AMdEX Reference Architecture – version 1.0.0

L. Thomas van Binsbergen[1], Merrick Oost-Rosengren[1,2], Hayo Schreijer[3], Freek Dijkstra[4], and Taco van Dijk[5]

[1]University of Amsterdam
[2]AMS-IX
[3]deXes
[4]Surf
[5]Waag

February 2024

Contents

Chapter 1

Introduction

1.1 About this document

Status of the document The current version of the document is 1.0.0; the first publicy released version. A revised version of the high-level architecture is found in Chapter 2 and has been extended with a section on 'connecting to AMdEX'. A complete and revised description of the UNL use case is found in Section 4.1. The RDX and DIPG use cases are in their first, complete version. The chapter on policy administration and enforcement has seen its first revisions. An index has been added.

Refer to this document as follows
L. Thomas van Binsbergen, Merrick Oost-Rosengren, Hayo Schreijer, Freek Dijkstra, and Taco van Dijk. *AMdEX Reference Architecture – version 1.0.0.* Ed. by L. Thomas van Binsbergen. Feb. 2024. DOI: `10.5281/zenodo.10565915`

Intended audience This document is intended for organizations and individuals interested in learning about the AMdEX approach and/or interested in participating in AMdEX as a member of the AMdEX community, service providers in particular. Contributors to other data exchange initiatives might find it interesting to learn about the AMdEX approach and to gain insight into how their initiative may connect to AMdEX.

Acknowledgments The authors would like to express gratitude to Mike Kotsur for feedback on earlier versions of this document and to participants of AMdEX community events for providing feedback. Further gratitude is expressed to the students of the Software Engineering Master programme of the University of Amsterdam who contributed to parts of the architecture in this document, in particular: Jorrit Stutterheim, Quinten Colthof, Tessa van Lobbrecht, Jonathan Karels, and Florine de Geus.

Purpose of the document The primary goal of this document is to describe the AMdEX approach to sharing data and the AMdEX reference architecture comprised of governance solutions and technical solutions. Central to the approach are the AMdEX community members that cooperate as a consortium within AMdEX dataspaces and within a certain

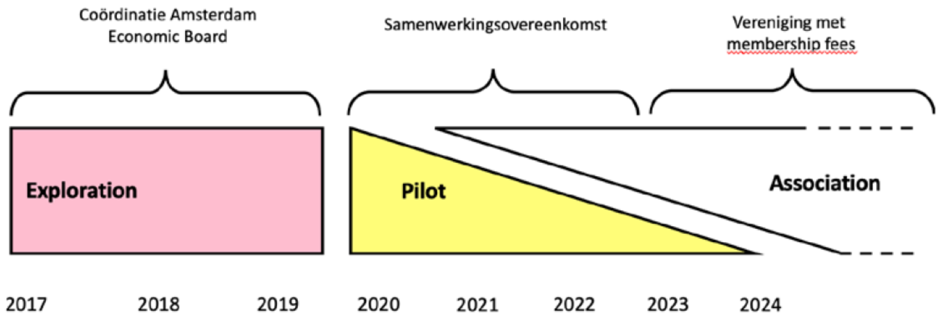

Coördinatie Amsterdam Economic Board
Samenwerkingsovereenkomst
Vereniging met membership fees
Exploration
Pilot
Association
2017
2018
2019
2020
2021
2022
2023
2024

Figure 1.1: The timeline for the AMdEX initiative laid out in 2020.

ecosystem. This document also explains how AMdEX works in practice, both from a user's perspective, in relation to other initiatives, and by presenting existing use cases. In particular, this document describes:

- The principles guiding the design of the AMdEX technical architecture
- The users, roles, components and APIs that form AMdEX dataspaces and ecosystems
- The administration and enforcement of policies within AMdEX
- Use cases from the AMdEX Fieldlab, explained in terms of the aforementioned roles and components

An important aspect of AMdEX is the explicit connection made between data sharing agreements and conditions and applicable laws (soft infrastructure) on the one hand and the (hard) technical infrastructure on the other hand. This document describes how that connection is made and the process by which laws and agreements are integrated into the technical domain.

1.2 Initiative and Fieldlab Project

The Amsterdam Data Exchange (AMdEX) initiative is a collaboration between the organizations Amsterdam Internet Exchange (AMS-IX), Amsterdam Economic Board (AMEC), the University of Amsterdam (UvA), Surf, and Dexes. The founders joined forces with the goal of developing neutral, generic, and independent infrastructure that ensures (data) sovereignty in data sharing initiatives. In 2020, the timeline for the initiative depicted in Figure 1.1 was laid out.

After the initial exploration phase, a fieldlab project was funded by the 'European Regional Development Fund' (ERDF) or 'Europees Fonds for Regionale Ontwikkeling' (EFRO), providing the opportunity to develop pilots and to work out the operations of the future AMdEX association. The main goals of the AMdEX Fieldlab project were to:

- Develop a generic prototype with basic functionality in support of data exchange processes with:

  - an open, federated infrastructure
  - that supports at least a few data exchange 'archetypes' (see 2.1 and 3.2.2),
  - potentially scales to many-to-many applications and large data volumes, and
  - the possibility for service providers to test and improve their solutions.

- Execute a number of selected use cases with which the functionality of the prototype can be tested in practice.
- Develop business cases for scaling to a fully operational AMdEX organization.
- Interact with regional, national, and international initiatives to build a community and (potential) collaborations.

The design for the open, federated infrastructure is discussed in Chapter 2. The use cases executed within the project are described in Chapter 4. The first version of this document serves as a deliverable within the AMdEX Fieldlab project. Specifically, this document (partially) substantiates the following deliverables (in Dutch):

> **D3.2**: Referentie-architectuur gericht op maximaal hergebruik en schaalbaarheid, met de infrastructuur-componenten met mechanismen voor noodzakelijke aanpassingen en met gestandaardiseerde interfaces voor koppeling van deelmarktplaatsen en diensten.
>
> **D3.1**: 'Technologiekaart' met een overzicht van bestaande benaderingen en technische opties.
>
> **D4.3**: Prototype proxy service[1] voor t.b.v. interoperabiliteit met andere dataspaces. Ook komen data, diensten en gebruikers van Dexes beschikbaar op de AMDEX infrastructuur (M24).
>
> **D4.4**: Prototype van de koppeling van de Dexes Datadeel Marktplaats via de Proxy service[2].
>
> **D5.1**: Rapport over minimaal 4 geselecteerde use cases, met een bloemlezing van (experimentele) resultaten en leermomenten.

Deliverable D5.1 is also covered separately.

A higher-level goal of the AMdEX Fieldlab project has been to discover what 'AMdEX' should be for it to realize the goals and vision identified by the project partners. The following possible responses to this question have been discussed and considered:

- A philosophy on how data exchange processes are to be executed and should be governed in order to promote (data) sovereignty, give control to participants and facilitate legal (e.g. GDPR) compliance. An important aspect of the philosophy is that AMdEX dataspaces exist in support of ad hoc consortia of AMdEX participants and are governed by certain 'rules of engagement'.

[1]Rather than using the term 'proxy service', the capabilities referred to are instead realized by the data exchange service and the connectors (see Section 2.4) by which dataspace members connect to the service.

[2]See the previous footnote.

- An advanced ‘authority provider’ and ‘notary’ operating according to EU standards on dataspaces.
- A framework for supporting data exchange ecosystems.
- A community whose members share the philosophy, adhere to the rules of engagement, and find potential consortium partners within the community to initiate exchanges.
- Or simply a community of people and organizations working on data exchange.
- An association whose members establish certain rules of engagement[3], steer the executive organization, and collectively decide on other aspects of governance
- An executive organization that provides several services to the community, of which a few possible suggestions are listed below:
  - Facilitate the community by organizing meetings, networking, support during funding-application processes, management of (research) projects, etc.
  - Facilitate consortia (to be) by supporting the consortium building process and providing templates of legal agreements, data exchange archetypes, and data sharing conditions.
  - Communicate on behalf of the community, e.g., through the organization of meetups and forms of external communication such as press releases and a website.
  - Maintenance of the reference architecture, e.g., through the collection or development of standards, protocols and reference implementations.
  - Operating a ‘trusted infrastructure’ through which dataspace members can connect to certain architecture components hosted by the organization (see 2.2 and see 2.3).
  - Provide certification to organizations to confirm, for example, the compliance of the organization AMdEX rules of engagement and the technical capability to fulfill certain roles, as laid out in this document (see Section 2.2)

An association or executive organization does not yet exist; the full (legal) structure of the planned AMdEX organization is still to be established. The precise governance and operations of the organization are also not yet finalized, not formalized and subject to change. The items listed above reflect the discussions held within the AMdEX Fieldlab project.

1.3 The Approach

The goal of AMdEX is to support data exchange and data processing activities between organizations. AMdEX is neutral with respect to the type of data exchanged between organizations, their motivations for choosing to participate, and the conditions organizations wish to apply. As such, AMdEX dataspaces, in which organizations are connected for the purpose

[3]Separate from laws, consortium agreements and sharing conditions.

of data exchange, are not devoted to a particular domain (e.g. financial or medical) and are not targeted towards specific kinds of users or organizations. The dataspaces are expected to be created for a specific, shared purpose – such as an individual use case – are relatively small in size, and are interoperable – an organization can participate in multiple dataspaces and data can be exchanged between dataspaces (if permitted by sharing conditions).

Although neutral, AMdEX impose governance principles for sharing data on participants in AMdEX, as determined by the members of the AMdEX organization. Moreover, consortia should adhere to applicable national and international regulations. This topic is discussed further in Chapter 3 on 'Policy Administration and Enforcement'.

The AMdEX approach is to raise trust within a consortium by providing certain neutral soft and hard infrastructure. The key novelty is in the governance procedures, consortium agreements, and laws and regulations (soft infrastructure) that are directly integrated into the dataspace (hard infrastructure) that supports the exchange of data between members. Monitoring and control mechanisms within the hard infrastructure are connected to components that determine end enforce the compliance of transactions. Chapters 2 and 3 address the legal and technical challenges of the AMdEX approach.

1.3.1 Principles

The following set of principles underpin the AMdEX approach:

1. AMdEX is neutral with respect to:
    - the goals and intentions of the dataspace members
    - the type of data being exchanged or processed
    - the techniques used to exchange or process data
2. AMdEX does not store, process or consume data assets within dataspaces
    - It follows that AMdEX is not the member of any dataspace
    - AMdEX may process *meta*-data for the purpose of governance and enforcement
3. Data processing within AMdEX dataspaces should proceed in accordance to:
    - the AMdEX Rules of Engagement (established by AMdEX members),
    - the consortium agreement made between members of a dataspace,
    - the conditions established between members of a dataspace as part of an individual transaction (regarding a resource or service), and
    - the applicable laws and regulations

In the future, AMdEX dataspaces may also explicitly consider public values expressed as social policies or codes of conduct. The extent to which AMdEX is compatible with certain values is reflected upon in the next section.

1.3.2 Public values

The founding members of AMdEX have collaborated within the AMdEX Fieldlab project with Waag[4] to investigate how *public values* can be embedded in dataspaces. As such, public values have been an explicit consideration in the design of the architecture. Specifically, a goal for the project has been to support dataspaces that serve public interests – even though AMdEX is intended to be neutral with respect to the goals and intentions of dataspace members (first principle in Section 1.3.1).

An example of a dataspace that serves a public interest is a *datacommon* (see Section 2.1). Together with Waag, the AMdEX partners have investigated datacommons, although a thorough evaluation through a use case has yet to be performed.

The extent to which the architecture enables dataspaces that are aligned with public values is reflected upon throughout this document. Examples of public values are: accessibility, transparency, inclusivity, equity (including the protection of vulnerable and marginalized groups), transparency, public benefit or interests (e.g sustainability, advancement of science/ and healthcare), and the privacy and security of individuals. Below is a list of aspects of the architecture that align well with public values:

- The architecture is designed to facilitate the integration of privacy-enhancing technologies in data exchange process, as illustrated by the application of synthesis data in the UNL use case in Section 4.1.2.
- The UNL use case has also been used to investigate the trade-off between two public values: energy efficiency and privacy through data synthesis [27].
- The architecture is designed to give control to individual members through the specification of policies, which may be individual policies (such as consent, see Section 4.2.3), or social policies and codes of conduct on behalf of a community (see Chapter 3).
- Transparency is promoted by the architecture through the publication of (formal) interpretations of laws and contracts and through the logs of exchange processes held by the Notary components (see Section 2.3). Meta-policies may be needed to determine who is authorized to read which policies.
- The application of open standards promotes collaborations for public benefit.

[4]https://waag.org

Chapter 2

High-Level Architecture

This chapter describes the high-level architecture of AMdEX following a top-down approach: starting from introducing terminology, to describing users, roles, software components and API interfaces. The architecture comprises of components whose development and operationalization are the responsibility of different members of the AMdEX community. The high-level architecture lays-out how the components work together to realize data exchange scenarios within AMdEX dataspaces. In addition, the architecture is a reference for the various members of the AMdEX community. The current version of this chapter describes the architecture at the level of detail obtained by the partners of the AMdEX Fieldlab project. As a deliverable for the AMdEX Fieldlab project this document forms the initial input of a continuous process of revision.

2.1 Scope and Terminology

In this document, an **asset** refers to an **algorithm** or a **data asset**, where a data asset is either a data stream, a dataset or a data point (an individual value such as an integer).

The scope of the architecture includes support for the following processes:

- The **sending** and **receiving** of assets from one organization to another which may involve the transfer of an individual dataset, a subscription to a data-stream, or transfer of an algorithm,
- the **processing** of data by applying an algorithm to one or more input data assets, producing a new data asset that can be shared,
- the procurement of data assets, algorithms, and services through **offers** and **agreements** contingent on conditions categorized as:
  - **pre-conditions**, to be satisfied before an agreement is made, a transaction is performed, or an asset is accessed or used, or
  - **post-conditions** that continue to hold after the asset has been made available and/or has been accessed or used, or that apply to the result of data processing.

The term **data exchange** is used for all of the aforementioned processes. Data exchange processes are often interlinked. For example, an *agreement* can be made that gives a company access to certain analysis results, obtained by the *sharing* of data from the owner of a sensor with a specialized company *processing* the data in order to produce the analysis results.

The described data exchange processes are intended to support various patterns for the exchange of data encountered in practice, referred to as data exchange **archetypes**, such as:

- The sharing of a (possibly anonymized or pseudonymized) dataset by an organization directly or indirectly with another organization, referred to as **data sharing**
- Compute-to-data solutions in which the holder of a data asset executes an algorithm provided by a data consumer on the asset and only shares the result with the data consumer. Such a solution is also referred to as **data visitation**.
- Third-party solutions in which an organization functions as an intermediary to, for example, store datasets or run algorithms on behalf of the consortium.
- Privacy-enhancing or privacy-preserving techniques, possibly provided as a service by a third party, such as:
  - Secure Multi-Party Computation (sMPC) and secret sharing
  - Federated Machine Learning (FML) techniques
  - Differential Privacy (DP)
  - Synthesized data generation

Exchange processes are executed within a **dataspace**, an umbrella term describing several kinds of data exchange collaborations such as:

- **Datatrust**: A dataspace in which a trustee stewards data rights in the interest of a group of beneficiaries often for economic, social or cultural benefit of the group.
- **Datacommon**: A dataspace in which data is pooled and shared as a common resource. Datacommons specifically address power imbalances by democratizing access to and availability of data, and often has a cause for the public/common good. A commons is governed by the collective, including those affected by the exchange of data.
- **Datamarket**: A dataspace in which data is used in a transactional manner and incentives for sharing data are financially driven.

More detail on possible user flows is provided by analyzing the roles of users and various service providers.

2.2 Users and Roles

Organizations wishing to form a consortium and collaborate within a dataspace have to come to an agreement on several aspects of the data exchange, ranging from the business case and

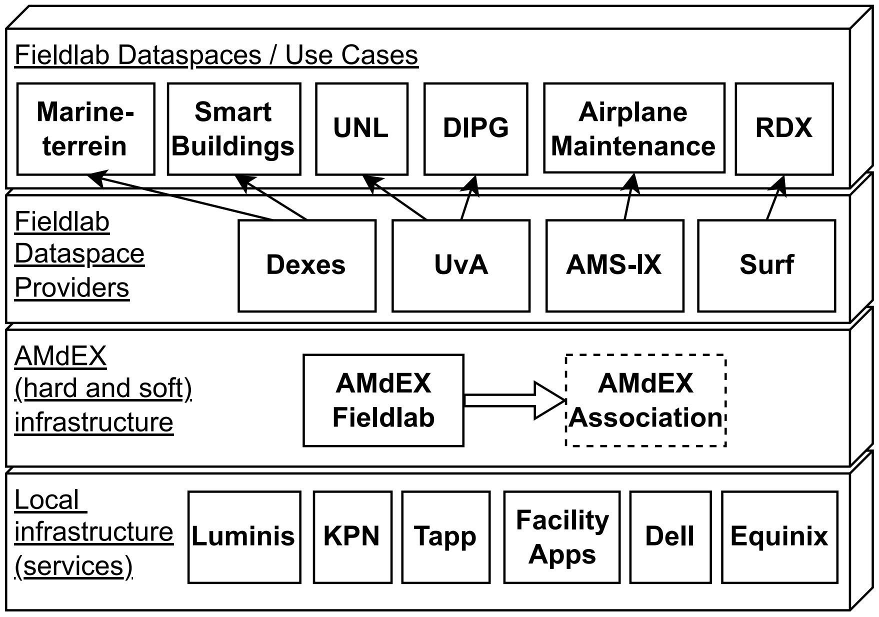


Figure 2.1: Layout of the different entities involved in AMdEX Fieldlab use cases. The AMdEX Fieldlab consortium is to be replaced by a future AMdEX association.

legal requirements to the chosen functionality and operationalization, as identified by the Data Sharing Coalition [5]. Organizations may not be aware of all the technical solutions and the possible service providers that can provide meaningful services to consortium-to-be. AMdEX can support new consortia by helping organizations find the archetype suitable to their use case and help identifying whether any roles are not fulfilled by the members of the consortium. For example, perhaps a compute provider is needed to apply a trusted third-party archetype. In addition, AMdEX will provide templates of legal agreements and archetype specifications (see Section 3.2) that serve to configure the technical infrastructure on which a dataspace runs.

Dataspace providers A dataspace is set up and hosted by a **dataspace provider**. Dataspace providers are essential members of the AMdEX community as they are the face of AMdEX to many users (consortia members). They are responsible for the operationalization of many of the components of the AMdEX architecture, and should abide by the AMdEX guiding principles and rules of engagement. A dataspace provider can service many (possible interconnected) dataspaces for different consortia that are not necessarily bound to a single sector. However, dataspace providers may specialize and focus on specific sectors, as each sector may come with its specific requirements. Figure 2.1 shows the relation between dataspaces (supporting use cases), dataspace providers running dataspaces, service providers, and the AMdEX infrastructure and organization using the AMdEX Fieldlab project as an example. Dataspaces with many common governance aspects may be gathered within an **ecosystem**. Figure 2.2 provides a high-level overview of different roles that interact in a dataspace through the AMdEX infrastructure.

The technical role of AMdEX is to (directly or indirectly) realize (automated) governance

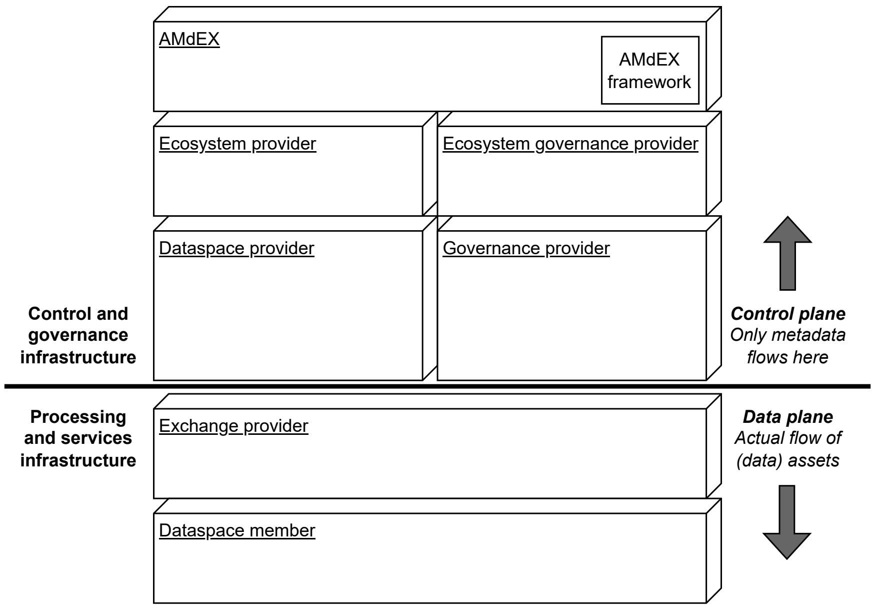


Figure 2.2: The types of entities involved in facilitating dataspaces, reflecting the division between the data plane (bottom), control plane (top) and governance plane (top-right). There is a one-to-many relation between some of the vertical layers of the diagram, e.g., AMdEX supports multiple dataspaces and a dataspace supports multiple members.

in dataspaces. This can be achieved in the following ways:

- AMdEX provides the reference architecture, standards, protocols and reference implementations of components that can be configured according to archetype and agreement templates. Together, these elements form the '**AMdEX framework**'.
- In addition to the above, AMdEX operationalizes certain components in addition to the components operationalized by dataservice providers. Close collaboration and cooperation between AMdEX and dataspace providers is needed in both cases.

The components operationalized by AMdEX, ecosystem and dataspace providers are organized in a **control plane**. A hard distinction is made with the **data plane** in which data exchange processes execute. As explained further in Section 2.3, the data plane and control plane interact to enable the essential features of AMdEX dataspaces, such as asset discovery, access control and auditing. In accordance to AMdEX principles (Section 1.3.1), the control plane does not store or process (data) assets. Only meta-data is processed. Figure 2.5 is a variant of Figure 2.2 in which the components of the AMdEX architecture, discussed in Section 2.3, are mapped onto the entity types.

