being processed refers or to the metadata associated with the object/properties referred in the access request, without need of introducing variables in the language, we introduce the following three keywords, whose appearance in the object component must be substituted with the actual parameters of the access request in the evaluation at access control time.

dataset indicates the identifier of the dataset to which access is being requested;

d_metadata indicates the identifier of the metadata associated with the dataset to which access is being requested.

a_metadata indicates the identifiers of the properties of the dataset to which access is being requested.

The *object* component has the form:

$$(\textit{object-id}[.\{a_1,\ldots,a_n\}],\ [\textit{condition}])$$

where:

- *object-id* is the identifier of an object or of a data category;
- $\{a_1,\ldots,a_n\}$ is a set of attributes appearing in the schema of object *object-id*, where *object-id* is the identifier of a dataset;
- *condition* is a boolean formula of conditions over the dataset content, or over the properties of the metadata associated with *object-id* or with its attributes.

The following are examples of the *object* component.

- (`Financial`, _) denotes all objects belonging to the `Financial` data category.
- (`Financial`, **d_metadata**.`creator`=BeSafe) denotes all datasets belonging to data category `Financial` that have been created by BeSafe.
- (`InsurancePlan`.{`id`, `name`, `surname`}, **dataset**.`type`=basic) denotes all rows of dataset `InsurancePlan` projected over attributes `id`, `name`, and `surname` and such that they correspond to users with a basic insurance.
- (`Financial`, **a_metadata**.`visibility`=public) denotes all public attributes of objects belonging to the `Financial` category.

Symbols	$\langle$, $\rangle$, (,), $\wedge$, $\vee$, $\neg$	parenthesis and boolean operators
Reserved identifiers	**subject**	bounded to the identity (if defined) of the subject making a request.
	dataset	bounded to the identifier of the dataset to which access is being requested.
	d_metadata	bounded to the identifier of the metadata associated with the dataset to which access is being requested.
	a_metadata	bounded to the identifier of the metadata associated with the properties of the dataset to which access is being requested.

Figure 3.2: List of symbols and reserved identifiers of the policy language

3.2.3 Operation

The *operation* component of a policy rule corresponds to the operation to which the policy rule refers. Note that abstractions can also be defined on operations, specializing operations or grouping them in sets. An operation corresponds then to the identifier of a basic operation, or the identifier of an abstraction. For instance, a basic operation can be `read`, and `download`. An abstraction can be `Any` (root of the operation hierarchy denoting any operation) or `Access` assuming that `Access` groups the basic operations.

3.2.4 Purpose

The *purpose* component of a policy rule allows the reference to a specific purpose or to an abstraction defined in the purpose hierarchy (Section 2.4). Considering the purpose hierarchy in Figure 2.6, examples of purposes are: `Any` (denoting any purpose), `Scientific`, `Education`, and `Commercial`.

3.2.5 Condition

The *condition* component defines conditions that, as already discussed in Section 2.6, can be of two kinds:

- conditions on subjects and objects;
- conditions on contextual information such a location and time of an access request.

Note that conditions on subject profiles, datasets, and metadata can be equivalently specified in the *condition* component or in the corresponding *subject* and *object* component. Since these two options do not affect the semantics of the rule, we assume that the *condition* component includes only conditions on contextual information. Such conditions are represented through predicates that trigger the corresponding checks. Figure 3.2 summarizes the list of symbols and reserved identifiers of the policy language. Figure 3.3 reports the BNF syntax of the access policy rules.

3.3 Enforcement of the policy rules

Given an access request, to determine whether the request is allowed or denied the policy engine has to retrieve all *applicable* policy rules. A policy rule applies to the given access request when the subject, object, operation, and purpose of the request *are covered* by the corresponding

policy_rule	::=	⟨subject, object, operation, purpose, [condition], sign⟩
subject	::=	(subject-id,[condition])
object	:=	(object-id[.{list}], [condition])
operation	::=	identifier
purpose	::=	identifier
subject-id	::=	identifier
object-id	::=	identifier
condition	::=	simple-condition | condition ∧ condition |
		condition ∨ condition | ¬ condition
simple-condition	::=	keyword.identifier math-op value |
		keyword.identifier math-op keyword.identifier | predicate
math-op	::=	= | < | > | <= | >= | IN
keyword	::=	**subject** | **dataset** | **d_metadata** | **a_metadata**
predicate	::=	identifier(list)
list	::=	identifier | list [, identifier]
value	::=	string
identifier	::=	letter | identifier [character]
character	::=	letter | digit
sign	::=	+ | −

Figure 3.3: BNF syntax of the access policy rules

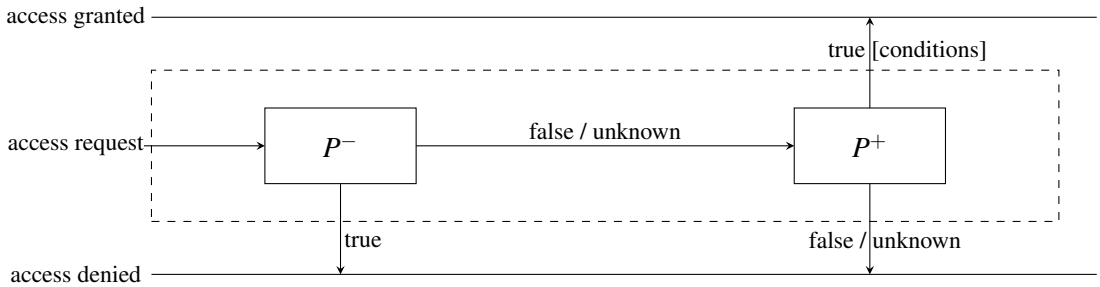

Figure 3.4: Policy evaluation flow

components of the policy rule. The coverage relationship is based on the fact that policy rules defined over subject, object, operation, and purpose abstractions propagate down in the corresponding hierarchies. The applicable policies are partitioned into two groups: P^- includes all negative rules and P^+ includes all positive rules. Figure 3.4 shows the policy evaluation flow. The negative rules are first evaluated against the access request. If the evaluation result is 'true' (i.e., there exists a negative rule such that the possible conditions in the subject, object, and condition components of the rule are satisfied by the access request), the access is denied. Otherwise, the request is redirected and evaluated against the set of applicable positive rules in P^+. If the evaluation result of policies in P^+ is 'true' (i.e., there exists a positive rule such that the possible conditions in the subject, object, and condition components of the rule are satisfied by the access request), the access is granted or "conditionally" granted. Otherwise, the access request is denied. Granted means that the access request is permitted as it is. Conditionally granted

Members of the HumanResource group cannot read for Any purposes the Financial information	
subject:	(HumanResource, _)
object:	(Financial,_)
operation :	read
purpose :	Any
condition :	TRUE
sign:	−

Members of the HumanResource group can read for Commercial purposes all attributes of type personal information of Company objects when connect from the company network	
subject:	(HumanResource, _)
object:	(Company, **a_metadata**.type=personal_info)
operation :	read
purpose :	Commercial
condition :	ORIGIN(mycompany.com)
sign:	+

A New Zealand national who is also member of the Marketing group can read for Scientific purposes the name, surname, dob, gender, and coverage of the insuranceholders in InsurancePlan who are leaving in New Zealand	
subject:	(Marketing, **subject**.citizenship=NZ)
object:	(InsurancePlan.{name, surname, dob, gender, coverage}, **dataset**.country=NZ)
operation :	read
purpose :	Scientific
condition :	TRUE
sign:	+

Figure 3.5: An example of access policy rules

means that the object of the access request has to be modified to take into consideration the conditions (if any) specified in the policy rule that are defined over the object content. As an example, consider the policy rules in Figure 3.5, and the access request ⟨Billy, InsurancePlan, read, Commercial⟩. The first two policy rules apply to the access request since Billy is connected from a PC of the mycompany network, is a member of the HumanResource group (see the profile of Billy in Figure 2.1), and InsurancePlan is a Financial dataset (see the metadata associated with InsurancePlan in Figure 2.3(b)). The negative rule is evaluated to true and therefore the access request is denied. Consider now another access request coming from Anna and stating that she is willing to read the attributes name, surname, dob, and gender of the InsurancePlan dataset for StatAnalysis purpose, that is, ⟨Anna, InsurancePlan.{name,surname,dob,gender}, read, StatAnalysis⟩. In this case, the third policy rule in Figure 3.5 is the only applicable policy. The evaluation of this rule against the access request returns 'true' with condition '**dataset**.country=NZ' since Anna is a New Zealand citizen and belongs to the Marketing group (see the profile of Anna in Figure 2.1). Furthermore, the attributes mentioned in the access request are a subset of the attributes specified in the policy rule, and StatAnalysis is a specialization of the Scientific purpose reported

InsurancePlan

name	surname	dob	gender	coverage
Alice	Rossi	1990-01-05	female	life
Eva	Clark	1978-05-05	female	vehicle
…	…	…	…	…

Figure 3.6: Dataset accessed by Anna

in the rule (see Figure 2.6). The access request is then "conditionally" accepted, meaning that the request has to be modified so to include the conditions on the dataset content specified in the policy rules. In the example, the applicable policy grants the access only to the tuples of the InsurancePlan that refer to New Zealand citizens. This constraint is then enforced when the access request is executed over the InsurancePlan dataset. Figure 3.6 shows the content of the InsurancePlan dataset that is returned to Anna. The result includes only the tuples of the insuranceholders leaving in New Zealand.

4. Ingestion policy

This chapter describes how the MOSAICrOWN policy model and language can be used for representing the *ingestion policies*. An ingestion policy is a high level policy that defines the protection to apply to the data uploaded to the data market. The expressive and flexible model designed in MOSAICrOWN can be adopted to cover these wide requirements. The uniformity in the syntax greatly facilitates the representation and management of the policies. Ingestion is associated with two goals: (1) selection of the portion of data to transfer to the data market; and (2) transformations that have to be applied when uploading the data (wrapping and sanitization). The experience in MOSAICrOWN clarifies that the domain to cover in the representation of transformation requirements is extremely wide. The use of semantic web technology provides flexibility, but a significant effort is still required to build a software able to cover the semantic gap between the description of the policy and its application.

The concrete development of the proposed adaptation to ingestion of the policy derives from the consideration of all the use cases, as described in the next chapter. The most interesting application is represented by Use Case 2, where declarative policies specify the protection techniques that can be applied to data in a way that complies with multiple privacy regulations. The data wrapping technique to be applied depends on their content and different policies can be applied to each market. This is particularly relevant for international companies that operate in many markets, each with its own regulation.

4.1 Policy rules

An ingestion policy is composed by a set of rules that regulate how datasets or properties of datasets must be transformed before being stored in the data market. An ingestion policy rule has the form:

$$\langle subject, object, transformation, purpose, output \rangle$$

- *subject* identifies the subject to which the policy rule refers;
- *object* identifies the object to which the policy rule refers;
- *transformation* identifies the transformation to which the policy rule refers;
- *purpose* identifies the purpose (or a purpose abstraction) to which the policy rule refers;
- *output* identifies the output of the *transformation*.