Dataspace members The members of a consortium, and thereby the users interacting with a dataspace, are classified according to the following roles (visualized in Figure 2.3):

- **Data provider**: An organization making one or more data assets available for data sharing. The data provider is not necessarily the owner of the data asset but may

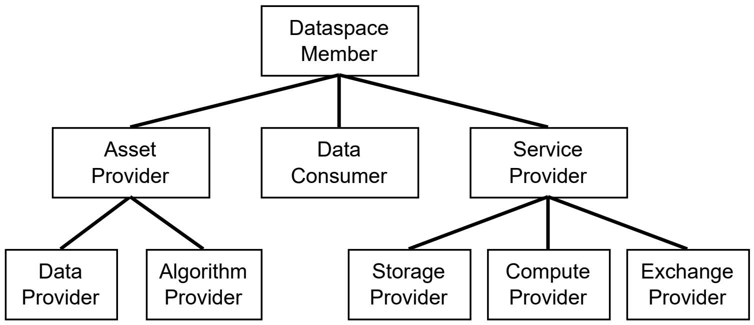


Figure 2.3: A taxonomy of dataspace members.

provide the asset on behalf of the owner. The organization can register (offer) a data asset in a catalog alongside the pre- and post-conditions for accepting the offer and accessing or using the data.

- **Algorithm provider**: An organization making one or more algorithms available for data processing. The algorithm provider is not necessarily the owner of the algorithms. The organization can register (offer) the algorithm in a catalog alongside the pre- and post-conditions for accepting the offer, accessing the algorithm and using the algorithm. Together, data providers and algorithm providers are referred to as **asset providers**.
- **Data consumer**: An organization wishing to access or use data assets made available (directly or indirectly, via processing) by data providers within their dataspace. A data consumer can submit requests by selecting a template corresponding to one of the data exchange archetypes offered by the consortium. The resulting data exchange process may be a simple data access request or may consists of several processing steps.
- **Data service provider**: An organization that offers a particular service to other members of the dataspace, such as (temporary) storage and data processing (i.e., providing 'compute' as a service), and connectivity between members ('exchange').

  An exchange provider may provide connectivity by, for example, running a message-brokering service. An exchange provider may also play a more active role, coordinating the execution of, for example, a federated machine learning protocol.

Note that roles may overlap. For example, a compute service provider can *consume* an algorithm and a dataset in order to apply the algorithm to the dataset and then *provide* the result as a dataset to members of the dataspace. Common overlapping roles are, for example, standardization (algorithm and compute), secure multi-party computation (compute and exchange) and logging/tracing (exchange and data provider[1]).

Dataspace users The roles described above have in common that they are fulfilled by *users* – individuals that interact with the software, possibly on behalf of an organization – and software components. Figure 2.4 shows the users and components associated with dataspace member roles. In the next section, these roles will be concretized as inter-communicating

[1]Viewing the produced logs and traces as a dataset possibly subjected to sharing conditions.

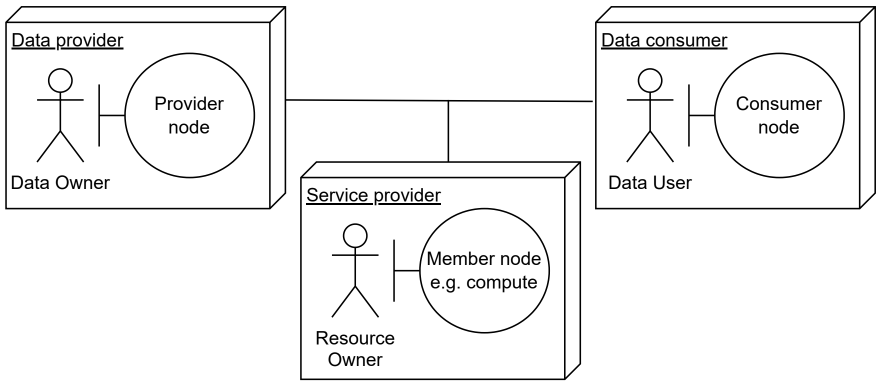


Figure 2.4: A dataspace member role (box) is fulfilled by the combination of a user (stick figure) and a software component (circle).

software components with certain capabilities in the data plane. As we shall see, the users do not only interact with components in the data plane, but also in the control and governance plane.

Connectivity with an AMdEX dataspace itself may be provided as a service. The service provider then acts as a proxy by playing one or more roles in a dataspace on behalf of an organization or individual. For example, a software company may play the role of data provider on behalf of a sensor owner and connect through an exchange provider with other dataspace members.

2.3 Components

An overview of the components of the high-level AMdEX architecture is given in Figure 2.5. The discussion around the components in this section is focused on control and governance infrastructure within the control plane, i.e. without access to actual (data) assets, only meta-data. The architecture does not prescribe that all control and governance takes place in the control plane. Exchange providers typically provide process orchestration, for example to coordinate the execution of a federated machine learning or secure multi-party computation protocol. Dataspace members may also perform their own policy enforcement. For example, a data provider may (double) check conditions to ensure their policies are properly enforced. In general, components running in the data plane can exert a more powerful form of control, more easily and effectively, by being 'closer' to the sources of resources and their access points. The trade-off between operating on actual data or meta-data only is discussed throughout this document, with a bias towards the latter, more challenging case. Note that an implementation for a particular dataspace or use case does not have to implement all the components or can implement multiple components as a single service[2].

The architecture contains components that can be categorized in various ways. Some components are *internal* in that only other components interact with them and users do not. Components may be *optional* in that (valuable) data exchange processes can be executed

[2]Although for various reasons this may not be recommend, such as scalability and adaptability.

without them, if desired. A component may be *modular* in that the component itself consists of smaller sub-components that help to achieve its function. A component may be *distributed* in that it is running across difference sites, requiring a procedure to ensure consistency. A component may be *federated* in that multiple instances of it exist, possibly hosted at different sites, which collaborate to achieve a function together and display a form of hierarchy. A component may be operationalized by (i.e., run within the network domain of) a dataspace member, a dataspace provider, an ecosystem provider, or the AMdEX organization. And components can be categorized based on the type of functionality they implement. In this categorization we refer to the data plane, control plane[3] and governance plane.

Data exchange processes are realized in the data plane with the exchange of assets between, and the execution of algorithms by, members of a consortium. The components of the data plane are described in Section 2.3.1. The control plane consists of components that can help consortium members exert control over the data exchange processes such as by offering and finding available assets, enforcing conditions on transactions and processing steps, and orchestrating the automated execution of data exchange processing steps. The components of the control plane are described in Section 2.3.2. The governance plane covers those components of the control plane that facilitate the direct integration of governance aspects into the data exchange processes. The components of the governance plane are described in Section 2.3.3. In accordance with AMdEX principles (see Section 1.3.1), the components of the control plane and governance plane only process meta-data.

2.3.1 Data plane

The components of the data plane (listed at the end of this subsection) work together to realize **data exchange processes**.

Exchange processes A data exchange process consists of a series of steps producing a data asset that is of interest to a data consumer. Some of these steps may be required in order to satisfy the policies of one or more involved data providers, e.g. anonymization. The steps are performed by instances of components of the data plane operationalized by different consortium members, depending on the role the members play in the consortium. Conceptually, every dataspace member is responsible for one component referred to as a **member node**. In reality, member node instances are expected to have their own (potentially complex) architecture, distributed across the local network of the organization that owns the component. Depending on the roles consortium members play, the member node of a consortium member is expected to perform certain steps, but not necessarily certain others. For this reason, the member node component describes certain shared functionality and is extended by a **consumer node**, a **provider node**, and a **compute node** to provide functionality specialized to certain roles. Just as a consortium member can play multiple roles, a member node can instantiate multiple types of member nodes. For example, a compute provider (role) is expected to be fulfilled by a member node that simultaneously instantiates a consumer, provider and compute node. Table 2.1 displays the processing steps performed

[3]`https://www.rfc-editor.org/in-notes/rfc3746.txt`

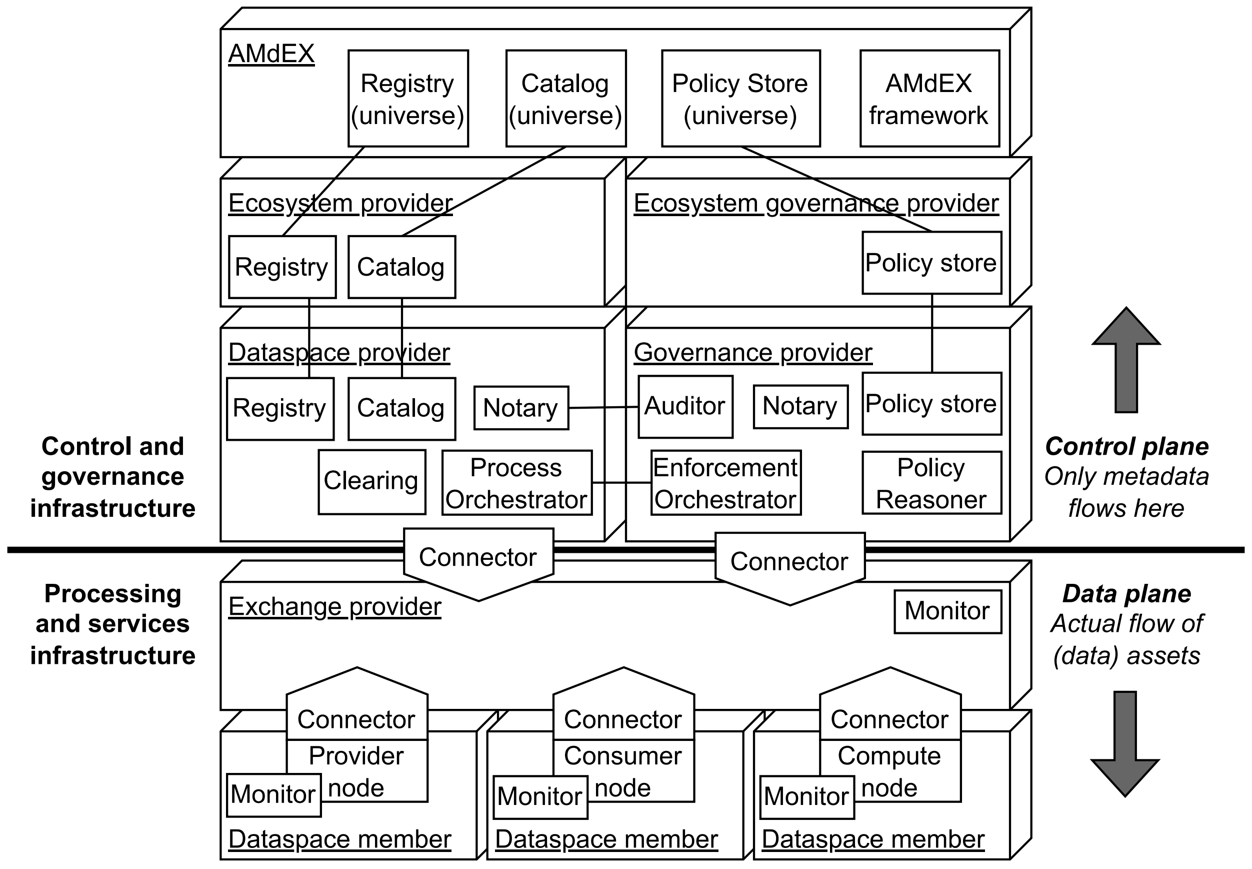


Figure 2.5: A categorization of the AMdEX architecture components within a data plane, control plane and governance plane (right-hand side of control plane). The components are also organized according to which (type of) entity runs the component.

by the different types of member nodes. At a minimum, a member node provides certain shared functionality that is required by all dataspace members, such as identification.

- **Member node**: This component connects dataspace users to other members/users and connects to components in the control plane. Specialized (extensions) of the node are listed below. This component captures shared functionality such as providing an identity, discovery, and connection to the control plane.

- **Consumer node**: An extension of a member node that initiates data sharing and processing requests and may receive data sets or acquire access to data streams.

- **Provider node**: An extension of a member node that resolves URIs for assets (datasets, streams, or algorithms) and registers assets in the catalog, subject to sharing and processing conditions.

- **Compute node**: An extension of a member node that performs compute steps, possibly on instruction, receives assets (same capabilities as a consumer node), and sends datasets or provides access to a datastream (same capabilities as a provider node).

Step	**Description**	**Inputs**	**Effect**	**Node**
init	initiate an exchange request	template, assets, role assignments	clearing starts	consumer
receive	receive an asset	asset, sender ID	stream access or asset download from a remote source	consumer/ compute
send	send an asset	asset, receiver ID	access to a local stream or asset is given	provider/ compute
compute	run an algorithm on some input data	algorithm, input data assets	a data asset is produced locally	compute

Table 2.1: The processing steps that can be performed by the different types of member nodes of the data plane.

- **Monitor**: A monitor is a module of a member node that provides information about the execution of processing steps to the Process Orchestrator and Enforcement Orchestrator.

2.3.2 Control plane

The control plane contains components (listed at the end of this subsection) for onboarding members, registering and findings assets, submitting exchange requests and executing transactions.

Member and asset registration The registry and catalog components (discussed below) are federated components of which instances are hosted at multiple **registration levels**. The registration levels form a hierarchical tree structure, depicted in Figure 2.6. Motivated by AMdEX Fieldlab use cases, three registration levels have been identified: universal, ecosystem, and dataspace. The hierarchy can be extended by adding additional levels. The levels form 1-to-many relations. There is one 'universe' that can support multiple ecosystems which in turn can support multiple dataspaces.

As federated components, instances of the registry and catalog will be operationalized by different organizations depending on registration levels. Any components at universal level are expected to be operationalized by AMdEX, or a party on behalf of AMdEX, and the components at dataspace/ecosystem level are operationalized by dataspace/ecosystem providers. A dataspace member may decide to host its own catalog – effectively adding a fourth layer to the hierarchy – possibly simplifying integration in a dataspace and facilitating connections to non-AMdEX dataspaces or catalogs.

From request to transaction An **exchange request** specifies which data exchange process a data user would like to initiate[4] with the request (see Section 2.3.1 and Table 2.1).

[4]In some cases it may be desirable to have exchange processes initiated automatically in response to some event, e.g. at the end of every month or when a new version of an asset is available. In these cases another node than a consumer node can be imagined as initiating the request.

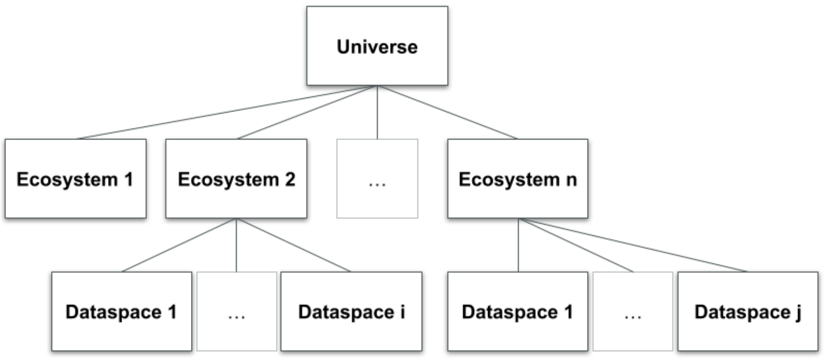


Figure 2.6: Registration levels at which members and assets can be registered.

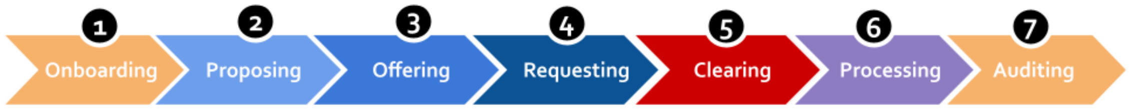


Figure 2.7: The lifetime of an exchange process as it goes through various stages.

The lifetime of a request is visualized in Figure 2.7. An exchange request comprises of an exchange template and a collection of assets and role assignments that instantiate the exchange template (REQUESTING, Figure 2.7). The assets and resources involved in the request have been registered in one or more of the applicable catalogs and offered by registering usage conditions (together referred to as an offer) in the policy store (OFFERING). Different offers can be registered in the policy store as being applicable to the same catalog entry. Before assets can be offered and requested, all involved members must be registered in the registry of the dataspace (ONBOARDING). A data user can go through an optional process in which a proposal is negotiated with other consortium members (PROPOSING). An example of this mostly social process is given in Section 4.2. The role assignments of the request determine how the abstract roles of the template are concretized by member nodes (see also Section 3.3), resulting in a sequence of processing steps, starting with an 'init' and ending with a 'receive' of the same consumer node.

Before a request is turned into an **agreement**, the owners of the assets may have to give permission for the usage of their asset(s) within the request. Similarly, the owners of compute nodes (compute providers) may have to give permission for the usage of their infrastructure within the request. These permissions may be manually provided when needed or can be derived from policies. Additionally, the request needs to be checked for compliance with pre-conditions, the consortium agreement and other higher-level policies. Some of the pre-conditions may require certain clearing steps to be fulfilled, e.g., the requesting data consumer may have to pay for the upcoming transaction. The acquisition of manual permissions can also be seen as clearing steps. As such, a request is transformed into an agreement by a joint accepting, clearing and checking procedure (CLEARING). The agreement is registered in the policy store together with data confirming the (successful) execution of the clearing steps.

The template of a request describes the data exchange process as a sequence of processing steps (from Table 2.1) assigned to roles. This sequence of processing steps is referred to as a

plan and forms an integral part of the agreement. An agreement is executed by executing the processing steps of the plan computed for the request (PROCESSING). These steps can be executed manually by the members involved, or automatically through orchestration. At each step, the processing of the step will be checked (again) for compliance against the latest policy and with the latest policy information. This can be done by the relevant components in the control plane or the data plane (see Section 2.3.3 and Chapter 3 for further discussion). When successful, the execution results in a completed transaction and a number of log files that together form a dossier used for ex-post (after the fact) compliance checks (AUDITING). The completed transaction is registered in the policy store, together with a reference to the dossier. When unsuccessful, e.g., because certain steps were not performed or dynamic policy enforcement prevented a non-compliant action (see also Section 3.4), a dossier is still produced with all relevant logging information needed for auditing. The auditing process also involves the checking of post-conditions, possibly considering certain external information in addition to the dossier. This may cause one or more parties to trigger a process in which the compliance of the transaction is disputed. Any disputes to the transaction are also registered in the policy store.

- **Registry**: The registry is a federated component of which instances keep track of *registered members* at a particular registration level. A registration is the result of an on-boarding process. Members are registered as performing one or more roles at a certain registration level. The higher level registries can be used to find new members for joining a dataspace based on the role they play (services they provide).

- **Catalog**: The catalog is a federated component of which instances keep track of *registered assets* at a particular registration level. Assets are registered with meta-data, exchange conditions and applicable exchange archetypes. The conditions needs to be compliant with the consortium agreement, the exchange archetypes and policies at higher policy levels (see Section 3.2). Members can register and find assets registered at their level and higher levels. Unless otherwise stated when offered, visibility of offered resources travels 'downward' in the registration hierarchy, e.g. resources offered at the ecosystem level can be found in the Catalogs of dataspaces within that ecosystem. Assets can be registered multiple times in different catalog instances and under different conditions. A catalog at the dataspace level can receive exchange requests for one or more assets that are to be exchanged according to an exchange template provided alongside the request. Requests are processed by the Clearing component.

- **Clearing**: This component turns exchange requests to agreements by ensuring and checking that all pre-conditions of the request have been satisfied, i.e., that all manual permissions for executing the requests' plan have been given, that all sharing conditions have been met and that the steps of the plan would not violate the consortium agreement, laws and regulations and other high-level policies. The result of clearing is an agreement on a plan that is ready for execution. The clearing component is modular with several sub-components/modules fulfilling specific functions of the clearing process such as collecting manual permissions and processing payments.

- **Process Orchestrator**: This component orchestrates data exchange processes (when needed) by sending out instructions for executing processing steps to member nodes and receives feedback from member nodes on successfully executed processing steps. The orchestrator drives and tracks the step-by-step execution of a plan. Execution may succeed, resulting in a (completed) transaction, or may fail, in which case logging information is (also) made available. In the data plane, a data exchange provider may provider similar functionality (see Section 2.4.2).

- **Process Notary**: This internal (and possibly distributed) component serves as a ledger to keep track of the status changes to exchange requests, agreements and transactions throughout the entire life-cycle of a request. The ledger retains a log of status updates based on input from the Catalog, Clearing and the Process Orchestrator components.

2.3.3 Governance plane

The governance components (listed at the end of this subsection) are responsible for the control processes that directly relate to the governing of the dataspace, including compliance with laws, regulations, ecosystem rules, consortium agreements and data exchange conditions. The AMdEX approach to automating compliance considers both ex-ante ('before the fact') and ex-post ('after the fact') enforcement, observing the social reality in which not all violations can be prevented. Ex-post processes such as auditing based on logs and other, external information are necessary for various reasons. There may not be a consensus between consortium members on the precise legal interpretation that is to be applied, on the events that occurred or the qualification given to these events. Moreover, new information may come to light that could potentially change the assessment of the compliance of a transaction. This is especially relevant to (post-)conditions that can only be checked outside of the control of the dataspace, e.g., when a consumed data asset is made available outside the consortium against the will of the consortium. Therefore, compliance checks occur at various stages in the lifetime of a data exchange process.

1. During OFFERING: the exchange conditions assigned to an asset need to be consistent with the consortium agreement and higher-level policies. For example, the consortium may disallow payment requirements and privacy regulations demand a legal basis for processing (i.e. lawful processing based on consent or a legal obligation).

2. During CLEARING: the exchange (pre-)conditions are checked and, where needed, processes are started to satisfy the pre-conditions. The plan produced by instantiating an exchange template is checked for compliance.

3. During PROCESSING: individual processing steps are checked for compliance. This happens both before and after they are executed:

    - **before**: a member node can request permission from the Enforcement Orchestrator, or a Process Orchestrator checks permission before sending an instruction
    - **after**: a Process Orchestrator was informed that a processing step was executed and checks this event for compliance

In both cases, the Process Orchestrator requests compliance decisions from the Enforcement Orchestrator and all communication is logged, including the observation of any violations. Local policy reasoning by a member is possible as well, e.g. in order to verify the compliance of the step against local policies or to verify whether any obligations are assigned to the member. The possibility of checking compliance *during* the execution of a processing step is discussed in Section 3.4.

4. During AUDITING: a (fully or partially executed) transaction and the logs produced during the execution of the transaction are assessed. At present this is still a manual process; possibilities for (partial) automation are to be investigated. Manual assessment is needed for many types of post-conditions encountered in practice. Observations that may affect compliance with post-conditions can be (manually) brought into this process, e.g. the observation that a data asset is used for a differnet purpose than intended. This topic is discussed further in Chapter 3 and some of the use case descriptions in Chapter 4.

The role of monitoring Compliance is assessed through the submission of policy queries to a Policy Reasoner. In order to answer a query, the Policy Reasoner needs a policy set and policy information as input. The policy set is the result of composing those policy sets, registered at the policy store, that are deemed relevant to an exchange process. Which policy sets are deemed relevant is determined by the consortium, the owners of assets, and the providers of services, used in the request. The resulting composition may be inconsistent in that reasoning with the composition is impossible or will not yield unambiguous results. Inconsistency is to be checked when policy sets are registered in the policy store.

Policy information is all (other) information needed to assess compliance, such as information about the assets and members involved in a request. Not all such information can be (made) available, further explaining the need for ex-post enforcement processes. Some information is to be provided dynamically, within the lifetime of an exchange process. **Monitors** are components of the data plane that provide (dynamic) information about processing steps as they relate to a particular data exchange process. For example, a monitor can provide the amount of records produced by a compute step, or the K-anonymity achieved by data synthesis. Informing the process orchestrator of executed processing steps can also be seen as monitoring. As such, monitoring is crucial for governing exchange processes. However, as components of the data plane, dataspace members are in control over the monitoring information provided. Manual governance and enforcement practices are required to ensure members make the necessary information available, e.g., through certification. Additional information about monitoring is provided in Section 3.4.

- **Policy Store**: The Policy Store is a federated component of which instances keep track of registered policy sets at a particular registration level. Policy sets are registered with meta-data such as links to normative sources (e.g., legal sources) and version numbers. Depending on the registration level, different authorities can register policies in the Policy Store (see Section 3.2). Members are able to instantiate templates of data sharing and data processing conditions to associate the resulting conditions with the assets they register in the Catalog. A policy store provides both an API- and user-interface for registering, finding and selecting policy sets.

- **Policy Reasoner**: This internal component receives policy queries to determine the consistency of policies (validation) and the compliance of plans (ex-ante enforcement), processing steps (dynamic enforcement), and transactions (ex-post enforcement). In the latter three cases, a policy query consists of a sequence of processing steps and is sent by the Clearing component, Process Orchestrator and Auditor components respectively through the Enforcement Orchestrator. Validation queries are sent by the Catalog when assets with conditions are registered and by the Policy Store when policy sets are registered. The response to a query is based on a policy set received from the policy store and additional policy information provided alongside the request (e.g., meta-data about members, assets, etc., analogous to policy information in attribute-based access control).
- **Enforcement Orchestrator**: This internal component mediates between the Policy Reasoner and the Clearing, Process Orchestrator and Auditing components to ensure the reasoner is given the required policy information to answer a particular policy query. The Policy Reasoner informs the Enforcement Orchestrator of any missing information. In response, the Enforcement Orchestrator will send information requests to the components capable of producing the information. For example, the Registry can provide the attributes of a dataspace member and Clearing can obtain manual permissions.
- **Enforcement Notary**: This internal (and possibly distributed) component serves as a ledger to keep track of the policy queries to the Policy Reasoner and policy decisions by the Policy Reasoner related to a specific transaction. The ledger retains a references to versions of policy sets, provided policy information, and the decision made by the Policy Reasoner.
- **Auditor**: This component provides a user-interface to the transaction logs produced throughout the lifetime of data exchange requests and stored in the process and enforcement notaries. These logs necessarily only contain meta-data. To audit the compliance of transactions of individual processing steps, monitors are required at the nodes in the data plane, or it should be possible to bring in outside information. The logs produced by monitors in the data plane are necessarily distributed and may themselves be subjected to access control. A comprehensive auditing process thus requires a solution within the consortium. The integration of control plane and data plane auditing requires further investigation.

2.4 Connecting to AMdEX

For the average user that shares data or uses data, AMdEX will be a nearly invisible component. Service providers, instead, offer services to data owners and data users, helping data owners and users to share data. The data owners and users can then see AMdEX merely as a means to gain confidence that the data is being shared in a trustworthy way. Whereas users interact with service providers, service providers interact with AMdEX in support of

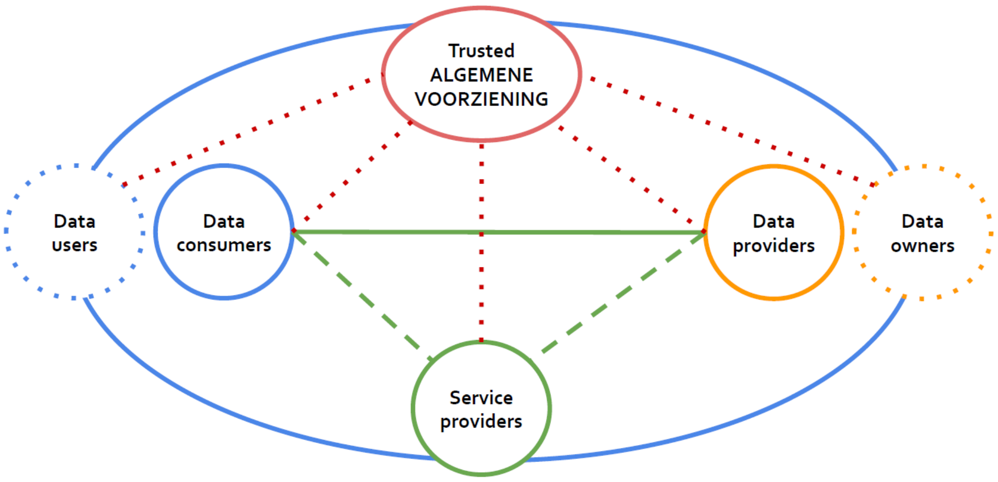


Figure 2.8: A conceptual scheme of the roles that connect to AMdEX

their users. This section provides additional details on the realization of AMdEX dataspace or ecosystem, focusing in particular on the connections between participants.

Participants can roughly be divided in the following three categories (roles in Section 2.2) in a data sharing scheme (see also Figure 2.8).

- **Data providers**: parties that collect data or handle data for entitled parties like data owners.
- **Data consumers**: parties that provide services and applications to data users to create value based on available data.
- **Service providers**: parties that support the process of sharing data.

To connect to AMdEX, the above mentioned providers will need to make actual connections and interfaces with AMdEX provided components. Do note that every data provider, data consumer and service provider connects to AMdEX, directly or indirectly, in order to facilitate a distributed, safe and trusted data exchange.

The roles of data owner and data user, as entitled parties, are important from a legal viewpoint in particular, but the do not necessarily need to make any technical connection to AMdEX when they are serviced by the service providers.

2.4.1 Dataspaces and ecosystems

A dataspace is a virtual environment where data providers and data consumers connect, and where specific rules are applicable, e.g. as captured in a consortium agreement. A dataspace can be a closed environment where only specific parties are connected as members and perform transactions. A dataspace can be a public space, in the sense that it is open for anyone to offer assets accessible to other parties on the dataspace. Figure 2.9 shows how an organization (or individual) can be a member of multiple dataspaces at once, potentially playing a different role in each. In both cases, however, a participant needs to connect to

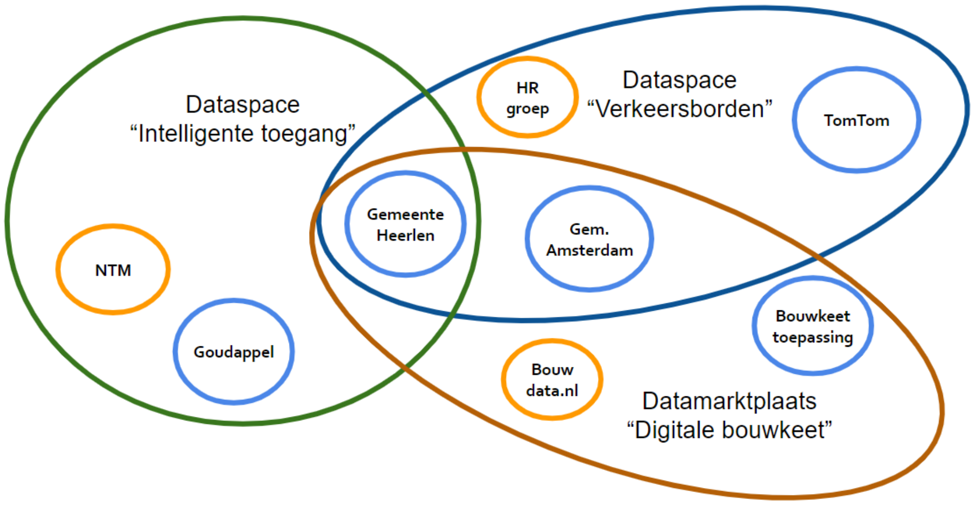


Figure 2.9: Organizations and individuals can play different roles in different dataspaces.

infrastructure in order to connect to other participants, either directly – through their own member node – or indirectly – as a user of a provider offering the connection as a service.

In the Netherlands, parties in a specific economical or social domain (sector), organize themselves in an ecosystem. When data providers and data consumers become a member of an ecosystem, they connect to one or more dataspaces within the ecosystem. An ecosystem may form a potentially very large and complex landscape of providers and consumers of various assets and services, each with their specific expertise and (partial) knowledge on data exchange processes. For this reason, parties may offer 'connecting to AMdEX' or 'running shared AMdEX services' as a service to ecosystems and their members. The mentioned shared services ('Algemene Voorzieningen' in Figure 2.11) are, among possible others, some or all of the control and governance components discussed in the previous section. In summary, both individuals, dataspaces and ecosystems require service providers.

2.4.2 Service providers

Several types of service providers may be involved in running a dataspace or ecosystem. Some service providers are members of dataspaces, offering services as data transformation, data analysis, and compute or storage resources (see Figure 2.3 and Figure 2.10). For example, a compute provider may efficiently process data streams providing large quantities of data and forward aggregated results to a storage provider for publication. Other service providers play a role in the infrastructure of a dataspace, such as dataspace and ecosystem providers hosting (parts of) the shared service (in the control plane, see Figure 2.5). Data exchange providers (in the data plane, see Figure 2.5) assist data providers, data consumers and member service providers with their connection to other members of dataspaces and ecosystems. Crucially, data exchange providers also connect to the dataspace and ecosystem providers running shared services, helping to ensure that members follow the rules and conditions of the ecosystem and the dataspaces (through policies, as discussed further in Chapter 3). Note that the service providers themselves must adhere to laws and governance

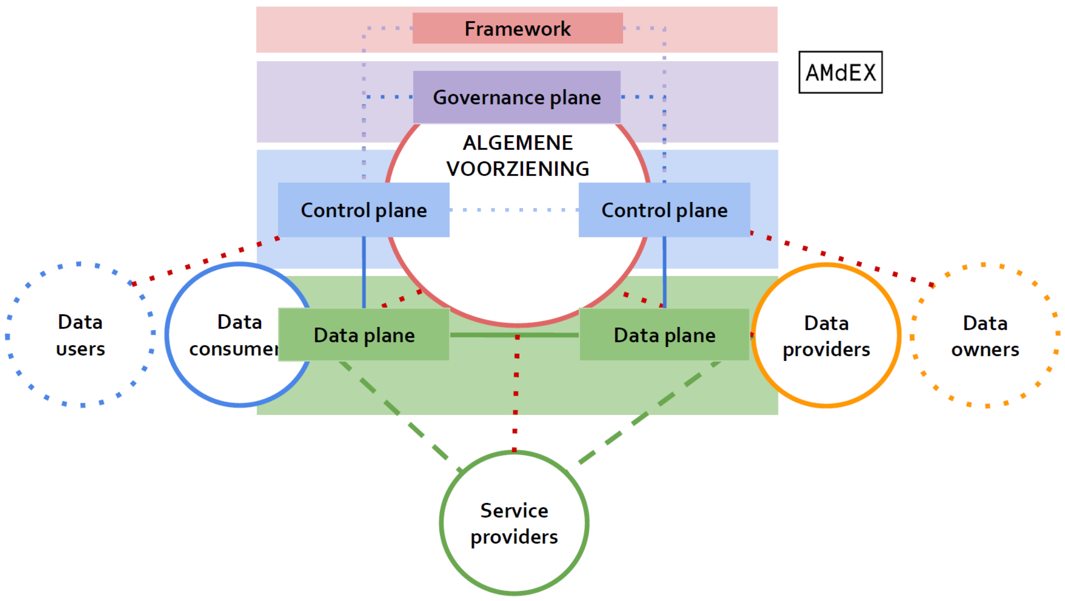


Figure 2.10: Member service providers offer support in the exchange process between providers and consumers in the data plane.

rules in order to guarantee the sovereignty of participants and the privacy of data subjects. Therefore, service providers need to be audited and certified.

Some data exchange providers may provide primarily communication as a service. For example, a telecommunication company offering fast network communication to connect sensors producing large data streams. Other data exchange providers may play additional roles on top of the communication, e.g. to orchestrate the execution of multi-party computation protocols such as federated machine learning, differential privacy or secret sharing. In these cases, data exchange providers are also hosting process orchestration and may provide compute power as a service, i.e. are also playing the role of dataspace provider, data exchange provider and compute provider simultaneously.

2.4.3 Connecting to AMdEX services

The previous subsections describe various connections between participants of dataspaces with different roles. For these connections, the AMdEX architecture describes two types of **connectors**:

- **Data plane connector** or **exchange connector**: a connector between member nodes directly or via a data exchange provider, and
- **Control plane connector** or **trust connector**: a connector between the shared services and a data exchange provider or member nodes directly.

Connectors are involved in most[5] actions, transactions and events transpiring in a dataspace. A connector provides connections on several interfaces (APIs), some of which are mandatory

[5]Note that some user actions may be directly with a component in the control plane, as indicated by the dashed line in Figure 2.10, such as a user registering a dataset in a Catalog.

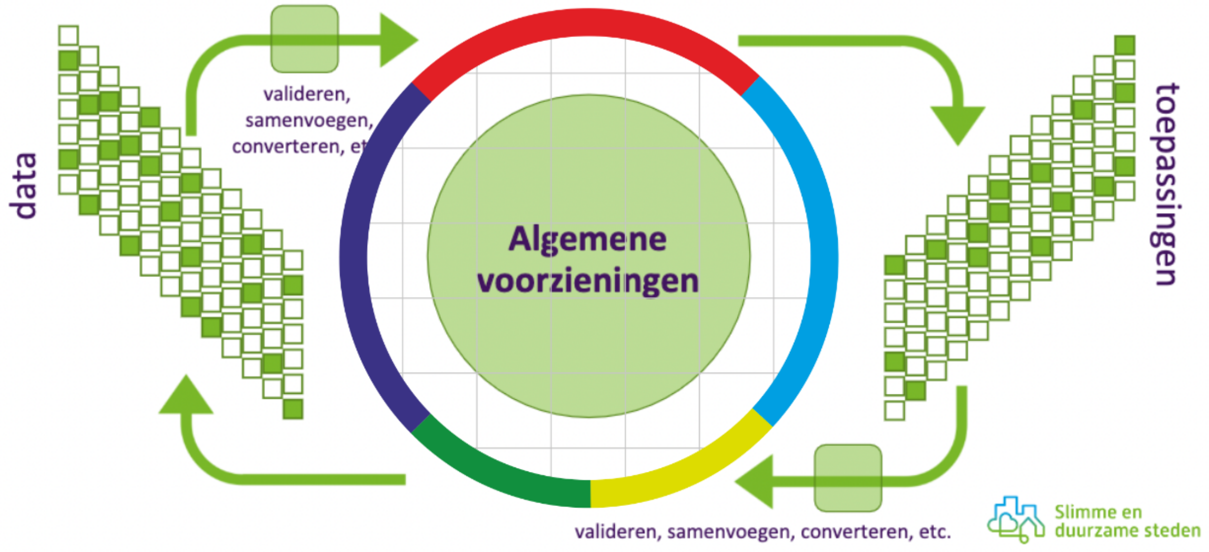


Figure 2.11: The ecosystem approach to data exchange where data assets and applications (‘toepassingen’) can be exchanged through the application of shared services (‘Algemene voorzieningen’) provided as part of the AMdEX framework.

or optional to implement. The connectors are the points of contact between the (local) infrastructure of the participants and therefore require implementation or integration within the local infrastructure. Through connectors, members communicate to other members and the shared services of the control (and governance) plane either directly or via a *data exchange provider*. In the latter case, the data exchange provider implements the interfaces (APIs) as if a member node and the host of the shared services, relaying messages on top of any additional message it may send (e.g. for orchestration or enforcement purposes).

Important motivations for including ‘connectors’ in the architecture are to achieve separation of concern[6] and to offer a standardized[7] way of accessing the Registry for identification and authentication, and the enforcement services (Reasoner and notaries through the Enforcement Orchestrator) for authorization, auditing, and accountability. These two aspects of the control plane connector are captured by the following mandatory interfaces:

1. The **trust API**. The trust API connects the Registry and provides identification services, keeping track of all parties that are a member of an ecosystem or dataspace. The API provides token-based authentication to secure further communication between parties. Each time an AMdEX service or a service provided by data providers, data consumers and service providers is used, a call will be made to the trust registry to check whether parties are valid members and to provide tokens for further transaction.

2. The **authorization API**. The authorization API provides access to the governance and enforcement services described in Section 2.3.3 and Chapter 3. As such, the

[6]In [31], it is discussed how the ‘sidecar’ design pattern enables connector implementations to be separated from the message-passage mechanism of the implementation of the data exchange provider.

[7]Although standardization has not yet been achieved.

API is used for policy and compliance related requests at different moments of the lifetime of an exchange process: OFFERING, CLEARING, PROCESSING and AUDITING (see Figure 2.7 and Section 2.3.3). Most importantly, the authorization API gives authorization for individual processing steps such as data access when permitted by policies. Authorization requests provide information that may be collected by the Process Notary and Enforcement Notary for accountability purposes.

The following additional APIs may be implemented:

3. The **catalog API**. This API is used for publishing assets and offers in the Catalog. Asset and service providers can publish their assets and services so they are visible to members in the ecosystem and dataspaces. Each assets has one or more links to 'offer' policies that determine the pre- and post-conditions for the offered asset or service.

4. The **policy store API**. In the Policy Store the offers and agreements are stored that can be used for authorizations. Asset and service providers use this API to publish 'offers' and 'agreements' (see Section 2.3.2) in the Policy Store so these are part of the collection of policies that are enforced on an ecosystem and/or dataspace. Users given the right authority may publish policies that interpret laws, regulations and consortium agreements (see Section 3.2).

5. The **accountability API**. For the purpose of accountability, parties should keep track of logging information in order to provide audit trails of events and transactions that take place on their infrastructure as part of data exchange process steps. This information is provided to the Process Notary through the accountability API together with (other) monitoring information. The information held by both notaries form the audit trails for complete data exchange processes. Note that this is a controversial feature as some of the logging information may be sensitive (e.g. reveal company secrets). Further research is required to see whether these logs can be made available only under certain conditions, reusing the existing components of the architecture for enforcement.

6. The **auditor API**. The Auditor API gives access to the logs held by the Process and Enforcement Notaries. Only users with special authority should receive authorization to access these logs. This information can be used for manual auditing or (partially) automatic auditing. Future research is required for understanding the extent to which auditing may be automated based on the available logs and its use for accountability.

Table 2.2 gives an overview of the APIs of the control plane connector together with relevant examples of implementations and information models.

To improve adaption, we strive to provide a "standard data connector" that provides basic API implementations to the shared services and can be easily extended and specialized for provision by data exchange, dataspace or ecosystem providers. This approach is inspired by various European initiatives in which data connectors are being developed (e.g. for IDSA [14]). Ideally, AMdEX supports these data connectors and, vice versa, are the AMdEX connectors supported by other data exchange initiatives.

Connector API	**Example standard**	**Example implementation**
Trust API	OpenAuth	iShare
Authorization API	XACML, eFLINT	AMdEX policy reasoner
Catalog API	DCAT	Dexes Catalog API
Policy store API	ODRL, eFLINT	Keyrock policy enf., Karels 2023 [21]
Accountability API	–	–
Auditor API	–	–

Table 2.2: An overview of the conconnector APIs with examples of comparable implementations and relevant languages, standards and information models.

2.4.4 Dataspace interoperability

Interoperability defines how data and 'trust' can be exchanged between various parties and components. For AMdEX, the separation of the actual data exchange and the control and governance process that supports the exchange are essential (see the AMdEX principles in Section 1.3.1). AMdEX does not handle data (only meta-data) and, as such, AMdEX developments on the control and governance processes (as is reflected in the focus of this document). Instead, dataspace (and ecosystem) members handle the actual data exchange. In other words, for AMdEX interoperability is primarily about how the exchange of "trust" between parties, dataspaces, and ecosystems. Trust is concretized through the preservation of sovereignty (ownership, authority) and control (the ability have an influence) as assets and services are made available across dataspaces and ecosystems. Technically, the interconnection is realized by the federated nature of the Registry, Catalog and Policy Store components, as described in Section 2.3 and suggested in Figure 2.5. Chapter 3 gives further detail about the administration and enforcement of policies, providing a form of sovereignty and control to dataspace members.

In addition, interoperability is also about the exchange of data and trust with dataspaces that are outside of the scope of AMdEX, e.g. residing in other European Union member states. For example, a data provider that is part of an ecosystem in the Netherlands should be able to give access to its data to a data consumer that is connected to a comparable ecosystem in Denmark.

The following design decisions have been made with a special consideration on interoperability, both within and outside of AMdEX.

1. AMdEX uses open standards for connectors and APIs,
2. AMdEX connectors should incorporate external connectors based on open standards,
3. AMdEX embraces iShare as the European standard for identification and authentication to enable inter- and intra-dataspace interoperability.

Chapter 3

Policy Administration and Enforcement

This chapter zooms in on the application of 'policies' in the AMdEX architecture and the AMdEX Fieldlab project, and answers the following questions:

- What does AMdEX consider as 'policies'? What types of policies are in scope?
- Who is responsible for specifying (and selecting) policies? And at which stages of setting up, maintaining or interacting with a dataspace should this happen?
- What information should policy specifications contain? And in what language can policies with the required information be formulated?
- How are policies enforced within the architecture? What are different approaches for doing so? And how do these approaches differ in terms of trust required and control offered to stakeholders?
- How can policy reasoning be made accountable and auditable?