Informally, an ingestion policy rule states that *subject* can perform *transformation* over *object* with which is associated purpose *purpose*. The result of the transformation is a new object identified by *output* and inheriting the metadata associated with *object* extended with the metadata that describe

how the new object has been computed (i.e., the metadata corresponding to the transformation performed over the object). We now describe more in details the different components of an ingestion policy rule.

Subject. The *subject* component refers to the subject in charge of transforming an object. For instance, a subject could be the data market provider or an external party. The *subject* component corresponds to the identifier *subject-id* of a subject.

Object. The *object* component identifies the object or the set of objects to which the ingestion policy rule applies. Like for the access policy rules, we can specify conditions on the objects to which the ingestion policy rule applies. Such conditions may refer to the metadata associated with the objects, or to the metadata associated with the properties of the objects. Note that while the access policy rules also support conditions on the content of objects, the ingestion policy rules do not support them because a transformation always applies to the whole content of an object or, in case of structured objects, to a subset of the attributes of an object. Also for the ingestion policy rules we use keyword **d_metadata** for referring to the metadata associated with the object that is ingested in the data market, and keyword **a_metadata** for referring to the properties of the ingested object. The *object* component has then the form:

$$(\textit{object-id}[.\{a_1,\ldots,a_n\}],\ [\textit{condition}])$$

where:

- *object-id* is the identifier of an object or of a data category;
- $\{a_1,\ldots,a_n\}$ is a set of attributes appearing in the schema of object *object-id*, where *object-id* is the identifier of a dataset;
- *condition* is a boolean formula of conditions over the metadata associated with *object-id* or associated with its attributes.

The following are examples of the *object* component.

- (`CardHolder`, **a_metadata**.`type`=identifier) refers to the attribute of dataset `CardHolder` of `type` 'identifier';
- (`Financial`, _) refers to datasets that belong to the `Financial` category;
- (`Any`, **d_metadata**.`creator`=Til) refers to any dataset created by 'Til'.

Transformation. The *transformation* component identifies the transformation that has to be enforced on the *object*. As discussed in Section 2.3.3, MOSAICrOWN considers two types of transformations: wrapping (e.g., encryption) and sanitization (e.g., k-anonymity, ℓ-diversity). By analyzing the different transformations, we can see that each of them is characterized by a *signature* stating the input parameters and the format of the output. A transformation can then be modeled as a predicate with a signature of the form TRANSFORMATION_NAME(*list_of_parameters*)::*output*. As we will see later on, the output of a transformation is modeled through the *output* component of the ingestion policy rule. The following are examples of transformations, where *object* and *output* are the components of an ingestion policy rule.

policy_rule	::=	⟨subject, object, transformation, purpose, output⟩
subject	::=	subject-id
object	:=	(object-id[.{list}], [condition])
transformation	::=	identifier(list)
purpose	::=	identifier
output	::=	identifier | identifier/identifier | **dataset**
subject-id	::=	identifier
object-id	::=	identifier
condition	::=	simple-condition | condition ∧ condition | condition ∨ condition | ¬ condition
simple-condition	::=	keyword.identifier math-op value | keyword.identifier math-op keyword.identifier
math-op	::=	= | < | > | <= | >= | IN
keyword	::=	**d_metadata** | **a_metadata**
list	::=	identifier | list [, identifier]
value	::=	string
identifier	::=	letter | identifier [character]
character	::=	letter | digit

Figure 4.1: BNF syntax of the ingestion policy rules

- TUPLE_SYMMETRIC_ENCRYPTION(*key*): symmetric encryption applied at the tuple level over object *object* with key *key*. The output of this transformation is *output*.

- KANONYMITY(k): k-anonymity applied over object *object* with parameter k. The output of this transformation is *output* that represents a k-anonymous version of the *object*.

Purpose. Like for the access policy rules, the *purpose* component allows the reference to a specific purpose or to an abstraction defined in the purpose hierarchy (Section 2.4).

Output. The *output* component corresponds to the unique identifier of the object obtained after the application of the transformation specified in *transformation* to object *object*. The identifier in the *output* component can then appear in the *object* component of the access policy rules, thus allowing the specification of rules that regulate the access to the transformed object. For concreteness, in the following examples, the *output* component resembles the form of an URI (see Section 4.2) where keyword **dataset** can be used to refer to the identifier of the original object. For instance, suppose that the *output* component of an ingestion policy rule includes value "wrapped/**dataset**" and that keyword **dataset** is bounded to `CardHolder`. Then, "wrapped/**dataset**" corresponds to "wrapped/`CardHolder`", which is the identifier of the protected version of `CardHolder`.

Figure 4.1 reports the BNF syntax of the ingestion policy rules, and Figure 4.2 illustrates an example of ingestion policy rules.

The Data Market must encrypt all objects created by Til after 1940-01-01 for `Commercial` purposes	
subject:	`DataMarket`
object:	(`Any`,**d_metadata**.`creator`=Til ∧ **d_metadata**.`date` > 1940-01-01)
transformation :	TUPLE_SYMMETRIC_ENCRYPTION(key1)
purpose :	`Commercial`
output:	wrapped/**dataset**

The Data Market must generalize the quasi-identifier attributes of all the `Financial` datasets produced for `Any` purposes up to level lev1	
subject:	`DataMarket`
object:	(`Financial`, **a_metadata**.`type`=quasi-identifier)
transformation :	GENERALIZE(lev1)
purpose :	`Any`
output :	sanitized/**dataset**

Figure 4.2: An example of ingestion policy rules

4.2 Encoding of the MOSAICrOWN ingestion policies

Like for the access policies, MOSAICrOWN ingestion policies are expressed in ODRL [IV18]. We have then reused the ODRL constructs when possible and added new ones when needed for representing concepts that are specific to our policies. In particular, the components of the MOSAICrOWN ingestion policy rules can be mapped onto the following ODRL entities.