The answers to these questions provided in this chapter are the result of a coming together of a multiple lines of theoretical research conducted at the University of Amsterdam (jointly with TNO) and the practical objectives of the AMdEX Fieldlab project. This 'coming together' has not been completed and future research and experiments are needed to answer remaining questions. This chapter reflects the state of the discussion towards the end of the AMdEX Fieldlab project.

3.1 Scope

In the context of data exchange, the term 'policy' can simultaneously refer to: (a) the technical term encountered in the fields of distributed computing, databases, networking, and cloud computing, and to (b) the social construct with which organizations determine how members of the organization are expected to behave in certain situations, e.g. organizational policies and governmental policies. In the technical case, the term policy is referring to a

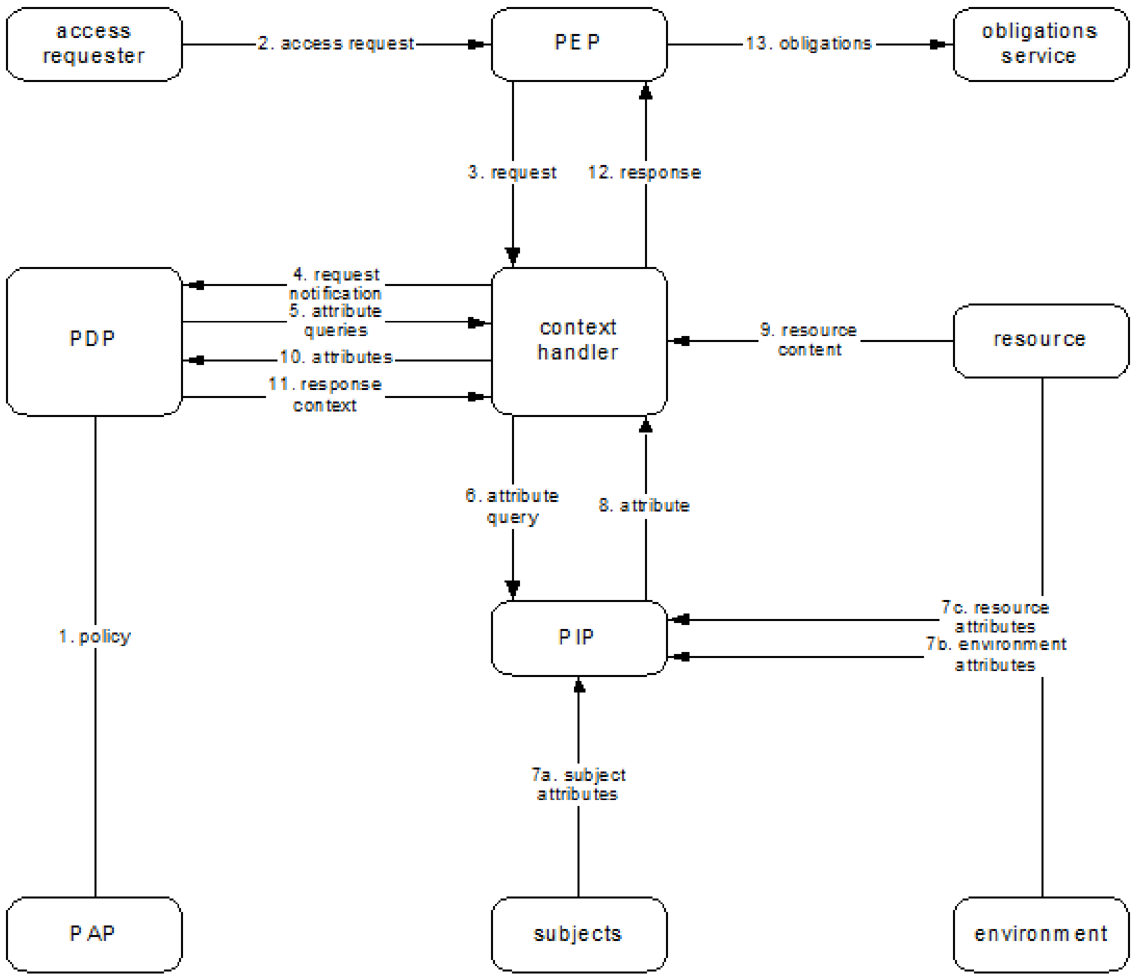


Figure 3.1: The architecture for policy enforcement proposed by the XACML standard [24].

mechanism by which an infrastructure is controlled, most often dynamically, and policies can be seen as configuration parameters set by actors that are not otherwise capable of controlling the infrastructure. Typical examples are routing policies, access control policies, resource management policies, and firewall rules. These kinds of policies tend to be fine-grained, domain-specific and not widely applicable in other contexts. On the other hand, the social kind of policies tend to be high-level and regulate the behaviors of human actors rather than infrastructure components (but they may also be domain-specific). In both cases, policies are specified in separation from the executive processes in order to maximize adaptability. One of the goals of AMdEX is to bring the two meanings of the term 'policy' together by explicitly connecting social policies with technical policies that influence behavior within a technical infrastructure.

A similar motivation is behind the XACML architecture for (attribute-based) access control [24] shown in Figure 3.1. The architecture explicitly separates the technical means of *enforcing* (technical) policies from the *administration* of (organizational) policies set by an organization to govern the access to its resources. As a result, policies can be easily adapted without having to modify the technical infrastructure. Conversely, new enforcement mechanisms can be introduced to control access to resources without having to modify the way policies are expressed. Policies can also be migrated to (reused across) other infrastructures without (much) modification.

The AMdEX architecture follows this approach, with a Policy Store playing the role of XACML's *policy administration point* (PAP), a Policy Reasoner playing the role of *policy decision point* (PDP), an Enforcement Orchestrator playing the role of Context Handler, and both Process Orchestrators, Clearing modules, and Member Nodes (potentially) playing the role of *policy enforcement points* (PEPs). Alternatively, or in addition, a member may also decide to provide their own policy enforcement, for example a data provider hosting a Policy Reasoner (PDP) to verify access requests to their assets. Besides administering policies as an PAP, the Policy Store should also provide an API to extract policy sets that are used by the Policy Reasoner to make decisions on particular cases. For these purposes, policy sets should be publishable in the sense that they cannot be altered, although possible superseded by newer, updated versions, and remain available.

The conversion of organizational policies to access control policies is typically performed by system administrators in close consultation with privacy or security officers, as often seen in the healthcare domain. This process is typically not formalized and the connection between the two types of policies may not be explicated. Moreover, besides organizational policies, access to resources is also regulated by national and international laws and regulations, and temporary agreements can be made between parties that require (temporary) adaptations to the policies, e.g., in the form of delegations, consent for processing or data sharing agreements (contracts). The XACML policy language is comparatively low-level and specific in nature, making it difficult or impossible to express the higher-level, social policies directly.

At the University of Amsterdam, jointly with TNO, experiments have been conducted that demonstrate how 'social policies' can be formalized in machine-readable fashion. The taken approach is based on the fundamental normative concepts of the legal scholar Hohfeld [18]. The FLINT language is based on Hohfeldian concepts and can be used to specify normative documents in a machine readable fashion [8, 7]. The eFLINT language [3] is an executable variant that enables implementation as a policy decision point (Policy Reasoner). The eFLINT language has been developed with several important design criteria in mind that improve policy administration:

- **Expressiveness**. The language makes it possible to express both technical policies and social/organizational policies within the same language. Moreover, the two can be explicitly connected such that system-level decisions are based on higher-level reasoning on the normative positions of actors in the system. This property has been demonstrated 'in the lab' using the DIPG use case [2] further discussed in Section 4.2.
- **Modularity**. Policies are specified as the composition of smaller policy fragments, maximizing reuse and extensibility of policies. Extensions and compositions can also be made dynamically, i.e., as the system is running.
- **Accountability**. The language is rule-based and uses a form of logic programming (classical AI) that makes it possible to explain policy decisions. Moreover, the language

keeps track of executed actions, observed events, and their effects, making it possible to trace the input used in policy decisions.

Within this chapter, the term 'policy' is to be interpreted in the broader sense, encompassing at least the following types of policies considered to be relevant to data exchange processes: (inter)national laws and regulations, data sharing agreements, consortium agreements, sharing conditions, and access and usage control policies. This type of generality is hard to achieve using conventional access control mechanisms. Usage control mechanisms generalize access control with policy languages that are significantly more expressive [28]. Usage control is proposed as a control mechanism in the reference architecture document of the International Data Spaces Alliance (IDSA) [25]. More information on policy specification and usage control can be found in Section 3.3 and Section 3.4.

3.2 Administration

Policy administration is the process by which policies are registered, extended[1], and selected for use within dataspaces. The process raises both legal and technical questions, partially answered by this section.

- Who is responsible for the policies and the decisions made based on the policies? How can policies be administered in an accountable and reusable fashion?
- What policies (in particular: laws and regulations) should be applied within a certain dataspace and are relevant to a certain action? Who decides this? How can this be recorded?
- Via what process are social policies (often legal documents) formalized as executable policies? And who performs this task?

3.2.1 Policy types

Policies are administered in the federated Policy Store component of the architecture at different registration levels by different administrators on behalf of authorities. For example, the owner of a data asset may determine the conditions under which the asset can be used which are formalized and administered by a data custodian on behalf of the owner. An overview of different types of policies and their corresponding authorities and administrators are listed in Table 3.1. The federated nature of the Policy Store and the Catalog make it possible to register the same asset under different conditions within different dataspaces or ecosystems.

The policy types can be seen to form a hierarchy (visualized in Figure 3.2) corresponding to some extent to the hierarchy of the registration levels. Each level introduces details and specifics as relevant to a particular dataspace, ecosystem or resource and does so in relation to applicable legislation. For example, the international GDPR privacy regulation is applicable

[1]For reasons of accountability, policies may not be removed or modified once they have been used in accountable decision making processes. This topic is discussed in Section 3.4

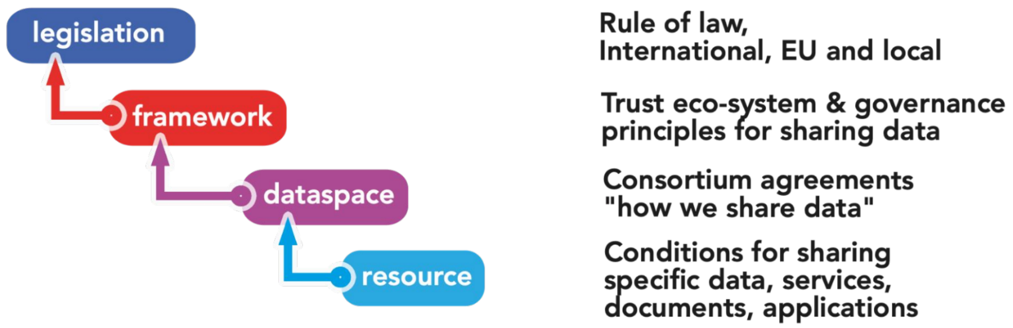

legislation
framework
dataspace
resource
Rule of law, International, EU and local
Trust eco-system & governance principles for sharing data
Consortium agreements "how we share data"
Conditions for sharing specific data, services, documents, applications

Figure 3.2: The hierarchy formed by policies applied within the AMdEX architecture.

to the scenarios of the DIPG case study (see Section 4.2) and is specialized by the regulatory document of the DIPG Network consortium. The document determines that, in terms of the GDPR, the members are jointly considered a joint controller, members can be considered the controller of specific data assets, and the Dutch Childhood Oncology Group (DCOG), that operates the DIPG registry, is considered a processor [30].

3.2.2 Policy construction

We consider at least three parties involved in the administration of a particular policy. Firstly, the authority with the (legal) power to instigate the relevance of a policy to a particular situation, expressing the policy using natural language in a document with a certain legal status (e.g., a legislation or contract). Secondly, the experts that formalize the natural language expression of the policy by forming a legal interpretation of the policy and expressing it as a technical, executable policy. Thirdly, the administrator responsible for determining which executable policies are applicable to a certain ecosystem, dataspace or resource. This determination is done by registering a policy within one or more Policy Stores, possibly through the instantiation of policies (policy templates) already registered (see below). These responsibilities may overlap and are highly inter-dependent. For example, the authority determines when a particular policy is legally applicable whereas the administrator determines applicability at the technical level. As another example, the formalization of policies requires intimate knowledge about the policies this policy extends or specializes.

The types of policies discussed previously necessarily contain large amounts of legal knowledge and technical knowledge. Legal expertise is required to interpret legislation, to formulate agreements and to align these agreements with legislation and other relevant policies. Software expertise is required to turn legal interpretations into computational artifacts that are consistent, non-ambiguous, explicate inter-dependencies, and can be used for effective reasoning. The construction of policies is therefore inherently a collaborative exercise between legal and software experts, but also between (legal and software) experts responsible for different types of policies within the policy hierarchy; the person formalizing a consortium agreement may not be the person formalizing the legislations on which it depends.

Policies are expected to be frequently revised and multiple, alternative versions may co-exist (e.g., for archiving and accountability purposes, or because multiple interpretations

Authority	**Policy type**	**Administrator**	**Mechanism**	**Registration level**
(Inter)National authorities	Legislation	?	Formalized to be reused across ecosystems and dataspaces	Universe
(Inter)National authorities	Sector-specific legislation	Ecosystem provider	Formalized to be reused across the dataspaces of this ecosystem	Ecosystem
Ecosystem authority	Ecosystem agreement	Ecosystem provider	Policies established to govern all dataspaces within the ecosystem	Ecosystem
Dataspace authority	Consortium agreement	Dataspace provider / custodian	An agreement between consortium members that determines archetypes of data exchange and dataspace-level conditions	Dataspace
Data/Resource owner	Resource conditions, offers	Data producer / service provider / custodian	Conditions on the use of a particular resource, set as part of an offer and turned into an agreement when the offer is accepted	Dataspace
?	Agreements, transactions	Notary (Automated)	Created once a request has been made, offers accepted, and necessary clearing checks performed	Dataspace
Data/Resource owner	Dispute	Data producer / service provider / custodian	Perceived non-compliance of a transaction can be reported	Dataspace

Table 3.1: Overview of different types of policies in the AMdEX policy hierarchy with the (legal) authorities determining the policies and the administrators formalizing and registering these policies at a certain registration level.

can be given). Policies cannot be deleted or modified; revisions happen by publishing new fragments that supersede other fragments from a particular moment onward. This decision promotes accountability and is needed to preserve the consistency of previous offers and agreements. Moreover, laws are revised in a similar fashion, as is reflected in the design of `wetten.nl`, the website on which Dutch laws are published in a structured and searchable format. To maximize reuse, policies should be modularized and configurable, e.g., as blueprints or templates. These requirements demand an expressive policy specification language and a rich data model for structuring policies, as is further discussed in Section 3.3.

Data Exchange Archetype Templates The observations made in this section show that the process of policy construction is complex and faceted, and it is unsure what this process will look like in the long term. Within the AMdEX Fieldlab project, a pragmatic approach has been taken, defining and applying policies in an ad-hoc fashion depending on the goals of a particular use case. However, the policies discussed in the context of the AMdEX Fieldlab use cases and use cases studied by the Data Sharing Coalition[2] have been analyzed in order to extract patterns in data exchange. Following the approaches of Shakeri [29] and Zhang [33], these patterns have been formalized as so-called *data exchange archetypes*.

Data exchange archetypes are frequently occurring patterns in the exchange of data assets, algorithms and resources. Figure 3.3 and Figure 3.4 visualize data exchange archetypes

[2] `https://datasharingcoalition.eu/use-cases/`

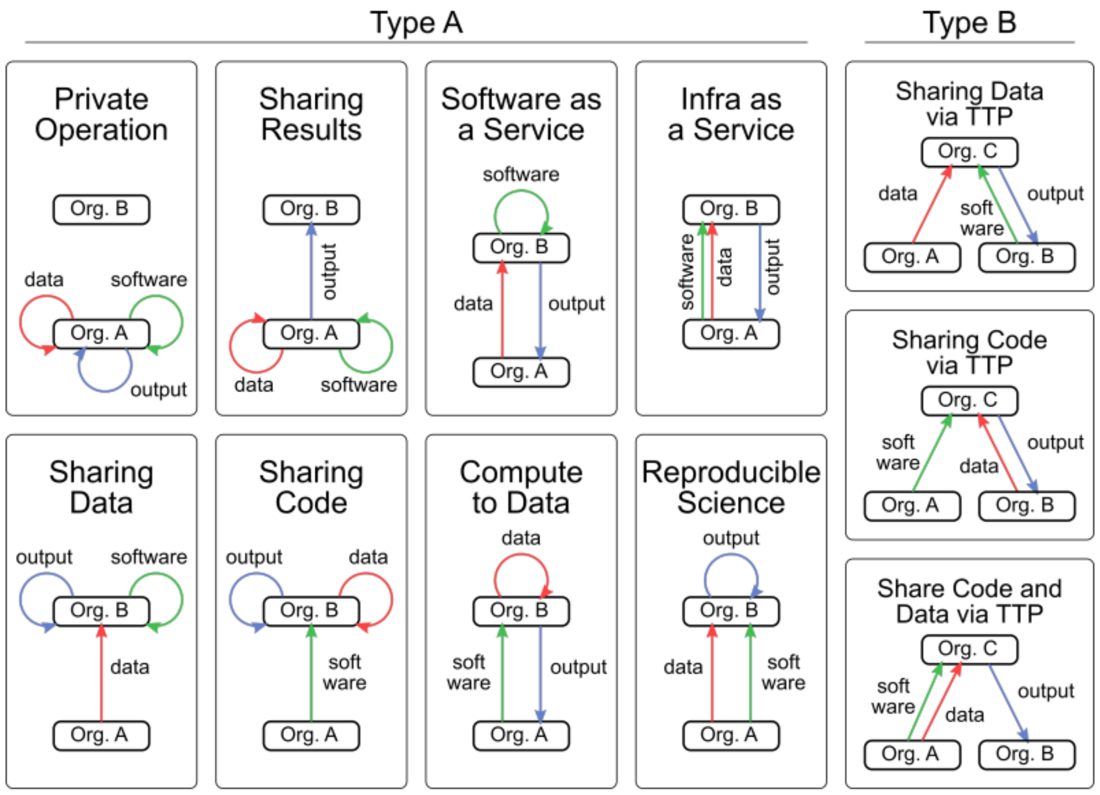


Figure 3.3: Exchange archetypes taken from [29]. TTP abbreviates 'trusted third party'.

of exchange processes involving one[3], two or three parties. The archetypes contain enough details to be useful as policies for configuring a data exchange infrastructure, although details specific to a particular use case or dataspace are omitted. The missing details are to be provided as part of an instantiation process. The archetypes are formulated once and for all as *templates* with certain parameters, such as which party plays the role of data producer or compute provider. The parameters are instantiated when an archetype is applied for a particular request within a dataspace.

A collection of data exchange templates have been formulated in the eFLINT language and is available online[4]. The collection covers the archetypes of Figure 3.3 and Figure 3.4. Templates for archetypes, sharing agreements, and data sharing conditions are to be made available as part of the AMdEX Framework and can be offered by dataspace or ecosystem providers to simplify the policy administration process for members. A consortium determines which archetypes can be applied within the dataspace of the consortium. Data producers and service providers can further restrict within which archetypes their assets and resources can be used and may set additional conditions they may wish to put in place. The repository containing the archetype templates also demonstrates how the templates are instantiated by applying the templates to use cases of the Data Sharing Coalition.

[3]The 'private operation' archetype does not really describe an *exchange* process.

[4]`https://gitlab.com/eflint/data-exchange-templates`

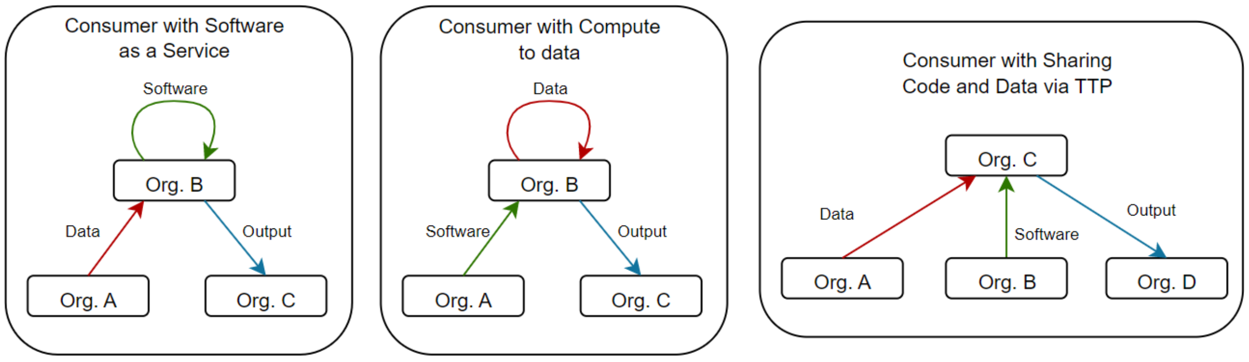


Figure 3.4: Additional archetypes involving three parties.

Policy composition Based on these ideas, the process by which policies are composed and applied within a dataspace contains at least the following steps:

1. Interpretation and **formalization of normative sources** such as legislation, AMdEX' rules of engagement, ecosystem rules, agreement templates, archetypes templates and condition templates. The resulting policies are abstract in the sense that certain details are omitted and are to be specified in the later steps.

2. Creating a **consortium agreement** by selecting a sharing agreement template and selecting the templates for applicable legislation, archetypes and conditions. Details of the consortium and its members are provided to instantiate the templates, e.g., determining which member plays which role in an archetype and setting the parameters of conditions (e.g., the length of an embargo period).

3. Creating **usage conditions** as part of the offering of assets or services. This is achieved by specializing condition templates. The chosen conditions must be consistent with the consortium agreement. These may be pre- and post-conditions.

4. As part of **auditing**, the trace of a data exchange process may be checked for compliance against the original and additional or alternative policies (interpretations).

A data model is required for the described composition of policies, as well as a modular policy specification language that supports templating.