- *subject* corresponds to **assignee** and its value must be an URI.
- *object* corresponds to **target** and its value must be an URI.
- *transformation* corresponds to **action** and its value can be *derive* or *anonymize* denoting the application of a wrapping technique (*derive*) or of a sanitization technique (*anonymize*). The *refinement* property is then used for defining the specific transformation that has to be applied to the target. This property includes a constraint composed by **leftOperand**, **operator**, and **rightOperand**.
  - **leftOperand** takes as a value one of the new keywords defined in the MOSAICrOWN vocabulary. These keywords are used for defining the transformations and their parameters. In particular, keyword *mosaicrown:method* means that the constraint is defining a wrapping/sanitization technique that is then specified in **rightOperand**.
  - **operator** always takes value *eq*, which corresponds to the equality operator.
  - **rightOperand** can be a generic ODRL node that describes the specific technique employed. In the following examples, it is a *xsd:string* containing the name of the selected method.

  The *refinement* property can also include the *Logical Constraint* construct of ODRL, meaning that the transformation can be the result of a combination of different techniques.
- *purpose* corresponds to **purpose**.

```
{
  "@context": [
      "http://www.w3.org/ns/odrl.jsonld",
      "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "@type": "Set",
  "permission": [{
    "assignee": "http://org/datamarket",
    "target": {
      "@id": "http://org/data/Customer/accountid",
      "mosaicrown:semanticType": "identifier"
    }
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": {
        "leftOperand": "mosaicrown:method",
        "operator": "eq",
        "rightOperand": {
            "@value": "deterministic tokenization",
            "@type": "xsd:string"
        }
      }
    },
    "output": "http://org/data/wrapped/Customer/accountid",
    "purpose": "Any"
  }]
}
```

Figure 4.3: An example of ingestion policy rule specifying a single wrapping technique

- *output* corresponds to **output** and its value must be an URI that will be used to identify the output of the transformation. In the following examples, the URI shares the same structure as the URI of the target, but it is part of the *wrapped* data collection.

Figure 4.3 and Figure 4.4 show two examples of ingestion policy rules. The first ingestion policy rule in Figure 4.3 states that the data market has to apply the deterministic tokenization technique over attribute `accountid` of dataset `Customer` whatever is the purpose associated with the dataset. The resulted dataset with the protected attribute is part of the *wrapped* data collection as specified in the **output** component. The second ingestion policy rule in Figure 4.4 states that the data market has to apply suppression *and* then distortion on attribute `residenceaddress` of dataset `Customer` whatever is the purpose associated with the dataset. The resulted dataset with the protected attribute is part of the *wrapped* data collection as specified in the **output** component.

As discussed in the previous section, an ingestion policy rule can also refer to objects that satisfy specific conditions evaluated over their metadata. Figure 4.5 presents an example of two ingestion policies, one policy with uid *http://org/policy/identifier* and another policy with uid *http://org/policy/Customer/accountid*. The former states that data with `Any` purpose and classified as identifier must be transformed by the data market through the deterministic tokenization technique. Note that this policy rule does not have a **target** component since it is defined for a category of data (i.e., identifier data).

The second policy inherits the transformation specified in the previous policy and adds the

```
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "@type": "Set",
  "permission": [{
    "assignee": "http://org/datamarket",
    "target": {
      "@id": "http://org/data/Customer/residenceaddress",
      "mosaicrown:semanticType": "Address"
    },
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": {
        "and": [{
            "leftOperand": "mosaicrown:method",
            "operator": "eq",
            "rightOperand": {
              "@value": "suppression",
              "@type": "xsd:string"
            }
          },{
            "leftOperand": "mosaicrown:method",
            "operator": "eq",
            "rightOperand": {
              "@value": "distortion",
              "@type": "xsd:string"
            }
          }]
      }
    },
    "output": "http://org/data/wrapped/Customer/residenceaddress",
    "purpose": "Any"
  }]
}
```

Figure 4.4: An example of ingestion policy rule specifying multiple wrapping techniques

information about a specific target on which it applies. Specifically, this policy includes the following components.

- **inheritFrom**: the URI of the parent policy that is inherited by the current one. In the example, it is the policy that specifies how to protect identifying data.
- **target**: the URI corresponding to the *object* that is the target of the transformation.
- **output**: the URI associated with the output of the computation. The data owner can use this information to define access policies on the new transformed resource.

Figure 4.6 illustrates an example of how access policies can be defined over the objects produced by the application of a transformation. Specifically, the example shows two permissions stating that members of the `Administrative` group can `read` the wrapped `Customer` dataset

```
[{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://org/policy/identifier",
  "permission": [{
    "assignee": "http://org/datamarket",
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": {
        "leftOperand": "mosaicrown:method",
        "operator": "eq",
        "rightOperand": {
          "@value": "deterministic tokenization",
          "@type": "xsd:string"
        }
      }
    },
    "purpose": "Any"
  }]
},
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://org/policy/Customer/accountid",
  "inheritFrom": "http://org/policy/identifier",
  "target": "http://org/data/Customer/accountid",
  "output": "http://org/data/wrapped/Customer/accountid"
}]
```

Figure 4.5: An example of ingestion policy defined for data of type “identifier” and its application to a specific target

for `Any` purpose, and that `agentA` can `use` the `accountid` attribute for `Statistical` purposes. Note that action `use` is more generic than the `read` action, and therefore the permission of `agentA` over `accountid` is more powerful than the permission of any other subject who is member of the `Administrative` group.

```
{
  "@context":[
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "@type":"Set",
  "uid": "http://org/policy/wrapped/Customer",
  "permission": [{
      "uid": "http://org/policy/wrapped/Customer/adminCanRead",
      "assignee": "http://org/user/Administrative",
      "target": "http://org/data/wrapped/Customer",
      "action": "read",
      "purpose": "Any"
    }, {
      "uid": "http://org/policy/wrapped/Customer/agentACanUse",
      "assignee": "http://org/user/Administrative/agentA",
      "target": "http://org/data/wrapped/Customer/accountid",
      "action": "use",
      "purpose": "Statistical"
    }]
}
```

Figure 4.6: An example of access policy defined over a wrapped dataset

5. Use case support

This chapter illustrates the application of ingestion policies to regulate transformations on data for enforcing protection when data are generated hence fed into the data market in the different use cases of our project. In particular, the chapter shows the ingestion policies for data acquired from electronic vehicles (Use Case 1), financial data from different sources (Use Case 2), and sanitized data for analytics (Use Case 3).

5.1 Use Case 1: Data acquisition

The main goal of Use Case 1 (UC1) is to design and develop automotive tools facilitating the ingestion of data from electric vehicles (EV) respecting the privacy of their drivers. In this use case, the vehicle fleet manager and the EV charging infrastructure provider exchange data to derive mutually beneficial insights about the status of the electric vehicle charging infrastructure. Such data can include sensitive or Personally Identifiable Information (PII) about the EV drivers and therefore the EV drivers have to specify how their data have to be shared and protected. To this purpose, UC1 defines the following three predefined policies that a EV driver can select.

- **Incognito**: a policy with the most private policy settings. With this policy, the data about an EV driver cannot be shared with anyone.
- **Confidential**: a policy with "moderate" privacy settings. With this policy, the data about an EV driver can only be shared with the data market and must be stored in encrypted form.
- **Public**: a policy with the least privacy preserving setting (i.e., data are accessible without any restriction). With this policy, the data about an EV driver can be shared with the data market and there are no any further restrictions on the use of such data.

The ingestion policies introduced in the previous chapter can be adopted to easily specify how the data ingested in the data market have to be treated depending on the policy chosen by the EV driver owning such data. To this purpose, the MOSAICrOWN language can be used for specifying the UC1 policy profile associated with an EV driver as well as the transformations (ingestion policies) associated with the UC1 policy profile. Figure 5.1 illustrates a possible way of representing the UC1 policy profiles with the MOSAICrOWN ingestion policy language, and Figure 5.2 shows an example of an EV driver (i.e., DDLW4R7BKF) using the confidential policy for protecting her data. Once data are submitted along with the EC driver decision, it is possible to combine the meta-policy (Figure 5.1) with the URI of the data to obtain the policy shown in Figure 5.2.
In the example in Figure 5.2, it is possible to note that the final document contains the information necessary to:

- identify the original data that are being submitted (**target** component);

```
[{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://UC1/policy/incognito",
  "prohibition": [{
    "assignee": "http://UC1/dataacquirer",
    "action": "any", "purpose": "archive"
  }]
},{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://UC1/policy/confidential",
  "permission": [{
    "assignee": "http://UC1/dataacquirer",
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": {
        "leftOperand": "mosaicrown:method",
        "operator": "eq",
        "rightOperand": { "@value": "encrypt",
                          "@type": "xsd:string" }
      }
    },
    "purpose": "archive"
  }]
},{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://UC1/policy/public",
  "permission": [{
    "assignee": "http://UC1/dataacquirer",
    "action": "any", "purpose": "archive"
  }]
}]
```

Figure 5.1: An example of ingestion policies corresponding to the UC1 policy profiles

- allow the data market to perform the required transformation over the ingested data, expressed in the ingestion policy identified in the *inheritFrom* component.
- identify (component *output*) the final data collection that will be stored in the data market.

5.2 Use Case 2: Data wrapping

The main goal of Use Case 2 (UC2) is to enable collaborative analytics over financial microdata. Due to the sensitive nature of such data, they have to be properly protected when stored in the

```
[{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://UC1/policy/ev/DDLW4R7BKF",
  "inheritFrom": "http://UC1/policy/confidential",
  "target": "http://UC1/data/ev/DDLW4R7BKF",
  "output": "http://UC1/data/wrapped/ev/DDLW4R7BKF"
}]
```

Figure 5.2: Example of how to assign a UC1 policy profiles to an EV driver

data market. In the following, we then describe how the MOSAICrOWN ingestion policies can be adopted in the UC2 pipeline to automatically produce a wrapped or anonymized dataset starting from a source dataset.

5.2.1 Overview of the pipeline

A key concept in the UC2 is the *semantic data type* that determines how data must be protected. This information is then at the basis of the definition of the appropriate ingestion policies. Figure 5.3 shows the UC2 pipeline modified to take into consideration the generation of the ingestion policies. The pipeline takes the semantic data type produced by the detection model developed by Mastercard and the regulations that need to be enforced as input. Based on these two inputs, and the meta-policies provided by the DPO, the MOSAICrOWN ingestion policy rules are automatically generated by a dedicated *Policy Producer*. Such rules specify all the possible outputs resulting from the transformations of the available targets. Since the targets and outputs are fully qualified by unique URIs:

- each *output* can later be used as a **target** of access policies;
- ingestion policy rules can be used as an input to produce the configuration (UC2 configuration files) required by different UC2 dataprocessor implementations.