3.3 Specification

As discussed in the previous section, many policies, and policies of different kindes, are expected to be relevant to a dataspace or (the steps of) a particular data exchange process within that dataspace. For various reasons, these policies should be specified as separate policy fragments. Each fragment may be set by a different authority and registered by a different administrator. Individual policy fragments should also be linked to the legal text of which it forms an interpretation, promoting transparency and accountability. Different versions of legal texts and policy fragments may have to co-exist as outdated versions may have to be maintained for accountability purposes. The following information model for policy administration accounts for these situations.

- A **source fragment** is a legal text set by a certain *authority*, has a legal status, and has a certain *time period of validity*. Examples are sections, articles, members and clauses in laws, regulations and sharing agreements. A source fragment has zero or more source fragments as children forming a tree structure, e.g. the members of an article or the articles in a section. A source fragment can have zero or more alternatives set by different authorities (e.g. different jurisdictions) or administrators (e.g. different interpretations), or with a different period of validity.

- A **normative source** is *a nominated source fragment* and a *selection* between alternatives of the descendants of the source fragment for which holds that:

  - for every fragment exactly one alternative is selected,
  - all the selected fragments must be set by the same authority, and
  - there must exist a time window in which all the selected fragments valid.

  A normative source can be published in the sense that it remains available without alteration as a coherent policy set applicable to specific scenarios.

- A **normative template** is a normative source in which some concepts are left abstract – referred to as *parameters* – and to be concretized the moment the normative source is applied to a particular case, e.g. which parties play the role of buyer and seller in a purchasing agreement. Some open terms may remain abstract.

- A **policy fragment** is a policy specification written in a particular *policy language*, linked to a particular version of a *source fragment* for which it forms the formal interpretation, may be *versioned*, and has *dependencies* on other policy fragments.

- A **policy set** is a collection of policy fragments for which one can determine:

  - whether all the policy fragments have been written in the same language,
  - whether all the *dependencies* of all the policy fragments are included in the set,
  - whether the policy fragments together are *unambiguous* and *logically consistent*,
  - and all the linked source fragments together *form a normative source*. If this is the case, the policy set is referred to as a **normative policy**.

- A **policy template** is a normative policy formalizing a normative template. An *instantiation mechanism* should concretize the parameters and transform the policy template into a normative policy.

The relations between policy fragments, policy sets, source fragments, and normative sources are expected to be very complex in practice. The full process by which these relations are maintained for ecosystems, dataspaces and resources has not been determined yet nor validated in practice. Legal and software expertise will have to come together in this process, as well as representatives of multiple organizations with different roles and interests.

Category	**Requirement**	**Description**
Abstraction	Levels of abstraction	Specifications can be written at different levels of abstraction, e.g. social and system policies.
	Concretization	A concept can concretize another concept, e.g. ‘storage’ concretizes ‘processing’ from the GDPR.
	Inheritance	A concept can inherit from another concept, e.g. the ‘store asset’ action inherits the pre- and post-conditions of the ‘process asset’ action.
	Templating	A policy specification can have parameters that can be replaced by concrete values.
Composition	Modularity	Policy specifications can be written as (small) composable and reusable elements.
	Well-formedness	A composition of policy specifications can be analyzed to determine it is well-formed.
	Logical consistency	A composition of policy specifications can be analyzed to determine it has conflicting elements.
	Priority mechanism	A mechanism is available to determine priority when elements of a composition are conflicting.
	Versioning	A policy specification can have alternatives.
Legal concepts	Power	Actions can affect the permissions, prohibitions and obligations of (other) actors.
	Legal obligation	The expectation that an action should be performed in the future or a certain state is to be maintained can be specified.
	Explicit conflicts	Explicit permissions and prohibitions can be simultaneously assigned (logical inconsistency).
	Violation responses	The consequences of violations can be expressed.
	Open terms	A concept may be left open for later, context-sensitive interpretation (see Concretization).
	Source references	A policy specification can be linked to the legal document interpreted by it, e.g. a law or contract.

Table 3.2: A collection of requirements on policy specification languages.

Requirements on policy languages The information model does not prescribe a particular policy language for the policy fragments and supports recording policy fragments written in different languages. However, not every policy language is equally suited for AMdEX purposes. Based on the scope of policy administration and enforcement discussed in the previous sections and the information model described above, requirements on policy specification languages have been identified and listed in Table 3.2.

The requirements related to abstraction and composition are predominantly inspired by object-oriented programming (abstraction, modularity) and declarative (logic) programming practices (consistency and priority) and follow also from the information model.

The requirements related to legal concepts are based on legal theory and research into the representation of legal concepts in logic programming and automating compliance in general. The appeal of Hohfeld’s theory on fundamental legal concepts is that the framework is action-oriented on the one hand and the particulars of the legal concept of power and duty on the other hand. The power concept associates with actions of particular actors certain consequences: the modification of normative positions held by (other) actors. The normative positions of an actor are the obligations, prohibitions and permissions of an actor. Legal obligations can be of two kinds: the obligation to act (referred to as a 'duty') and the obligation of fact. An actor holding an obligation of fact must preserve the truth of a certain fact, e.g. to ensure that only anonymized or encrypted data is stored. An actor holding an obligation to act (duty) must perform a certain action[5] in order to satisfy the duty, typically before a certain deadline is reached.

A second important aspect of Hohfeld's theory is that all normative positions create a so-called normative relation between actors: between an actor wielding a power and the *recipients* affected by the performance of the associated action (i.e. having their normative positions changed) and the *holder* of a duty and the *claimant* that typically stands to benefit from the performance of the associated action. As discussed in Section 3.4, normative relations can contribute to accountability and can serve as a coordination mechanism.

Candidate languages The **eFLINT language** has been designed with several of the listed requirements in mind and is therefore a good candidate for usage in AMdEX. In particular, eFLINT is based on Hohfeldian concepts. Specifications are highly modular and extensible, making it possible to reuse policy specifications across applications and to concretize concepts left abstract (parameters or open terms) in other, inherited specifications [2]. Specifications can be written at the level of abstraction of the source fragments they formalize and well-formedness checks are available on compositions. A model checker has been developed for eFLINT that can be used to find possible logical inconsistencies [12]. Performing prohibited actions or not satisfying duties (in time) result in violations. Such violations can grant powers to actors to act on the violation. The associated actions of these powers may have normative consequences. As such, it is possible to specify responses to violations as well as the consequences of these responses.

However, not all criteria are met. In particular, eFLINT does not have its own module system. And contrary to the FLINT language on which it is based [8], eFLINT does not maintain references to source fragments. For both aspects it relies on an external mechanism for administering eFLINT fragments as modules and linking these with source fragments. The information model described in this section complements eFLINT in these regards. An example implementation of the information model can be found in [21].

As opposed to Deontic Logics such as found in [16], prohibitions are not explicated and are instead inferred from the absence of permissions. Access control languages typically also have rules that explicitly determine 'permit' or 'deny' decisions. As a result, eFLINT specifications cannot produce conflicts between permissions and prohibitions, which is a limitation from a legal perspective, as in legal/social practice such situations can occur.

The **XACML policy language** is highly modular; policies are formed by rules and can be gathered (composed) in policy sets [24]. Although modular, there are no abstraction mechanisms available in the XACML policy language. Templating may be possible using generic templating engines. Explicit conflicts between 'permit' and 'deny' decisions can

[5]Or one of a set of alternative actions.

occur and policy sets need therefore be associated with a priority algorithm that determines precedence between rules. Other legal concepts are not directly part of the language. Note that 'obligations' in access control literature, including on XACML, are not always equivalent to legal obligations. In the context of XACML, an obligation is an operation that needs to be performed by an enforcement point alongside the execution of a policy decision. Such obligations are not legal obligations in the sense that they describe an obligation held by an actor and do not refer to the expectation to act in the future or to maintain a certain state. The combination of a language and enforcement mechanism that supports *usage* control can potentially represent legal obligations.

The **Open Digital Rights Language** (ODRL) [19] has several ideas in common with Hohfeldian legal theory. In particular, normative positions can be associated with two actors, an assigner and an assignee, and normative positions are associated with actions. The information model of ODRL mentions obligations, duties, permissions, and prohibitions. The power concept is not explicitly mentioned, but depending on the enforcement mechanism employed by a user of ODRL, the 'grantUse', 'consequence' and 'remedy' concepts can potentially be used to realize powers and to express the consequences of violations. The duty concept is ambiguous in the sense that it refers both to a concept more akin to a pre-condition and to a Hohfeldian obligation to act [22]. Policies can also be composed and a basic form of inheritance between policies is supported, i.e. inheriting the rules of a parent policy. The meta-data on ODRL policies also includes a 'isReplacedBy' property, enabling a form of versioning. The ODRL language can be extended with so-called 'profiles' that extend the vocabulary and associate additional semantics to the terms of the new vocabulary.

ODRL inspired the AMdEX architecture with its usage of 'offers' and 'agreements'. ODRL rules are part of a policy set that is offered by an assigner and may be accepted by an assignee to form an agreement. Sharing conditions in AMdEX are used in a similar way: a sharing condition template is instantiated by a resource owner when making a resource available as an offer. A consumer requests resources that, after having their conditions enforced in clearing, results in an agreement ready for processing (see Section 2.3 and Figure 2.7).

A more detailed, future study is required for ODRL, (novel) ODRL profiles, and other policy languages, in particular languages for usage control [28] such as the Obligation Specification Language (OSL) [17], its successors, and LUCON, the language used by the IDSA [20].

3.4 Enforcement

The division between policy specification in the previous section and policy enforcement in this section follows the notion that an access control model is the combination of a policy language and a mechanism to enforce policies within a software system. This section discusses how policy information is gathered and exchanged, and how policy decisions are enforced. Access and usage control models have been analyzed to extract requirements on monitoring and enforcement which are part of Table 3.3. The legal requirements have been added in support of the legal concepts discussed in the previous section and the principles of 'accountability' and 'human-in-the-loop' (see in particular the 'external observations' and 'ex-post' requirements).

Category	**Requirement**	**Description**
Monitoring	Actions and events	Relevant events and actions can be observed.
	Attributes	(Changes to) Attributes of the system, actors and resources can be observed.
	Expected actions	The mechanism keeps track of actions that are expected to be performed (may result in violations).
	Invariants	The mechanism keeps track of whether certain invariants are maintained (may result in violations).
	Continuous monitoring	Monitoring is continuous or at small intervals.
	External observations	Users can bring in information and observations about events and actions external to the system.
Enforcement	Interception	Actions with enforceable pre-conditions can be prevented following a policy decision.
	Roll-back	Actions with enforceable post-conditions can be undone following a policy decision.
	Interruption	Actions can be interrupted and undone following a violation caused by the action.
	Effects	Actions and events can affect attributes in a way specified by the policy language.
Legal requirements	Accountable	The decisions made by the mechanism can be justified, traced and linked to legal interpretations.
	Static ex-ante	Determine whether an application is compliant before it is executed, i.e. compliance-by-design
	Dynamic ex-ante	Violations can be prevented at run-time
	Dynamic ex-post	Run-time violations can occur and be acted upon either automatically or through a user-action
	Static ex-post	Decisions can be (re-)analyzed for compliance, e.g. given new interpretations or information.

Table 3.3: A collection of requirements on policy enforcement mechanisms.

The original UCON usage control model [28] goes beyond access control by allowing attributes to be mutated as the consequence of actions ('effects') and thereby enabling access to be conditioned by the executed of prior actions. The UCON+ model goes further and includes the monitoring of 'continuous events' [15]. The ODRL language can be used to express 'expected actions' whose fulfillment needs to be monitored [19].

The ability to monitor expected actions and invariants makes it possible to monitor compliance with obligations of fact (invariants) and obligations to act (expected actions). The recorded expected actions (and their deadlines) can be compared with observations about executed actions to determine whether violations have occurred. Ex-post mechanism are needed to enforce legal obligations: obligation holders can be pro-actively encouraged (e.g. reminded) to fulfill their obligations, but responses (penalties, remedies) to violations may ultimately be necessary.

In Section 2.1 a distinction is made between pre- and post-conditions, referring to conditions on assets/resource that can be checked before or after the asset/resource is accessed or used. The post-conditions can be further distinguished as conditions that:

(a) can be enforced (directly) and automatically,

(b) must be enforced at some future moment in time,

(c) must be upheld continuously, or

(d) cannot be enforced automatically within the system at all.

Examples are:

(a) the application of algorithm X must yield an output with K-anonimity > 5,

(b) when my dataset is used as part of an analysis, I must be notified within 24 hours,

(c) the result of analyzing public dataset X must remain publicly available, and

(d) the owner of dataset X must be acknowledged when an analysis using X contributes to a published paper

Post-conditions of type (a) are enforced using the 'roll-back' control mechanism. The post-conditions of types (b,c,d) correspond to (legal) obligations of fact or to act and require 'ex-post' control. In addition, type (b) requires monitoring expected actions, type (c) requires monitoring invariants, and type (d) requires external observations. In case of the example, a data owner may observe that a party they have shared a data asset with has recently published a paper without acknowledgments whose results are expected to be based on the shared asset. By bringing this information into the system, a violation can be raised and acted upon. Note also that example (a) is an example of a policy that has information as input that is derived from the contents of a data asset. Conditions for which this is the case can only be enforced in the data plane as the control plane will not have access to the contents.

In Section 2.3.3 enforcement is discussed as occurring at several moments during the lifetime of a data exchange process, as part of OFFERING, CLEARING, PROCESSING, and

AUDITING. During OFFERING, additional policies may be set that are specific to the offered asset or resource. These policies are then composed with the relevant higher-level policies (see Figure 3.2) and the composition is checked for well-formedness and logical consistency. The policy language is expected to support such checks (see Table 3.2).

When REQUESTING the execution of a data exchange process, all policies relevant to the request are gathered and handled through clearing, resulting in a plan of concrete processing steps. The plan is checked for compliance with the applicable policies ('static ex-ante', Table 3.3). At this stage, not all conditions can be enforced, as some may require information that is only available at run-time, during processing (e.g. the K-anonimity of a data asset produced by processing). Moreover, policy information can be dynamic and may change as the plan is executed.

For these reasons, enforcement is also required in the PROCESSING stage ('dynamic ex-ante' and 'dynamic ex-post'). The formulated monitoring and ex-post enforcement requirements have also been inspired by run-time verification [1]. Several of the difficulties relating to run-time verification in the context of decentralized, distributed systems are explained in [10]. In particular, the enforcement mechanism promoted here is inherently *intrusive* in that observed violations may influence the behavior of system being monitored.

As a data exchange process is executed, logs are created and maintained by the Notary components. During AUDITING the logs can be analyzed in a form of *off-line* run-time verification or compliance checking [32]. Such 'static ex-post' checks are particularly important during disputes, which are typically caused by disagreement over the (validity of) policy information or the (interpretation of) policies used during decision making. Disputes can also be raised in response to external observations. A powerful Auditing component thus makes it possible to reason about past data exchange processing using additional policy information, additional (external) observations and against alternative interpretations of policies.

Acquiring policy information The enforcement described in this section relies on:

- the implementation of control mechanisms to enforce (pre- and post-)conditions that can be localized either in the data plane or the control plane,
- the implementation of monitors to provide policy information to reason about the satisfaction of conditions, and
- one or more Enforcement Orchestrators that mediate between the Policy Reasoner and the control mechanisms and monitors (playing the role of Context Handler in Figure 3.1).

The implementation of a system with such a powerful form of enforcement faces several challenges. Firstly, a condition may express that "during processing, no networking communication is permitted", but the enforcement of such a condition requires the ability to monitor (the existence of) network activity and the ability to interrupt processing once network activity is observed. In general, member nodes and exchange providers may not provide the control mechanisms to enforce the conditions expressed in policies. (The 'external observation' requirement serves as a mitigating fall-back.)

Secondly, even when the required monitoring and control mechanisms are available, the information provided by the monitor, the condition on which the decision is based, and the control mechanism executing the decision need to share a vocabulary and semantics (e.g. use and apply the same meaning to the term “network activity”). Semantic web[6] and linked data techniques can be applied to explicate such vocabularies (or more generally: ontologies) and to ensure “speaking the same language”. However, this raises the implementation effort, as local conversions may be needed to convert internal information to information structured according to the shared vocabulary. Vocabulary-independent approaches for querying linked data have been explored [11].

Accountability and the sensitivity of policy information The ‘accountable’ requirement is formulated in support of the auditing processes envisioned for AMdEX dataspaces. The prior discussions show that policy information used in policy decisions can be associated with different entities in a system – actors/users, assets/resources, system components, and processing steps – and can even come from outside the system. For accountability purposes, this information should be recorded and remain available for future auditing. However, some information may be sensitive, especially if it relates to human actors (such as consent), assets/resources or processing steps (e.g. execution logs). As an example of the latter, consider conditions that relate to the processing performed by a compute provider. These may be enforced locally based on information obtained by monitoring the execution of an algorithm. For accountability, (parts of) execution logs may have to be made available to an auditor. Compute providers may generally not be willing, if the logging information is considered to be sensitive.

Given their sensitivity, this kind of information should be treated as data assets, which can only be used under certain conditions (e.g. for the auditing purpose). It follows that AMdEX will not processes these assets, as AMdEX only processes meta-data (see the principles of Section 1.3.1). A comprehensive solution for accountability and auditing must thus be found within the consortium. Dataspace members should agree on the granularity and level of detail with which the information is stored, by which member(s), and under which conditions. An auditing and accountability framework that addresses these matters is subject of future investigations.

In [34], an approach is introduced in which policies are used to enforce a form of accountability. In the approach, the meta-data of a data asset contains policies that nominate ‘auditors’ – components in the system that need to sign off on actions performed on the data asset. As a result, the auditors are guaranteed to be notified about these actions, can reason about their compliance, and log the actions. The approach can be used to ensure that both actors involved in the action associated with a Hohfeldian, normative relation (see Section 3.3) are notified. This way, the claimant of a duty will be informed of the existence of the duty and can hold the holder of the duty accountable when the duty is violated.

[6]`http://www.w3.org/DesignIssues/LinkedData.html`

Chapter 4

Use Cases

4.1 University Personnel – UNL WOPI

The association University of the Netherlands[1] (UNL) is an umbrella organization acting on behalf of the universities of The Netherlands. The UNL collects data about the employees of the universities for the benefit of the universities and to report to the Ministry of Education. This process is known as WOPI ("Wetenschappelijk Onderwijs Personeel Informatie"). The universities have made agreements on the type of data shared and for which purpose. The existing agreements are made available at Edustandaard[2]. The challenge of this use case is the trade-off between the privacy of university personnel and the kinds of analyses that are required. Some analyses, after all, require data points at the level of individuals and cannot (easily) be performed on aggregated data, such as linear regressions. Another trade off that is between the control achieved through manual intervention and the efficiency of automated decision making. The goal of the AMdEX-UNL use case is investigate these trade-offs and to see whether the compute-to-data and third-party computation data exchange archetypes can offer solutions that are feasible from organizational and technical perspective.

The solutions are expected to meet the following *requirements*:

- The universities and the UNL should have an equal information position; both the universities and the UNL can request data exchange in equal fashion.
- The UNL can continue to report to the ministry on behalf of the universities; the information the UNL can acquire should be sufficient for producing the reports the ministry expects.
- Any conditions about offered data, shared data (such as regarding aggregation levels), and the level of detail at which the UNL can report to the ministry are recorded transparently.

The solutions rely on the following assumptions:

[1] `https://unl.nl`

[2] `https://edustandaard.nl/standaard_afspraken/definitieafspraken-personeel-universiteiten-wopi`

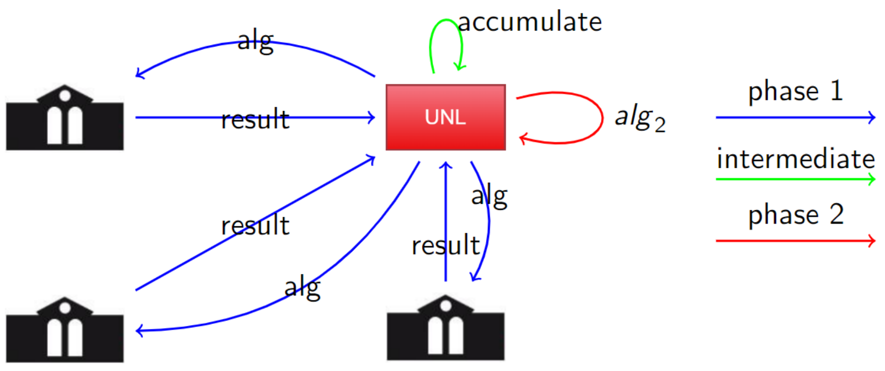


Figure 4.1: Scenario 1 of the UNL use case involves two data analysis phases and an accumulation phase. The local results (phase 1) are obtained by executing a received query (alg) and are sent to the requesting party (UNL, in the example). The requesting party accumulates the results and executes the second query on the joint results (phase 2).

Member	**User**	**Role**	**Component**
UNL	analyst(UNL)	data consumer / algorithm provider	consumer node
University X	analyst(X) custodian(X)	data consumer asset provider / compute provider	consumer node compute node

Table 4.1: The different users, roles and architecture components associated with the members of the UNL consortium. The consortium consists of a number of universities and the UNL association.

- The universities have agreed on a shared format for structuring WOPI-data and a shared understanding on how the fields are populated (e.g., use the same categories in the same way).
- The universities are willing and able to execute queries on their local WOPI-data.

The first assumption significantly simplifies the data processing required as the algorithms can run at the various sites without pre-processing of data. The second assumption is necessary for the compute-to-data concept.

The University of Amsterdam has developed a prototype that demonstrates the principles, advantages and disadvantages of compute-to-data and third-party-computation. The prototype makes it possible for data analyst and data custodians at the universities to experiment with these forms of data exchange and the aforementioned trade-offs. The experienced gained from interacting with the prototype is intended to provide input into the formation new agreements on WOPI-data processing. The following subsections describes the user scenarios realized by the prototype.