In the following, we describe the UC2 pipeline more in details.

5.2.2 Semantic data type generation

UC2 integrates a proprietary semantic data type detection model into the analytics platform to automatically identify the semantic types associated with the attributes of a source dataset. As depicted in Figure 5.4, the detection algorithm takes as input an attribute and produces as output the attribute type, that is, a metadata that classify the attribute. In particular, the algorithm first retrieves the information about the primitive type related to each attribute (e.g., integer, string, alphanumeric). Then, each attribute is analyzed to determine its semantic type (e.g., name, address, zipcode). For instance, Figure 5.4 shows three attributes of a dataset that are associated with the string type and subsequently classified as a *name*, *address*, and *date*, respectively.

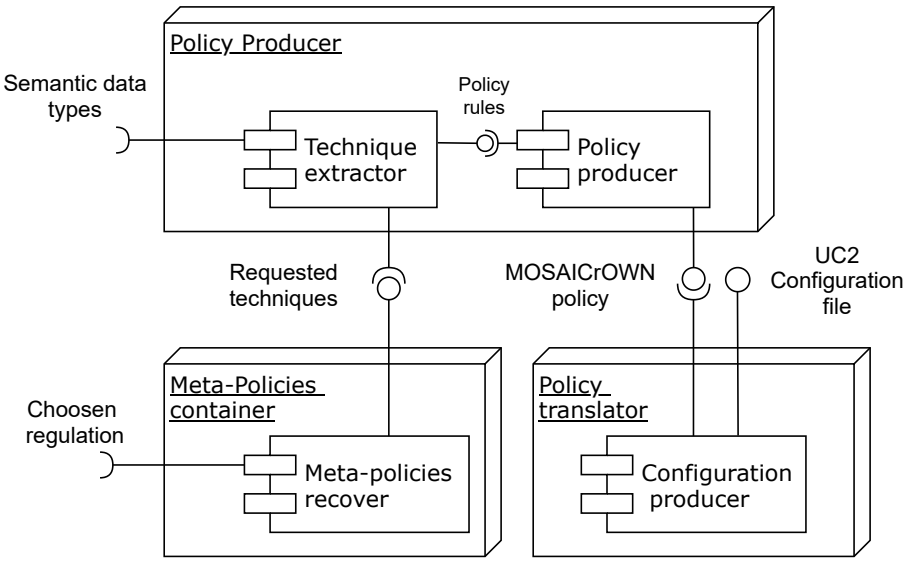

Figure 5.3: UC2 pipeline

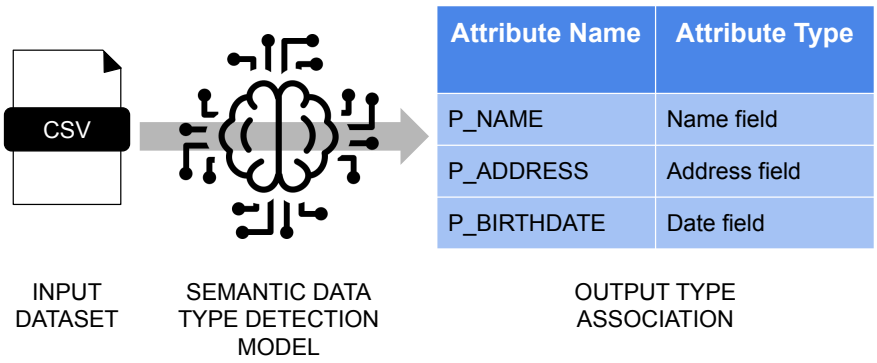

Figure 5.4: Workflow realized in UC2

5.2.3 Metadata-driven transformations and policy translation

The second step of UC2 pipeline aims at producing automatically the ingestion policy rules to transform the source dataset into a wrapped representation. This step takes as input the metadata produced by step 1 (metadata generation), as well as the regulations to be enforced and the meta-policies written in JSON by the DPO. The regulations depend on the environment in which the analytics platform is deployed. For instance, the platform could be located in Europe, meaning that the source datasets need to be transformed according to the restrictions imposed by GDPR. The meta-policies instead specify the kind of transformations that need to be applied to a given semantic type. The same transformation can be applied to multiple attributes within a source dataset and may depend on the regulations to be enforced. Figure 5.5 shows an example of meta-policy. It defines two rules: the first is associated with the GDPR privacy regulation and states that any attribute classified as *identifier* must be protected with *deterministic tokenization*; the second

```
[
  {
    "privacy_id": "1",
    "privacy_name": "General Data Protection Regulation",
    "privacy_acr": "GDPR",
    "dwt_associations": {
      "identifier": [
        "Deterministic Tokenization"
      ],
      "address": [
        "Distortion",
        "Suppression"
      ]
    },
    "area": "Europe"
  },
  {
    "privacy_id": "2",
    "privacy_name": "Local Government Development Program",
    "privacy_acr": "LGDP",
    "dwt_associations": {
      "identifier": [
        "Deterministic Tokenization"
      ],
      "address": [
        "Distortion",
        "Suppression",
        "K-based hashing"
      ]
    },
    "area": "Brasil"
  }
]
```

Figure 5.5: An example of UC2 meta-policy that describes the data wrapping techniques to be applied depending on the semantic data type

rule is associated with the LGDP regulation and states that any attribute classified as *address* must be protected with *distortion*, *suppression*, and *k-based hashing*. Note that since meta-policies can be written according to different dialects, they have to be translated into the MOSAICrOWN policy language to take advantage of the governance framework tools. Figure 5.6 illustrates an example of MOSAICrOWN ingestion policy derived from the meta-policy in Figure 5.5.