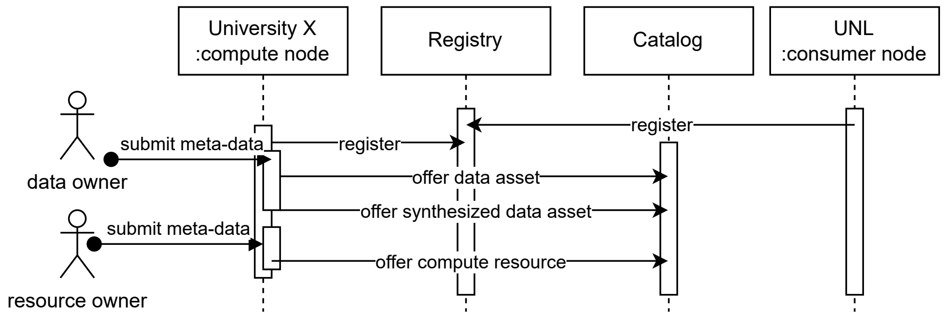


Figure 4.2: A sequence diagram showing the interactions between users and technical components during the onboarding of a university and of the UNL.

4.1.1 Scenario 1 – Manual Approval

Users and roles In the first usage scenario, a data analyst working for a consortium member submits an algorithm (an SQL query) that will be sent to be applied to local WOPI-data of one or more universities. A data custodian at a university has the power to inspect the algorithm, to decide to apply the algorithm, and to decide, after inspection, whether the results can be sent in return to the analyst. When results of multiple universities are sent back, the local results are accumulated and the joint results can be further analyzed (see Figure 4.1) by the analyst. Following from the first requirement, each consortium member has the ability to submit data processing requests (i.e., to act as a data consumer). All data providers (the universities) are able to determine whether to accept such requests. The UNL itself is not a data provider, but, following the second requirement, can make data processing requests. Table 4.1 lays out the consortium, the users, the roles they perform and the technical components required for the UNL dataspace.

Technical solution The sequence diagram in Figure 4.2 shows the onboarding process during which member nodes are registered in the Registry. Every university offers two data assets and their compute resource by submitting meta-data about these services to the Catalog. The first dataset is the WOPI-data of the university. The second data asset contains synthesized data produced using the WOPI-data. The dataset is produced by replacing personal information with synthetic data such that the dataset displays similar statistical properties as the original when used in analyses. The solution assumes the algorithm for data synthesis is agreed upon by the consortium and has been made available outside of the control of the AMdEX infrastructure. If desired, synthesis can be added as a processing step by applying a particular synthesis algorithm registered in the Catalog. The UNL registers its consumer node in the Registry and does not offer any services in the Catalog.

The sequence diagram in Figure 4.3 shows the interaction between the users and technical components that realize the described scenario. In the example, the analyst of the UNL requests the data processing, but the process can also be initiated by university analysts, in which case the UNL need not be involved. The request created by the analyst selects scenario 1 as a template and selects which universities' data is being requested by choosing assets and providing a role assignment, i.e., the selected universities are assigned the role of compute provider. The 'compute' message sent by the Process Orchestrator is a request

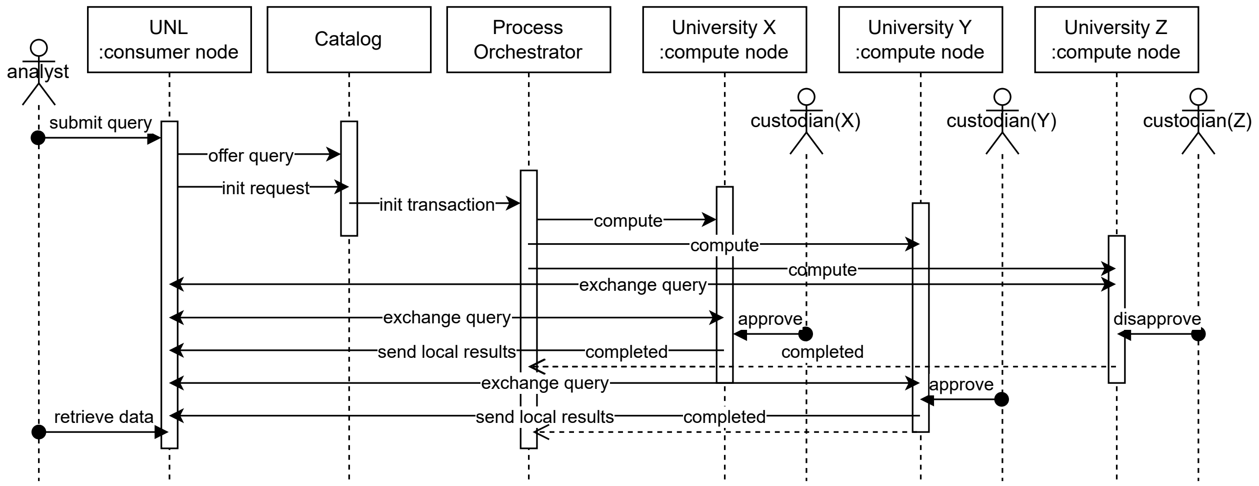


Figure 4.3: A sequence diagram showing the interaction between users and technical components for scenario 1 of the UNL use case in which data custodians give approval based on an inspection of (the results produced by) the query provided by the data analyst.

rather than an instruction; the application of the query to the local WOPI-data is initiated by the custodian. In the next scenario the Process Orchestrator instructs the compute node to perform the processing, possibly without explicit approval of the custodian. Note that the query and the data assets are exchanged only between nodes in the data plane.

The diagram is simplified in that confirmation messages about sent/received assets from member nodes to the Process Orchestrator are not (all) shown and in that the notary component has been omitted. No clearing component is required in this scenario. The local processing step is executed manually – the custodian retrieves the query, inspects it, optionally applies it, and after inspection of the results, decides whether to send the results to the consumer. The accumulation of local results and the second phase of the analysis are not shown as they are performed by the data analyst without involvement of other components and members of the dataspace. In the prototype the accumulation is performed automatically by the implementation of the consumer node. The node runs an interface in which an analyst can use an interactive query builder to build an SQL query for submission (phase 1) and for performing an analysis on the accumulated results (phase 2).

Reflections The parties involved in the case agree on a schema for WOPI-data (first assumption) and on individual queries to be executed. An important difference between the two is that the former is infrequently (at most once per year) and the latter can be done frequently, namely once for every request through the discussed approval process. This enables data analysts to submit a plethora of queries (throughout the year) depending on their specific needs.

Contrary to the existing WOPI-practices, the solution offered by the prototype does indeed give an equal information position to all participants (first requirement). (Assuming the data custodians actly fairly, i.e. do not discriminate against a particular member.) As a practical consequence, universities can benefit from available WOPI-data by performing analyses and use the extracted information to inform their policy decisions.

In Figure 4.3, the custodian of University Z has disapproved the request. As a consequence, the analyst only receives the processing results of two universities. This observation raises the question whether an agreement can be processed if not all parties agree and participate. This decision can be seen as a parameter of the applied template and is to be determined by the consortium. A related question is whether custodians should be made aware of the decisions of other custodians as this may influence their decision. The relevance of this question is best exemplified by looking at the extreme case in which all but one custodian disapprove. In this case the analyst will gain information specific to one university and the custodian of that university may not have approved the request if they were aware this was going to be the case.

The interface of the prototype makes it possible for the analyst to build an SQL query for submission interactively. This query can be tested on the local data to ensure it delivers the expected results before submitting it in a request. In some cases it may be desirable to test the (second) query on the dataset distributed across partners. Such a debugging process is common in data analysis, as it is in general-purpose programming. However, we do not want to include the manual intervention of data custodians in the debugging process of a data analyst as this is frustrating for both parties: the custodians may receive a lot of requests and the analyst may have to wait frequently and extensively. The next scenario describes how we experimented with the usage of synthesized data in requests that are automatically approved, streamlining the debugging process.

4.1.2 Scenario 2 – Automatic Processing and Clearing

In the previous scenario, the compute step is executed by the data custodian, giving the custodian control over the way the WOPI-data of their university is used. A practical downside is that manual approval forms a bottleneck in the process, whereas certain analysis requests may not require manual intervention. For example, the consortium may decide that queries on synthesized data are safe. To enable the system to adapt to the type of data that is requested – real or synthetic – the approval step should be handled by the AMdEX infrastructure. In solution visualized in Figure 4.4, the Enforcement Orchestrator is introduced for this purpose. The Clearing component is introduced to request manual approval when real data is used. For simplicity, only one university is included in the image and the interaction of the Enforcement Orchestrator with the Policy Store and Reasoner are omitted. The assumption is that before the request has been initialized, the consortium has registered an agreement in the policy store that determines that requests involving original WOPI-data require manual intervention whereas requests involving synthesized data do not.

The Process Orchestrator first gathers permission from the Enforcement Orchestrator before instructing the university to commence processing. In this example, real data is requested and the Enforcement Orchestrator asks Clearing to gather manual approval. The Process Orchestrator will only send the compute instruction when the processing is approved by the custodian and this fact is communicated as a permission to the Process Orchestrator. If synthetic data was requested instead, the Enforcement Orchestrator would inform the Process Orchestrator about its permission without the interaction with Clearing.

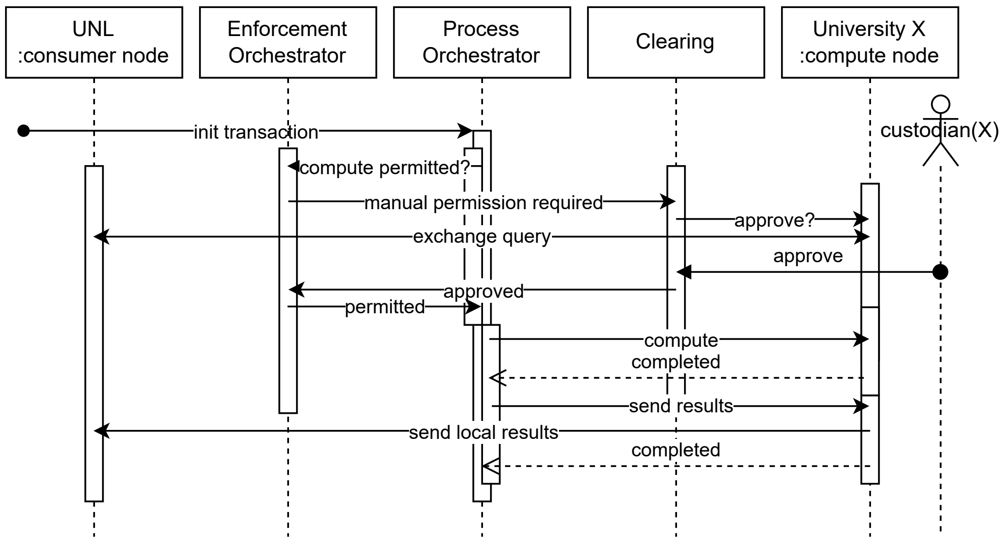


Figure 4.4: Alternative solution in which an Enforcement Orchestrator is used to determine whether manual intervention is required. If so, the Clearing component gathers the required approval from data custodians. The interaction of the Enforcement Orchestrator with the Policy Store and Reasoner are omitted.

Reflections In the prototype described, a fixed synthesis algorithm is used that is assumed to be agreed upon by the consortium partners. The synthesized dataset is offered in the Catalog together with the real dataset. The synthesis algorithm is applied locally without involvement of the AMdEX infrastructure which is therefore unaware of the algorithm applied and is unable to verify whether the dataset is indeed synthesized (and does not contain personal data). An alternative is for the UNL (as consortium representative) to offer a synthesis algorithm in the Catalog. In this way, the AMdEX infrastructure can be used to enforce the proper synthesis of data. For example, the template for the synthesized data request can be modified to include the compute step in which the synthesis algorithm is applied. (Alternatively, this compute step can be part of the on-boarding process, potentially avoiding some overhead.)

To avoid the transfer of the original data, the synthesis algorithm is to be applied locally by the compute node of the university. A mechanism may then be desired to verify that the result produced by this compute step is indeed the result of the (successful) application of the synthesis algorithm. One option is for an auditor to verify this by attempting to reproduce the synthesized dataset using the original dataset and the synthesis algorithm. This solution implies that the auditor has the right to access the original dataset, the synthesized dataset, and the algorithm. Furthermore, this solution implies that the synthesis algorithm can be re-played (e.g., using a seed to steer the internal pseudo-random process). This example demonstrates the importance of logging meta-data and in some cases also the archiving of (data and algorithm) assets. The latter reveals a fundamental trade-off in the design of the AMdEX approach: if AMdEX does not process (data) assets, then validation of the processing steps using the assets themselves must happen outside the AMdEX infrastructure. This observation underscores the importance of dispute resolution and external auditing processes. Other options are available though. For example, the consortium can involve a storage provider, storing encrypted assets in support of auditing/validation processes. In this model, access permission implies receiving a key that can be used to decrypt an asset. Such solutions have been investigated and employed in the context of secure cloud computing [26, 23] and homomorphic encryption can be used to perform computation on an encrypted asset without decrypting the asset first.

The company BlueGen.ai[3] provides data synthesis as a service. Their solution generates a report alongside every synthesized dataset that determines 'resemblance', 'utility' and 'privacy' statistics, comparing the synthesized data with the original dataset from which the dataset was generated. The existence of such a report can also be used to verify that the offered synthesized dataset can be used for automatically approved processing. The statistics in the report also provide a fine-grained control mechanism, enabling policies that, for example, determine that the privacy value and utility value of a synthesized dataset must be above a certain threshold. In a pilot, BlueGen.ai has successfully generated synthesized data for the UNL use case that scores highly on the aforementioned criteria.

A limitation of the first (and second) scenario is that row-level data may be need to be produced locally as a result of the first query in order for the second query to be able to produce the desired results and that custodians may not approve such requests if they consider it the case that too much personal information would be revealed. The first query submitted by the analyst may perform some aggregation and/or may perform a projection (drop some attributes) such that the result of the first query does not contain personal data. In some cases, it may be possible to determine automatically whether (too much) personal data is in the result and the consortium may decide that in these cases manual approval is not needed. In the prototype we have experimented with the concept of K-anonymity, determining for each individual how many other individuals contribute the same information in the result set. The number k is the smallest such number and acts as a measure to quantify the level of risk of revealing personal information associated with sharing the result set. Some analyses, however, require that the first query provides detailed row-level data. An example is linear regression. An alternative to denying such request or revealing the row-level data to the analyst is the use of a trusted third party (TTP). This scenario is explored in the next subsection.

4.1.3 Scenario 3 – Trusted Third Party

In the trusted third party (TTP) scenario an additional member[4] is part of the consortium with the role of compute provider, the TTP. The TTP is a 'third party' in the sense that as an entity, they are neither a data user or data owner, but instead provide a service. The TTP is, however, a member of the consortium and the dataspace. Figure 4.5 shows the exchange of assets in the TTP scenario, with Surf acting as a TTP. The consortium is laid out in Table 4.2.

In Figure 4.6 the interaction between the Process Orchestrator and the member nodes is visualized. In this scenario, the request of the analyst instantiates the TTP template and

[3]`https://bluegen.ai/`

[4]One of the existing members could also fulfill the role, e.g., one of the universities or the UNL.

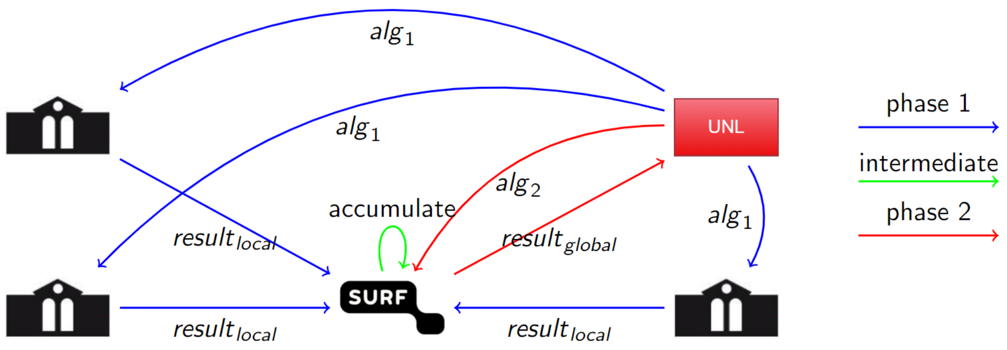


Figure 4.5: Scenario 3 of the UNL use case involves two data analysis phases and an accumulation phase. The local results (phase 1) are obtained by executing a received query (alg) and are sent to the trusted third party (Surf, in the example). The TTP accumulates the results and executes the second query on the joint results (accumulation). The global results (phase 2) are sent to the requesting party.

Member	**User**	**Role**	**Component**
UNL	analyst(UNL)	data consumer / algorithm provider	consumer node
Surf	resource owner	compute provider	compute node

University X	analyst(X)
	custodian(X)
		data consumer
		asset provider / compute provider
			consumer node
			compute node

Table 4.2: The different users, roles and architecture components associated with the members of the UNL consortium for the TTP scenario.

includes the second query. In the previous scenarios the second query was implicit and not known to the consortium. In the prototype, the custodians of the universities can inspect both queries to determine whether they approve their participation in the request. In the diagram these steps are omitted for brevity.

The member nodes that received instruction from the Process Orchestrator communicate back when they have sent the local processing results. This is important information for the Process Orchestrator, needed to decide when and whether to send the compute instruction to the TTP (besides being needed for logging). An important design decision is whether the compute instruction is sent only when all or more than one universities have sent their local results or whether one would suffice (see also the reflections paragraph in Section 4.1.1). The Process Orchestrator also needs to know whether local results were not sent by a university because the custodian did not approve or because the results have not been sent (yet) for another reason. In the latter case, the compute instruction may have to be sent again.,

Reflections An important consideration to this scenario is that the TTP needs to be trusted by the consortium members to appropriately handle the possibly sensitive data. An alternative solution, not requiring this kind of trust, is to employ the secure multiparty computation (sMPC) method in which a cryptographic protocol is executed between

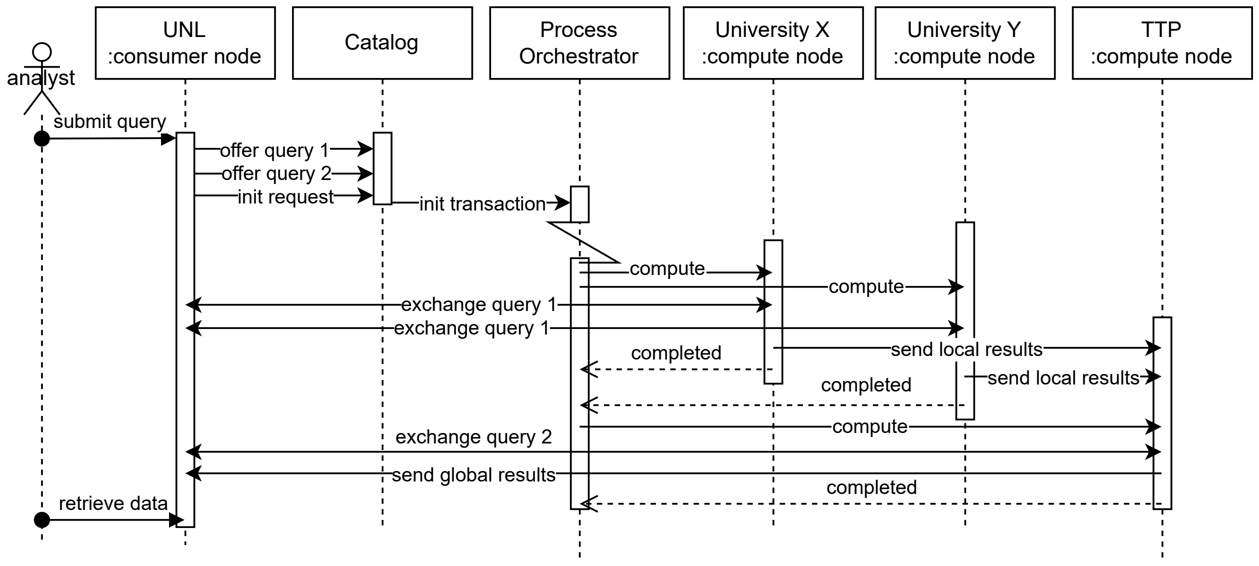


Figure 4.6: Alternative solution in which a trusted third party executes the processing step in which the second query is applied. The data analyst only receives the global results, not the local results produced at the universities. The process by which the custodians and the resource owner give approval for the processing is not shown.

dataspace members. The sMPC method is applicable in TTP scenarios without actually involving a TTP – the cryptographic protocol effectively replaces the third-party.

The different solutions explored in this section have in common that the UNL and universities have the same capabilities in terms of the submission of data analysis algorithms (requirement 1). This is achieved in the prototype by providing all analyst with the same compute node and interface for building queries and submitting requests. The assumption about a shared data format (assumption 1) has made this significantly easier, although bespoke pre-processing algorithm could have been added to each of the compute nodes. The second requirements requires the UNL to be able to execute sufficiently expressive queries in order to be able to report to the Ministry of Education. The main limitations of our solutions in this regard are that (a) the prototype is restricted to SQL queries for algorithms and that (b) custodians may decide not to make certain local results available if these are considered to be too sensitive. The TTP scenario was introduced to mitigate the latter. Most important is that the consortium members come to an agreement about the kinds of analyses that should be acceptable and that these are aligned with the requirements to report to the Ministry. Once made, it would be interesting to further explore to which extent these conditions can be automated, and what control mechanisms are needed, as we have done for the processing of synthesized data and with K-anonymity.

4.2 Rare disease – DIPG

This section describes a use case executed by the University of Amsterdam, focusing on policy administration and enforcement within a relatively simple data exchange archetype.

4.2.1 The DIPG Network

The DIPG use case concerns data sharing between medical institutions that treat patients with the rare DIPG disease. The DIPG Network is formed between these institutions with the goal of bringing together the horizontally[5] split data collected by the institutions in order to improve research into the disease through the availability of more datapoints. The DIPG Network is a consortium of international medical institutions that make data available on patients with the decease in a registry (called the 'DIPG Registry and Imaging Repository' and 'DIPG Registry' in short). In [2], the use case is introduced as follows:

> Diffuse Intrinsic Pontine Glioma (DIPG), also known as diffuse midline glioma (DMG), is a rare pediatric brain cancer for which there is no curative treatment, despite decades of clinical trials. Children suffering from DIPG face a dismal prognosis, with a median overall survival (OS) of eleven months and a two-year OS of less than 10%. In order to advance DIPG research, the SIOPE DIPG/DMG Network and Registry were established in 2011. The registry holds information on DIPG patients across Europe and a partner registry in North America – the International DIPG/DMG Registry – includes patient data primarily from the USA, Canada and Australia, with additional international members. The registry serves to improve DIPG research by granting members (conditional) access to selected datasets in order to perform analyses with more data points and thus higher efficacy.