5.2.4 Compatibility with the dataprocessor

The UC2 dataprocessor exposes an API that receives a JSON input specifying how to transform a dataset. Figure 5.7 shows an example of valid input for the UC2 dataprocessor. As it is visible from this figure, the input specifies:

- **column_name**, the name of the attribute to be transformed;
- **dwt**, a list of wrapping techniques that must be applied to the attribute;

```
[{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://UC2/policy/identifier",
  "permission": [{
    "assignee": "http://UC2/dataprocessor",
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": {
        "leftOperand": "mosaicrown:method",
        "operator": "eq",
        "rightOperand": {
          "@value": "deterministic tokenization",
          "@type": "xsd:string"
        }
      }
    },
    "purpose": "any"
  }]
},
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "uid": "http://UC2/policy/Customer/accountid",
  "inheritFrom": "http://UC2/policy/identifier",
  "target": "http://UC2/data/Customer/accountid",
  "output": "http://UC2/data/wrapped/Customer/accountid"
}]
```

Figure 5.6: An example of ingestion policy derived from the meta-policy and a dataset with the `accountid` attribute

- **type**, the semantic type of the attribute.

The input also includes the regulation under which the dataset is being transformed. This can be set with the `privacy_acr` element. The translation of a MOSAICrOWN ingestion policy into the dataprocessor input format is performed by the final policy translation step reported in Figure 5.3. In particular:

- each *data_wrapping* list item is obtained by a single MOSAICrOWN ingestion policy rule;
- *column_name* is part of the URI that identifies the target *object*;
- the list of data wrapping techniques (i.e., the *dwt*) is represented by the *refinement* clause of the *derive* operation;
- the *type* element is part of the URI identifying the policy and corresponds to the semantic data type metadata;
- the *policy_acr* element is derived from the *constraint* clause.

```
{
  "data_wrapping": [
    {
      "column_name": "accountid",
      "dwt": [
        "deterministic tokenization"
      ],
      "type": "identifier"
    },
    {
      "column_name": "ResidenceAddress",
      "dwt": [
        "suppression",
        "distortion"
      ],
      "type": "address"
    }
  ],
  "privacy_acr": "GDPR (Europe)"
}
```

Figure 5.7: An example of valid input for the UC2 dataprocessor

5.3 Use Case 3: Data sanitization

The main goal of Use Case 3 (UC3) is to develop techniques for supporting privacy-preserving analytics. Such analysis are performed over datasets that are properly sanitized. The sanitization techniques to apply to the datasets before storing them in the data market can be specified through the MOSAICrOWN ingestion policies. Figure 5.8 and Figure 5.9 illustrate two ingestion policies where the sanitization techniques are differential privacy and k-anonymity, respectively.

In particular, the ingestion policy in Figure 5.8 states that the dataset `Customer` must be transformed through the application of (ε, δ)-differential privacy, where $\varepsilon = 1$ and $\delta = 10^{-5}$. The sanitized dataset is identified via the URI reported in the *output* component. The ingestion policy in Figure 5.9 states that dataset `InsurancePlan` must be transformed through the application of k-anonymity, with $k = 10$, and that attributes `id`, `name`, and `surname` are identifiers (and they must then be suppressed) and that attributes `dob` and `gender` are quasi-identifiers (and they must satisfy 10-anonymity). Again, the sanitized dataset is identified via the URI reported in the *output* component. Since the majority of the tools developed in the UC3 expose their functionality through a REST-API, it is possible to implement a translator similar to the one employed in the last step of the pipeline of UC2 illustrated in Figure 5.3.

```
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "@type": "Set",
  "permission": {
    "assignee": "http://org/dataprocessor",
    "target": "http://org/data/Customer",
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": [{
        "leftOperand": "mosaicrown:method",
        "operator": "eq",
        "rightOperand": {
          "@value": "differential-privacy",
          "@type": "xsd:string"
        }
      },{
        "leftOperand": "mosaicrown:epsilon",
        "operator": "eq",
        "rightOperand": {
          "@value": "1",
          "@type": "xsd:integer"
        }
      },{
        "leftOperand": "mosaicrown:delta",
        "operator": "eq",
        "rightOperand": {
          "@value": "1e-5",
          "@type": "xsd:float"
        }
      }]
    },
    "output": "http://org/data/anonymous/DPCustomer",
    "purpose": "statistical"
  }
}
```

Figure 5.8: An example of ingestion policy for differential privacy

```
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "http://localhost:8000/ns/mosaicrown/namespace.jsonld"
  ],
  "@type": "Set",
  "permission": {
    "assignee": "http://org/dataprocessor",
    "target": "http://org/data/InsurancePlan",
    "action": {
      "rdf:value": { "@id": "odrl:derive" },
      "refinement": [{
        "leftOperand": "mosaicrown:method",
        "operator": "eq",
        "rightOperand": {
          "@value": "k-anonymity",
          "@type": "xsd:string"
        }
        },{
          "leftOperand": "mosaicrown:k",
          "operator": "eq",
          "rightOperand": {
              "@value": "10",
              "@type": "xsd:integer"
            }
        },{
          "leftOperand": "mosaicrown:identifier",
          "operator": "isPartOf",
          "rightOperand": [
              "http://org/data/InsurancePlan/id",
              "http://org/data/InsurancePlan/name",
              "http://org/data/InsurancePlan/surname"
          ]
        },{
          "leftOperand": "mosaicrown:quasi-identifier",
          "operator": "isPartOf",
          "rightOperand": [
              "http://org/data/InsurancePlan/dob",
              "http://org/data/InsurancePlan/gender"
          ]
        }]
    },
    "output": "http://org/data/anonymous/KAnonInsurancePlan",
    "purpose": "statistical"
  }
}
```

Figure 5.9: An example of ingestion policy for *k*-anonymity

6. Conclusions

This deliverable has presented the model and policy specification language for MOSAICrOWN. The policy work is at the center of the data governance framework to which Work Package 3 is devoted, as it enables data owners to specify (and then have enforced) policies regulating processing, sharing, and use of their data. The model and language have been designed with the goal to ensure simplicity and easiness of use on one side, and expressiveness and flexibility on the other side. Chapter 1 has discussed the state-of-the-art and illustrated the innovative aspects of the proposed policy model and language. Chapter 2 has introduced the basic concepts and elements of the policy model. The model responds to the requirements identified and nicely accounts for inclusion in the policy specification of wrapping and sanitization techniques developed in Work Packages 4 and 5, respectively. Chapter 3 has presented the MOSAICrOWN access policy. Access policies define policy rules concerning access to objects in the data market. Policy rules correspond to authorizations that need to be checked when a subject submits an access request. The chapter has then recalled the format of an access request, and then has illustrated the different components of the policy rules, their semantics, and how the policies are enforced. Chapter 4 has presented the MOSAICrOWN ingestion policy. Ingestion policies regulate how the data moved to the data market have to be stored. The chapter has described the format of the rules and their semantics and then shown how the high-level ingestion policy specifications can be encoded in ODRL, a W3C standard also chosen for expressing the access policy. The goal was to enjoy interoperability with current technology so to enable the deployment of the MOSAICrOWN policies in real-life scenarios. Finally, Chapter 5 has described how the ingestion policies can be used for supporting the MOSAICrOWN use cases in the ingestion phase.

Bibliography

[ABD19] E. Arfelt, D. Basin, and S. Debois. Monitoring the GDPR. In *Proc. of the 24th European Symposium on Research in Computer Security (ESORICS)*, Luxembourg, September 2019.

[ACK+10] C.A. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio. Exploiting cryptography for privacy-enhanced access control: A result of the PRIME project. *Journal of Computer Security (JCS)*, 18(1):123–160, January 2010.

[ADF+12] C.A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati. Minimising disclosure of client information in credential-based interactions. *International Journal of Information Privacy, Security and Integrity (IJIPSI)*, 1(2/3):205–233, 2012.

[BDF+18] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Protecting resources and regulating access in cloud-based object storage. In I. Ray, I. Ray, and P. Samarati, editors, *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of his 70th Birthday*. Springer, 2018.

[BK19] P.A. Bonatti and S. Kirrane. Big data and analytics in the age of the GDPR. In *Proc. of the 2019 IEEE International Congress on Big Data*, Los Angeles, CA, USA, December 2019.

[BNRV20] C. Brewster1, B. Nouwt, S. Raaijmakers, and J. Verhoosel. Ontology-based access control for FAIR data. *Data Intelligence*, 2(1-2):66–77, January 2020.

[Böh21] J. Böhler, editor. *D5.5 - Report on Data Sanitisation and Computation*. Deliverable of MOSAICrOWN, October 2021.

[BS02] P.A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security (JCS)*, 10(3):241–271, 2002.

[BS03] P. Bonatti and P. Samarati. Logics for authorizations and security. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.

[De 21] S. De Capitani di Vimercati, editor. *D3.4 – Final Tools for the Governance Framework*. Deliverable of MOSAICrOWN, September 2021.

[DFJ+10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)*, 35(2):12:1–12:46, April 2010.

[DFLS21] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Towards owners' control in digital data markets. *IEEE Systems Journal (ISJ)*, 15(1):1299–1306, March 2021.

[DG21] S. De Capitani di Vimercati and F. Giusto, editors. *D4.4 - Report on Encryption-based Policy Enforcement and Controlled Query Execution*. Deliverable of MOSAICrOWN, October 2021.

[DS20] S. De Capitani di Vimercat and P. Samarati, editors. *D3.3 - First Version of Policy Specification Language and Model*. Deliverable of MOSAICrOWN, June 2020.

[GPW+20] S. Ghayyur, P. Pappachan, G. Wang, S. Mehrotra, and N. Venkatasubramanian. Designing privacy preserving data sharing middleware for internet of things. In *Proc. of the 3rd International SenSys+BuildSys Workshop on Data:Acquisition to Analysis (DATA 2020)*, November 2020.

[IV18] R. Iannella and S. Villata. ODRL information model 2.2, 2018. https://www.w3.org/TR/odrl-model.

[JSSS01] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260, June 2001.

[Par20] S. Paraboschi, editor. *D4.3 - Final Encryption-based Techniques*. Deliverable of MOSAICrOWN, December 2020.

[PG18] M. Palmirani and G. Governatori. Modelling legal knowledge for GDPR compliance checking. *Legal Knowledge and Information Systems*, 313:101–110, 2018.

[ZDX+20] Y. Zhang, R.H. Deng, S. Xu, J. Sun, Q. Li, and D. Zheng. Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys*, 53(4), August 2020.