This section reports on a prototype developed to experiment with the integration and enforcement of normative documents within a dataspace for the DIPG Network. The use case is originally from the Enabling Personalized Interventions project[6] led by the University of Amsterdam. In the prototype, parts of the following types of normative documents are formalized using the eFLINT language [3]. A minimal version of the resulting specifications can be found online[7] of which an overview is provided by Figure 4.7.

- A regulation – the GDPR privacy regulation of the European Union [6].
- A consortium agreement – with elements derived from the 'regulatory document' of the DIPG Registry and Imaging Repository [30]

The aforementioned paper [2] discusses how formalizations of these norms can be utilized to make access control decisions in order to realize the conditional access mentioned in the

[5]In a horizontal split, data owners possess data of different data subjects. In a vertical split, the data of a subject is distributed across several data owners.

[6]`https://enablingpersonalizedinterventions.nl/`

[7]`https://gitlab.com/eflint/eflint-examples/-/tree/main/dex-dipg`

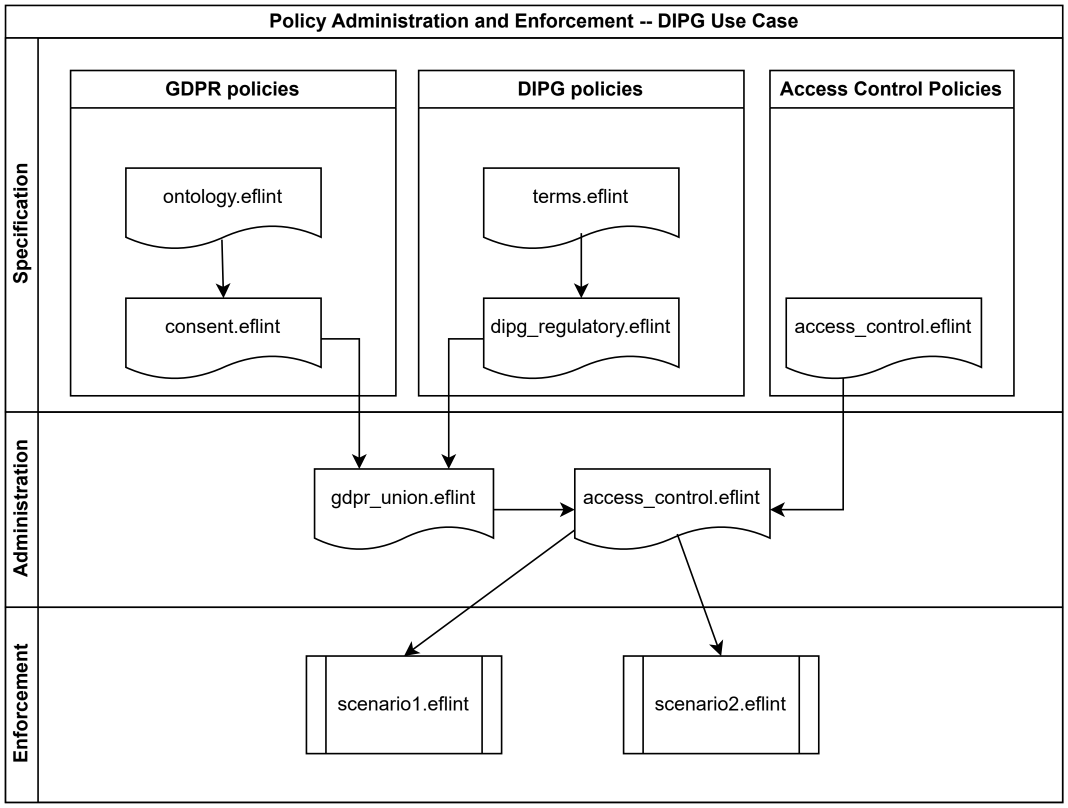


Figure 4.7: An overview of the dependencies between policy files used for the DIPG use case. An arrow $s \longrightarrow t$ indicates that the definitions from s are imported into t. The files containing "union" bring together concepts from different specifications. The scenario files are dynamically unfolding during the lifetime of an exchange process.

description of the use case above. The prototype described in this section demonstrates how this kind of access control can be realized in practice.

Referring to Table 3.1, the European Union is the authority that set the GDPR. The DIPG Network has selected an Executive Committee (EC) with several responsibilities. In the prototype, the EC is the authority that sets the consortium agreement. The dataspace provider (in this case the University of Amsterdam) has administered the formal interpretation of (aspects of) the GDPR and the DIPG Regulatory Document as policies. The prototype uses a basic form of Policy Store in which policies are stored as files. Separate files are used for various aspects of the policies to promote reuse and separation of concern. The files and their dependencies are shown in Figure 4.7.

The file "access_union.eflint" – uniting the union of the GDPR policy and the Regulatory Document with standard access control concepts – provides the context in which policy decisions are made by the Policy Reasoner. The files "scenario1.eflint" and "scenario2.eflint" are dynamically unfolding scenarios reflecting the policy information and queries communicated with the Policy Reasoner by the Enforcement Reasoner as part of an exchange process (steps PROPOSING to PROCESSING of Figure 2.7 in this use case).

Member	User	Role	Component
Hospital A	researcher	data consumer / algorithm provider	consumer node
Registry	custodian	data provider / storage provider	storage node

Table 4.3: The different members, users, roles and member nodes involved in Scenario 1.

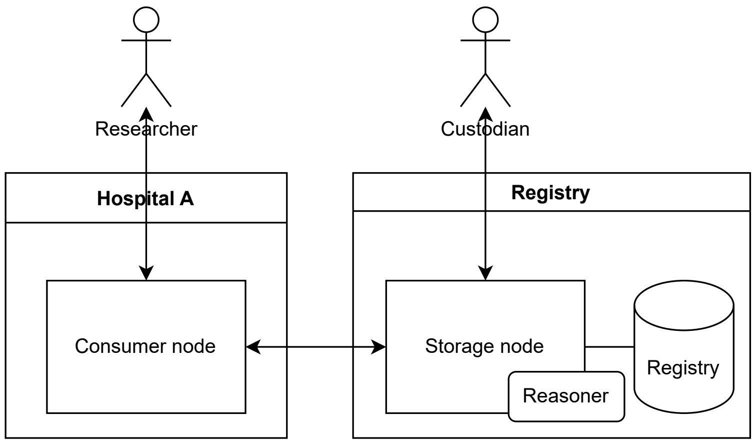


Figure 4.8: Diagram visualizing the dataspace members and users for DIPG scenario 1.

4.2.2 Scenario 1 – requesting access to data

Proposing a project In the first scenario, a researcher requests access to data held in the DIPG Registry through the submission of a project proposal (see the users and roles in Table 4.3 and Figure 4.8). At this moment, the researcher's institution is already assumed to be a member of the DIPG Network and the DIPG dataspace created for the DIPG Network (ONBOARDING, see Figure 2.7). The proposal stage involves a number of interactions between the researcher and the executive committee of the DIPG Network as displayed in Figure 4.9 (PROPOSING, OFFERING). Most importantly, the proposal needs to indicate precisely the research to be conducted in order to: (a) establish a precise purpose for the processing and

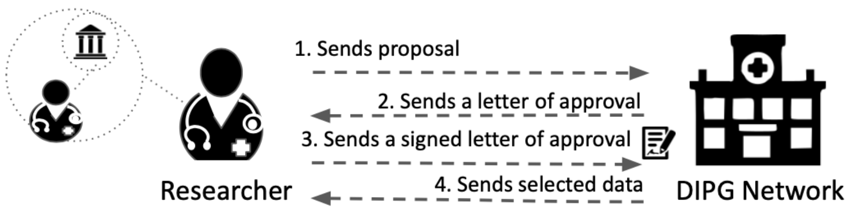

1. Sends proposal
2. Sends a letter of approval
3. Sends a signed letter of approval
4. Sends selected data
Researcher
DIPG Network

Figure 4.9: The PROPOSING process by which a researcher (on behalf of some institution) requests data for a particular research project and purpose. When the project is approved by the Executive Committee (EC), a data custodian (on behalf of the EC) selects the data and offers it to the researcher in the Catalog (OFFERING). Image taken from [2].

```
// 1) information about hospitals, obtained during ONBOARDING
+member(HospitalA).
+affiliated-with(Researcher, HospitalA).

// 2) project P1 is PROPOSED and then approved by EC
+project(P1).
propose-project(HospitalA,EC,P1). // P1 is 'project 1'
approve-project(EC,HospitalA,P1).
send-letter-of-approval(DIPG,HospitalA,P1).
sign-letter-of-approval(HospitalA,EC,P1).

// X1 is selected for the project (OFFERING)
+dataset(X1).
select-data(EC,HospitalA,P1,X1). // X1 is selected for project P1

// 3) access for Researcher follows project approval and selection of X1 (PROCESSING)
?Enabled(read(Researcher,X1)).
```

Listing 4.1: The eFLINT phrases (statements and queries) communicated by the Enforcement Orchestrator to the Reasoner with an indication when the communication happens during the execution of an exchange process (Scenario 1).

to (b) enable a precise selection of the registry's data needed for the research.

According to Article 6(1c) of the GDPR, processed data must be minimized with respect to the purpose of the processing. In the prototype, the registry is considered to be a simple database that holds records of patients according to an agreed upon structure and the proposal sent by a researcher contains an SQL query to be executed on this database. The custodian of the DIPG Network checks the query (similar to in the University Personnel use case of Section 4.1) to determine whether this is consistent with the proposed research and whether the selection is indeed 'minimal'. When the proposal is accepted, the custodian will make the selection available through an offer in the Catalog.

Accessing the selected data After these steps have been completed the researcher can request to download the selected data (REQUESTING). In the prototype, this request is a simple access control request (action 'read') to the database of the registry (PROCESSING). The ex-ante compliance check for this action involves several reasoning steps. The following policy fragment (from "access_union.eflint") shows that a project and member are required for which holds that the project has been approved (for the member), the actor sending the access request is affiliated with the member and the accessed asset is selected for the project. (For a full understanding of the policy fragments, the reader is referred to [3] and [2].)

```
Extend Act read Holds when (Exists project, member:
  approved(project,member) &&
  affiliated-with(actor,member) &&
  selected(asset,project))
```

The evidence that these conditions have been fulfilled is gathered by the Enforcement Orchestrator in the way suggested by the comment lines in Listing 4.1.

Reflections The condition that the processed data is minimal for the specified purpose is not automatically enforced by the Enforcement Orchestrator in the above example. The example is easily extended, however, with a policy that reflects the minimality requirement.

Member	User	Role	Component
Hospital B	steward	data provider	producer node
Registry	custodian	data consumer / storage provider	storage node

Table 4.4: The different members, users, roles and member nodes involved in Scenario 2.

The selection of data for the project by the data custodian of the registry can be considered as the qualification (by the custodian) that the selected (and subsequently processed) data is minimal with respect to the proposed project. After processing occurred, an authority may conclude that the offered data was not minimal or that the data was used for another purposes than the proposed project.

The suggested treatment of the minimality requirement is a form of 'ex-ante' enforcement based on a (potentially imperfect) qualification[8]. This extension of the example shows that a form of ex-post enforcement is required in which additional information can be provided that sheds a different light on a scenario. The Enforcement Notary is responsible for recording the policy, policy information and policy query used to judge the compliance of a particular scenario, making it possible to re-evaluate the compliance of a scenario with additional information (AUDITING). In this example, the policy is the used version of "access_union.eflint" and its dependencies, the policy information contains all but the last line of Listing 4.1, and the policy query is `Enabled(read(Researcher, X1))` (the last line of Listing 4.1). In the prototype, the Enforcement Notary keeps track of a dossier that contains these components.

As described in this section, the Policy Reasoner may be running locally at the site of the DIPG registry or centrally, hosted by a dataspace or ecosystem provider. In the former case, the *policies* (GDPR regulation, DIPG regulatory document, and the offers) need to travel from the control plane (Policy Store) to the DIPG registry site. In the latter case, the *policy information* about, for example, project approval and selected datasets, needs to be made available to the centralized Policy Reasoner via the Enforcement Orchestrator. If policy and policy information is considered to be sensitive, then centralizing (all) the reasoning or (all) the policy administration may not be desired and hybrid solutions may be needed, as we have explored in [9]. An example of sensitive policy information is consent for data processing, which is explored in the following scenario.

4.2.3 Scenario 2 – making data available

In the second scenario, a data steward of a DIPG institution prepares data for sharing with the DIPG registry (see Table 4.4 and Figure 4.10). The DIPG regulatory document lays out certain pre-conditions that need to be satisfied before this form of data sharing can occur. In particular, the document makes explicit connections with the GDPR with statements such as:

> Each Member is a controller under the GDPR with respect to the Data it enters into the DIPG Registry. In parallel, all Member jointly are the joint controllers

[8] Related to the general "qualification problem" from which it follows that no model of a situation can be complete for every possible usage of the model.

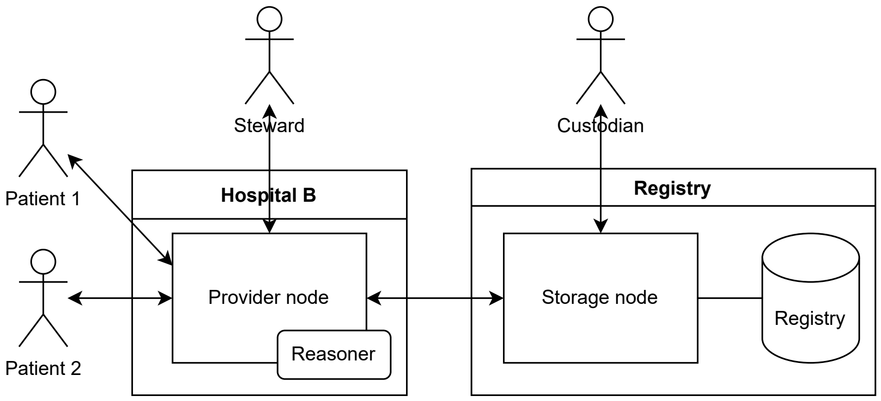


Figure 4.10: Diagram visualizing the dataspace members and users for DIPG scenario 2.

of the DIPG Registry and the aggregated Data contained therein.

The conditions focused on in the prototype for this use case are reflected in the following code fragments (from "dipg_regulatory.eflint" and "gdpr_union.eflint" respectively, see Figure 4.7).

```
Act make-data-available
  Actor member
  Recipient dcog                  // Dutch Childhood Oncology Group, on behalf of the Network
  Related to dataset              // the dataset being made available
  Conditioned by coded(dataset)   // which must be 'coded'
  Holds when member               // The sending insitution must be a member
```

```
Extend Act make-data-available
  Syncs with (Foreach donor: collect-personal-data
    (controller = member      // a DIPG member is considered a controller
    ,subject = donor          // a donor (patient) is considered a data subject
    ,data = dataset           // uniting the DIPG and GDPR notions of 'data'
    ,processor = dcog         // DCOG is considered the processor under the GDPR
    ,purpose = DIPGResearch) // the purpose of the processing is 'Research'
    When subject-of(donor, dataset)) // ensures consent is required for all donors
```

The second fragment formalizes several connections between the DIPG regulatory document and the GDPR through the application of `Syncs with`, further explained in [2], such that the `make-data-available` action effectively *inherits* all pre- and post-conditions of all the instances of the `collect-personal-data` action that it synchronizes with (one for each known donor). The following fragment (from "consent.eflint") shows a pre-condition is defined that determines consent must be given (by an identifiable subject), and that the data must be 'accurate' for the purpose for which it is collected:

```
Act collect-personal-data
  Actor controller
  Recipient subject
  Related to data, processor, purpose
  Creates processes()
  Conditioned by consent() && accurate-for-purpose()
  Holds when subject-of()
```

The 'write' action (from "access_control.eflint") synchronizes with `make-data-available` (in "access_union.eflint") to ensure these conditions are enforced when attempting to write to the DIPG registry:

```
Extend Act write Holds when
 (Exists member:                     // there exists a member institution
  affiliated-with(actor,member)) && // with which the actor attempting a write is affiliated
  Enabled(                           // and for which making data available is permitted
    make-data-available(member, DCOG, asset))
```

In conclusion, for data to be written, the following conditions need to be fulfilled:

- The actor should be affiliated with a member of the consortium
- The data should be coded – 'pseudonymized' according to the regulatory document
- Consent is given to the member by each donor for the processing of their personal data by the DCOG for the purpose of DIPG Research
- The data should be accurate for the purpose of DIPG Research

Some of these conditions are more easily checked by the steward (of a hospital) making the data available than the custodian (of the DIPG registry) receiving the data. For example, the steward may have access to the filled in consent forms of the donors. In contrast, the custodian might be a better judge (e.g. based on experience) to determine whether the data is accurate for purpose or sufficiently pseudonymized.

Related problems also appear when attempting to automate the enforcement of these conditions. Focusing on consent, the proof that a donor has provided consent may be sensitive information in its own right. For this reason, a solution may be preferred in which policy enforcement happens locally at the hospital, integrated in a *consent management system*. However, the custodian may prefer to perform their own checks, mitigating the risk that enforcement was not properly executed (e.g. consent was not given by all donors or data is not accurate for purpose). In practice, the executive committee of the DIPG requires to see the consent form templates used by members for collecting consent.

In the prototype, the scenario in Listing 4.2 is executed by a Policy Reasoner local to the hospital. The policy information is computed by running an algorithm to determine the provided data is pseudonymized (coded), to check that consent has been given by donors, and by simply asserting that the data is accurate for purpose. In an implementation of this use case we suggest to use local reasoning both at the site of the hospital providing data as well as at the site of the DIPG registry. In this case, both sites can use different qualifications of the facts that determine the satisfaction of the pre-conditions. For example, explicit consent may be required at the hospital site, whereas the availability of a consent form template may be considered sufficient at the DIPG Registry site. Similarly, both sites may run a different analysis algorithm to determine the accuracy of the data.

Reflections The focus on this use case has been policy administration and enforcement under different conditions. An important consideration is also that the DIPG Network is international and contains members both within the European Union and within the United States. Data sharing across between partners in both jurisdictions implies having to deal with different regulations, e.g. regarding privacy. To which extent this is possible and can be automated is both a legal and technical question requiring further investigation.

```
// 1) information about the hospital, obtained during ONBOARDING
+member(HospitalB).
+affiliated-with(Steward, HospitalB).

// 2) information about the patients/donors, e.g. from a consent management system
+donor(Alice). +donor(Bob). // Alice and Bob are patients, identifiable in the dataset
give-consent(Alice,HospitalB,DCOG,DIPGResearch).// Alice has given consent
give-consent(Bob,HospitalB,DCOG,DIPGResearch).  // Bob has given consent

// 3) information about the dataset (X1), asserted by the hospital
+dataset(X1).
+accurate-for-purpose(X1, DIPGResearch).
+coded(X1).
+subject-of(Alice,X1).
+subject-of(Bob,X1).

?Enabled(write(Steward,X1)).
```

Listing 4.2: The eFLINT phrases (statements and queries) communicated by the Enforcement Orchestrator to the Reasoner running locally at a hospital making data available to the DIPG Registry (Scenario 2).

The descriptions in this section have focused on the *ex-ante* compliance questions presented to the Reasoner formalized as `?Enabled(read(Researcher,X1))` in scenario 1 and `?Enabled(write(Steward,X1))` in scenario 2. However, this use case also demonstrates the importance of *ex-post* enforcement.

Firstly, by using data from the DIPG Registry, a researcher accepts the following post-condition: "The Researcher shall not disclose or provide access to the Data to any third party without the prior written consent of Executive Committee". A post condition like this can not be automatically enforced. Instead, we imagine a mechanism by which observations affecting the compliance of actions with such post-conditions can be brought into the system by a dataspace member, e.g. communicating that a researcher has unduly shared data with a third party.

Secondly, the (ex-ante) decisions of compliance with conditions should be accountable. In the prototype this is achieved by keeping dossiers of all policies and policy information used in compliance decisions. However, the members themselves should be accountable, e.g. keeping records of the consent preferences of their patients or of the pseudonymization algorithm used for 'coding'. A consent management system in which consent can be given and withdrawn by patients dynamically, and recorded by administrators, has been considered out of scope for this prototype.

4.3 Research Data Exchange

The Research Institute for Child Development and Education (RICDE) of the University of Amsterdam (UvA) employs about 200 researchers in the field of social and behavioral sciences. Most of the research analysis is based on collected datasets, and most of these datasets are confidential in order to protect the privacy of the participants in the studies.

The Netherlands strives to Open Science, where scientific knowledge (scientific publications, research data, meta-data, educational resources, software and source code, and open hardware) is openly available [13], and where datasets are Findable, Accessible, Interoperable, and Reusable (FAIR). However, this gives rise to the Open Science Dilemma: how to make confidential datasets accessible?

On the one hand Open Science advocates for the dissemination of as much data as possible on an open platform to promote scientific progress, enable transparency, and allow for the replication of analyses. Additionally, given that scientific research is often funded with public money, it is important to make research results and data accessible to the public. On the other hand, legal and sovereignty issues can limit the extent to which data can be openly shared. It is important to maintain control over confidential datasets, which can entail legal issues such as ownership and copyright, confidentiality of personal data, restrictions on informed consent letters, purpose limitations, bans on dual use, and prohibitions on resale.

4.3.1 Secure Analysis Environment (SANE)

The Secure Analysis Environment (SANE) is a Trusted Research Environment (TRE) developed by SURF in collaboration with ODISSEI and CLARIAH, and sponsored by PDI-SSH. ODISSEI and CLARIAH are two organisations that provide research infrastructures for social sciences and humanities, respectively. SANE is an variant of an compute-to-data archetype, the "Sharing Data via Trusted Third Party (TTP)" archetype, as defined in Section 3.2 (see Figure 3.3). In this archetype, SURF acts as a third party which performs the computation under control of the data owner, with the software provider (also) the data consumer (see also the third scenario of the UNL case in Section 4.1.3). Figure 4.11 gives a visual representation of the different actors in this archetype.

A Trusted Research Environment such as SANE provides control to the data owner, because the data is only made available in a secure environment. It can not be downloaded. Hence, it allows for the re-use of confidential data while ensuring its confidentiality. However, the current process requires a manual effort each time a researcher wishes to utilize a confidential dataset provided by someone else. This process is depicted in Figure 4.12. In this process, the researcher, acting as the data consumer, must locate the dataset on existing data repositories (e.g., DANS or OSF). Unfortunately, the lack of a download option for confidential datasets means that the only available course of action is to communicate via email with the data owner and hope that they are willing to either provide the dataset for download or make it accessible within a secure analysis environment. This method is not only tedious and time-consuming, but it also places a burden on the data owners.

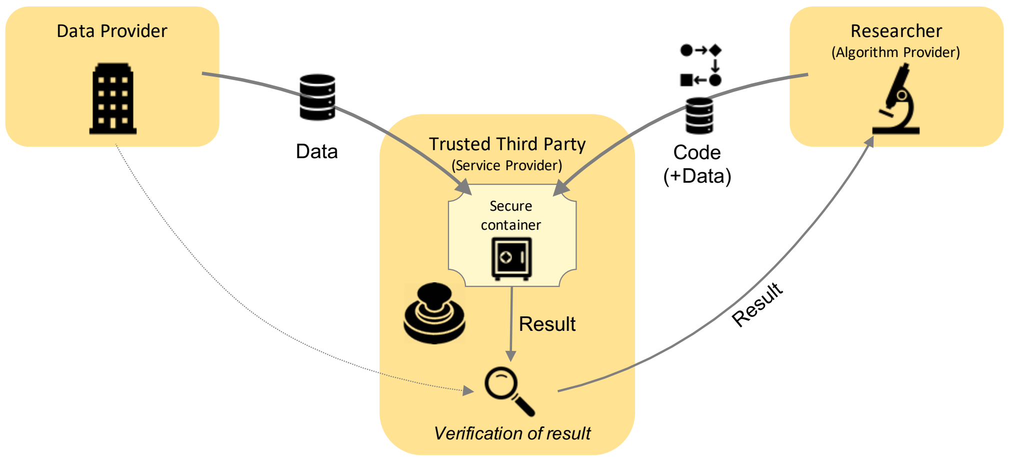
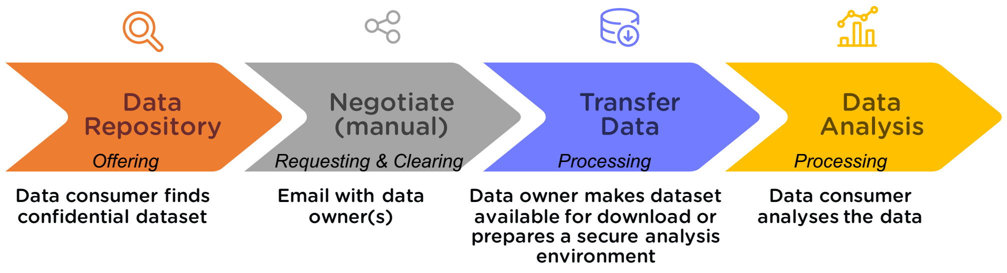


Figure 4.11: In the "Sharing Data via Trusted Third Party (TTP)" archetype, the Data Provider provides the data, and the Algorithm Provider provides the algorithm. The algorithm is run in a secure container at a Service Provider, and the output of this computation is first verified by the data provider before to ensure it does not contain any confidential information, before this output is released to the Algorithm Provider.



Data Repository
Offering
Data consumer finds confidential dataset
Negotiate (manual)
Requesting & Clearing
Email with data owner(s)
Transfer Data
Processing
Data owner makes dataset available for download or prepares a secure analysis environment
Data Analysis
Processing
Data consumer analyses the data

Figure 4.12: Manual workflow for re-use of confidential data by a researcher. The terms *Offering*, *Requesting*, *Clearing*, and *Processing* refer to the processing steps in Section 2.3.2.

4.3.2 Goals

Research Data Exchange (RDX) is a prototype that automates the process of making data available for re-use. This prototype served two purposes:

- Get experience on the usability of connecting a data repository to a trusted research environment, and get an understanding of the different roles involved (data owner, data provider, data steward, data consumer, service provider);
- Get insight in which data sharing conditions are desired in practise.

The research data exchange splits the workflow in two parts: one for the data owners (depicted in Figure 4.13) and one for the data consumers (depicted in Figure 4.14).

With RDX, a data owner can specify the data sharing conditions for each dataset (see Section 4.3.3) and makes the dataset available for re-use. This only needs to be done once, at the same time the (meta-)data is published on a data repository to make it findable.

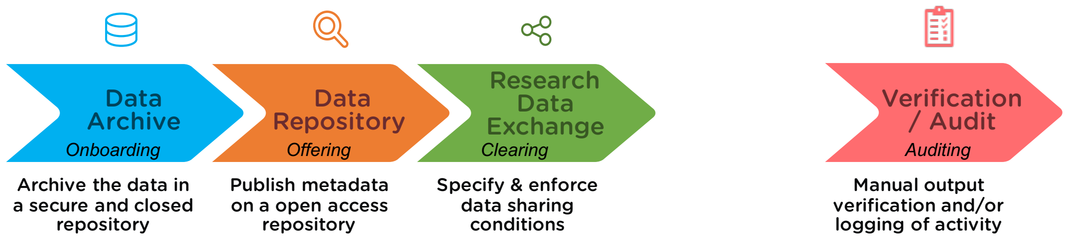

Data Archive
Onboarding
Archive the data in a secure and closed repository
Data Repository
Offering
Publish metadata on a open access repository
Research Data Exchange
Clearing
Specify & enforce data sharing conditions
Verification / Audit
Auditing
Manual output verification and/or logging of activity

Figure 4.13: Publication workflow (for data owners).

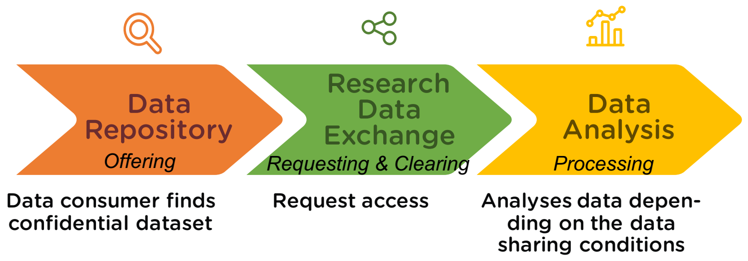


Figure 4.14: Re-use workflow (for data consumers).

If desired by the data owner, there is an option to perform manual output verification for each analysis conducted by a data consumer, as well as access the records of previous analyses conducted on the dataset. This level of control is crucial as it empowers the data owner to maintain complete oversight of the confidential dataset and its usage.

When a researcher is interested in re-using a dataset, they can still locate the (meta)data on an existing repository. However, instead of engaging in negotiations with the data owner for access, the RDX prototype automatically enforces access permissions. Depending on the data sharing conditions, the data consumer must first prove its affiliation (e.g., be part of an existing research community) and agree to the designated sharing conditions (e.g., non-commercial use or citation requirements). Once these conditions are met, the data consumer can proceed to download the data or conduct analyses within a secure analysis environment. The specific actions allowed are contingent upon the data sharing conditions established by the data owner in the publication workflow.

4.3.3 Data Sharing Conditions

Based on small-scale interviews with potential users, and earlier experience, we can list the following data sharing conditions. The list is roughly ordered from most important to least important.

- Verify identity and affiliation
- Sign data sharing conditions
  - Do not redistribute the dataset
  - Purpose-binding; dataset may only be used to answer a specific (pre-aproved) research questions

  - Give credit when data is re-used
  - Involvement in the analysis and co-authorship
- Verification of the analysis output before releasing the output
- Verification that the resulting paper does not contain confidential information
- Analyse in a secure analysis environment only
- Only allow verified algorithms for analysis

The prototype was able to enforce the following conditions:

- Verify email address
- Sign data sharing conditions
- Only make the dataset available after automatic verification of the above conditions
- No download allowed, analyse in a secure analysis environment only (optional)
- Verification of the analysis output before releasing the output (optional)

In total, five combinations of data sharing conditions are supported:

- Make available for download, after verification of email address and signing data sharing conditions;
- Make available for processing in a "tinker" secure analysis environment, after verification of email address and signing data sharing conditions, either with or without verification of the output by the data provider;
- Make available for processing in a "blind" secure analysis environment, after verification of email address and signing data sharing conditions, either with or without verification of the output by the data provider.

The tinker- and blind secure analysis environment were provided by the SANE pilot. The difference between the two is that the tinker SANE spins up a is a job submission process, where the data is made available in a remote desktop Windows environment that contains the data, and data processing tools (for the prototype R Studio and Jupyter notebook). For the blind SANE, the data consumer must specify a Python script that is executed in a VM were the data is made available. In both cases, the VM does not allow network connections, so it is not possible to upload the dataset elsewhere. The result of the analysis can only be uploaded to a specific output, where the result is logged and is optional verified by the data owner before it is released.

4.3.4 Findings

Based on user interviews and demonstrations with the prototype, we came to the following conclusions.

The prototype works as intended. We ran a few tests, and got positive responses, in particular from the University of Amsterdam. However, the technology is still new, and the concept of data-reuse, without data download is still novel. We found that the SANE pilot is getting noted by data stewards and digital competence centres. However, most individual researchers do not yet know about this possibility, let alone other privacy enhancing technologies (PETs) that are available nowadays. By extension, the RDX prototype was new as well for most researchers. The prototype proved to be of benefit as a conversation starter, that helps explain the possibilities that PETs offer.

Both data steward and data owner want to be involved. We found that both the data owners (the researcher who made a confidential dataset) as well as the data steward (a supporting role within the organisation, acting as the data provider or data custodian in the prototype) want to know who is using the data. The data owner primarily from the perspective of a personal responsibility to protect the privacy of the people who contributed their data to the study, and by extension, to the dataset. The data steward more from the perspective of GDPR compliance. We anticipate that when this technology is used more extensively, in particular the data steward wants to have a hand-offs, mostly overseeing role, and that the data owner wants to remain involved, if only as to seek potential collaborations with the people interested in their datasets.

Data owners feel involved in their dataset. This is perhaps not surprising, as they often spend a significant amount of work collecting the data, and feel personally responsible for protecting the privacy of the people involved. It can be argued that legally, the data belongs to the university and not the researcher who collected the data, and that for reasons of data provenance and data preservation, the data steward should have a more prominent role (think about the situation when the researchers finds a job elsewhere or retires). Despite professionalization of the data steward role in the last years, and appreciation of that role by researchers, the researchers still consider themselves the owner of "their" datasets.

For data owners, trust in the data consumer is paramount. Trust does not simply come from an affiliation, such as being affiliated with an university. At first glance, this seems like a subjective criterion, but upon further questioning, it was stipulated that "a fellow researcher who has published in high-impact journals before" can be trusted.

Logging may be used as alternative to verification. The prototype allows for the option that the data owner or the data provider verifies that the result of each analysis no longer contains any personal information. However, neither party seemed to view this as the ideal situation. Data stewards seem to feel that logging of the actions and output is sufficient. That would take away the burden of manually checking of each analysis, and still provides a means to correct data consumers that do not adhere to the data sharing conditions, even if it is after the fact. The data owners also seem to trust the consumer with the data, and do not need to check the output of each analysis. However, they do want to check the resulting publication(s) before they are published, since any recognizable data in a publication is a severe breach in privacy.

The prototype is not designed to protect against malicious actors. This is acceptable by data owners and data providers. That does not mean there is no concern, but the concern mostly stems from (a) risk of leaking identifiable information in a publication; or (b) risk of reputation damage, because a data consumers makes errors in the analysis or draws a faulty conclusion; or (c) risk of reputation damage, because the compute-to-data archetype is not well understood, and the general public does not distinguish between re-use of data and redistribution of data.

A secure analysis environment may be most relevant to larger data providers. For individual use cases, most data owners seem to prefer individually checking the reputation of the data consumer, and if sound, let them simply download the data. Only when the data reuse is more frequently reused, and individual check can not be as thorough, an automatic system of protection, as provided by the secure analysis environment, may partially replace this manual check.

Automatic publication of datasets is not yet solved. In the prototype, the computation was done by a trusted third party, SURF, who had to be trusted by the data provider to protect the data. However, this also means there must be a way for the data provider to make the confidential data sets available to the third party. In the prototype, this was a manual process. For actual deployment, this must be automated.

Bibliography

[1] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. “Introduction to Runtime Verification”. In: *Lectures on Runtime Verification: Introductory and Advanced Topics*. Ed. by Ezio Bartocci and Yliès Falcone. Cham: Springer International Publishing, 2018, pp. 1–33. ISBN: 978-3-319-75632-5. DOI: `10.1007/978-3-319-75632-5_1`.

[2] L. Thomas van Binsbergen, Milen G. Kebede, Joshua Baugh, Tom van Engers, and Dannis G. van Vuurden. “Dynamic generation of access control policies from social policies”. In: *Procedia Computer Science* 198 (2022). 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare, pp. 140–147. ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2021.12.221`.

[3] L. Thomas van Binsbergen, Lu-Chi Liu, Robert van Doesburg, and Tom van Engers. “eFLINT: A Domain-Specific Language for Executable Norm Specifications”. In: *Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. GPCE 2020. ACM, 2020, pp. 124–136. DOI: `10.1145/3425898.3426958`.

[4] L. Thomas van Binsbergen, Merrick Oost-Rosengren, Hayo Schreijer, Freek Dijkstra, and Taco van Dijk. *AMdEX Reference Architecture – version 1.0.0*. Ed. by L. Thomas van Binsbergen. Feb. 2024. DOI: `10.5281/zenodo.10565915`.

[5] Data Sharing Coalition. *Data Sharing Canvas – A stepping stone towards cross-domain data sharing at scale*. 2021.

[6] Council of the EU. “General Data Protection Regulation”. In: *Official Journal of the European Union* 59 (2016).

[7] R. van Doesburg and T. van Engers. “The False, the Former, and the Parish Priest”. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law*. ICAIL 2019. ACM, 2019, pp. 194–198. DOI: `10.1145/3322640.3326718`.

[8] R. van Doesburg, T. van der Storm, and T. van Engers. “CALCULEMUS: Towards a Formal Language for the Interpretation of Normative Systems”. In: *AI4J Workshop at ECAI 2016*. AI4J 2016. 2016, pp. 73–77.

[9] Christopher A. Esterhuyse, Tim Müller, L. Thomas Van Binsbergen, and Adam S. Z. Belloum. "Exploring the Enforcement of Private, Dynamic Policies on Medical Workflow Execution". In: *2022 IEEE 18th International Conference on e-Science (e-Science).* 2022, pp. 481–486. DOI: 10.1109/eScience55777.2022.00086.

[10] Adrian Francalanza, Jorge A. Pérez, and César Sánchez. "Runtime Verification for Decentralised and Distributed Systems". In: *Lectures on Runtime Verification: Introductory and Advanced Topics*. Ed. by Ezio Bartocci and Yliès Falcone. Cham: Springer International Publishing, 2018, pp. 176–210. ISBN: 978-3-319-75632-5. DOI: 10.1007/978-3-319-75632-5_6.

[11] André Freitas, João Gabriel Oliveira, Seán O'Riain, João C.P. da Silva, and Edward Curry. "Querying linked data graphs using semantic relatedness: A vocabulary independent approach". In: *Data & Knowledge Engineering* 88 (2013), pp. 126–141. ISSN: 0169-023X. DOI: 10.1016/j.datak.2013.08.003.

[12] Florine de Geus. "Model Checking Normative Systems". University of Amsterdam, 2022. URL: https://scripties.uba.uva.nl/search?id=record_51722.

[13] Stan Gielen and Jet de Ranitz. *Open Science 2030 in the Netherlands: NPOS2030 Ambition Document and Rolling Agenda*. Dec. 2022. DOI: 10.5281/zenodo.5105529.

[14] Giulia Giussani, Sebastian Steinbuss, Mario Holesch, and Nora Gras. *Data Connector Report*. International Data Spaces Association, Jan. 2024. DOI: 10.5281/zenodo.10591027.

[15] Ali Hariri, Amjad Ibrahim, Bithin Alangot, Subhajit Bandopadhyay, Antonio La Marra, Alessandro Rosetti, Hussein Joumaa, and Theo Dimitrakos. "UCON+: Comprehensive Model, Architecture and Implementation for Usage Control and Continuous Authorization". In: *Collaborative Approaches for Cyber Security in Cyber-Physical Systems*. Ed. by Theo Dimitrakos, Javier Lopez, and Fabio Martinelli. Cham: Springer International Publishing, 2023, pp. 209–226. DOI: 10.1007/978-3-031-16088-2_10.

[16] H. Herrestad. "Norms and Formalization". In: *Proceedings of the 3th International Conference on Artificial Intelligence and Law*. ICAIL 1993. ACM, 1993, pp. 175–184. DOI: 10.1145/112646.112667.

[17] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. "A Policy Language for Distributed Usage Control". In: *Computer Security – ESORICS 2007*. Ed. by Joachim Biskup and Javier López. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 531–546. DOI: 10.1007/978-3-540-74835-9_35.

[18] Wesley Newcomb Hohfeld. "Some Fundamental Legal Conceptions as Applied in Judicial Reasoning". In: *The Yale Law Journal* 23.1 (Nov. 1913), pp. 16–16. DOI: 10.2307/785533.

[19] Renato Iannella and Serena Villata. "ODRL information model 2.2". In: *W3C Recommendation* 15 (2018).

[20] Christian Jung and Jörg Dörr. “Data Usage Control”. In: *Designing Data Spaces : The Ecosystem Approach to Competitive Advantage*. Ed. by Boris Otto, Michael ten Hompel, and Stefan Wrobel. Cham: Springer International Publishing, 2022, pp. 129–146. ISBN: 978-3-030-93975-5. DOI: 10.1007/978-3-030-93975-5_8.

[21] Jonathan Karels. “Design of a Framework for Storing and Reasoning About Normative Sources with Formalised Interpretations”. University of Amsterdam, 2023. URL: https://scripties.uba.uva.nl/search?id=record_53692.

[22] Milen G. Kebede, Giovanni Sileno, and Tom Van Engers. “A Critical Reflection on ODRL”. In: *AI Approaches to the Complexity of Legal Systems XI-XII*. Ed. by Víctor Rodríguez-Doncel, Monica Palmirani, Michał Araszkiewicz, Pompeu Casanovas, Ugo Pagallo, and Giovanni Sartor. Cham: Springer International Publishing, 2021, pp. 48–61. DOI: 10.1007/978-3-030-89811-3_4.

[23] Ryan K. L. Ko, Bu Sung Lee, and Siani Pearson. “Towards Achieving Accountability, Auditability and Trust in Cloud Computing”. In: *Advances in Computing and Communications*. Ed. by Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, and Sabu M. Thampi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 432–444. DOI: 10.1007/978-3-642-22726-4_45.

[24] OASIS eXtensible Access Control Markup Language (XACML) Technical Committee. *eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01*. July 2017.

[25] B. Otto, S. Steinbuss, A. Teuscher, and S Lohmann. *IDS Reference Architecture Model – version 3.0*. Apr. 2019. DOI: 10.5281/zenodo.5105529.

[26] Lúcio H. A. Reis, Marcela T. de Oliveira, and Sílvia D. Olabarriaga. “Fine-grained Encryption for Secure Research Data Sharing”. In: *2022 IEEE 35th International Symposium on Computer-Based Medical Systems (CBMS)*. 2022, pp. 465–470. DOI: 10.1109/CBMS55023.2022.00089.

[27] Pepijn de Reus, Ana Oprescu, and Koen van Elsen. “Energy Cost and Machine Learning Accuracy Impact of k-Anonymisation and Synthetic Data Techniques”. In: *2023 International Conference on ICT for Sustainability (ICT4S)*. 2023, pp. 57–65. DOI: 10.1109/ICT4S58814.2023.00015.

[28] Ravi Sandhu and Jaehong Park. “Usage Control: A Vision for Next Generation Access Control”. In: *Computer Network Security*. Ed. by Vladimir Gorodetsky, Leonard Popyack, and Victor Skormin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 17–31. DOI: 10.1007/978-3-540-45215-7_2.

[29] Sara Shakeri, Lourens Veen, and Paola Grosso. “Evaluation of Container Overlays for Secure Data Sharing”. In: *2020 IEEE 45th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*. 2020, pp. 99–108. DOI: 10.1109/LCNSymposium50271.2020.9363266.

[30] SIOPE DIPG Network. “DIPG Registry and Imaging Repository – Regulatory Document”. Online. [Online, accessed 1st July 2021]. Oct. 2018. URL: https://dipgregistry.eu/Content/files/2018-10-10SIOPEDIPGRegistry-RegulatoryDocument_v%202.0_final.pdf.

[31] Jorrit Stutterheim. “DYNAMOS: Dynamically Adaptive Microservice-based OS; A Middleware for Data Exchange Systems”. University of Amsterdam, 2023. URL: https://scripties.uba.uva.nl/search?id=record_53890.

[32] Elham Ramezani Taghiabadi, Vladimir Gromov, Dirk Fahland, and Wil M. P. van der Aalst. “Compliance Checking of Data-Aware and Resource-Aware Compliance Requirements”. In: *On the Move to Meaningful Internet Systems: OTM 2014 Conferences.* Ed. by Robert Meersman, Hervé Panetto, Tharam Dillon, Michele Missikoff, Lin Liu, Oscar Pastor, Alfredo Cuzzocrea, and Timos Sellis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 237–257. DOI: 10.1007/978-3-662-45563-0_14.

[33] Lu Zhang, Reginald Cushing, Leon Gommans, Cees De Laat, and Paola Grosso. “Modeling of Collaboration Archetypes in Digital Market Places”. In: *IEEE Access* 7 (2019), pp. 102689–102700. DOI: 10.1109/ACCESS.2019.2931762.

[34] Xin Zhou, Reginald Cushing, Ralph Koning, Adam Belloum, Paola Grosso, Sander Klous, Tom van Engers, and Cees de Laat. “Policy Enforcement for Secure and Trustworthy Data Sharing in Multi-domain Infrastructures”. In: *2020 IEEE 14th International Conference on Big Data Science and Engineering (BigDataSE).* 2020, pp. 104–113. DOI: 10.1109/BigDataSE50710.2020.00022.

Index