SEcure Decentralised Intelligent Data MARKetplace

D3.3 Enabling tools for data interoperability, distributed data storage and training distributed AI models. First version.

Document Identification	
Contractual delivery date:	31/12/2023
Actual delivery date:	18/01/2024
Responsible beneficiary:	EGM
Contributing beneficiaries:	EGM, INRIA, LINKS, UC, SURREY.
Dissemination level:	PU
Version:	1.0
Status:	Review

Keywords:
Data, model, interoperability

This document is issued within the frame and for the purpose of the SEDIMARK project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No.101070074. and is also partly funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission or UKRI.

Document Information

Document Identification			
Related WP	WP3	**Related Deliverables(s):**	D3.1
Document reference:	SEDIMARK_D3.3	**Total number of pages:**	63

List of Contributors	
Name	**Partner**
Gilles Orazi Léa Robert Franck Le Gall Luc Gasser	EGM
Maroua Bahri Nikolaos Georgantas	INRIA
Alberto Carelli Andrea Vesco	LINKS
Juan Ramón Santana Pablo Sotres Víctor González Jorge Lanza Luis Sánchez	UC
Tarek Elsaleh	SURREY
Diarmuid O'Reilly Morgan Erika Duriakova Honghui Du Elias Tragos Qinqin Wang Aonghus Lawlor Neil Hurley	NUID UCD

Document History			
Version	**Date**	**Change editors**	**Change**
0.1	01/09/2023	EGM	Creation
0.2	14/11/2023	NUID UCD	Contribution to Section 4.3.1
0.3	27/11/2023	LINKS	Add contributions in Sect. 6
0.3	29/11/2023	UC, SURREY	Add contributions to Section 3 and 4
0.4	22/12/2023	EGM	Overall review of comments and cleaning.
0.5	18/12/2023	ATOS	Quality Format Review
1.0	10/01/2024	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	**Who (Partner short name)**	**Approval date**
Reviewer 2	Alberto Carelli (LINKS)	15.12.2023
Reviewer 1	Grigorios Koutantos (WINGS)	15.12.2023
Quality manager	María Guadalupe Rodríguez (ATOS)	10.01.2024
Project Coordinator	Arturo Medela (ATOS)	17.01.2024

Table of Content

List of Figures

List of Tables

List of Acronyms

Abbreviation / Acronym	Description
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CSV	Comma Separated Values
DCAT	Data Catalog Vocabulary
DCT	DC (Dublin Core) Term
DLT	Distributed ledger technology
Dx.y	Deliverable number y belonging to WP x
ETL	Extract Transform Load
ETSI	European Telecom Standards Institute
EVM	Ethereum Virtual Machine
FL	Federated Learning
FOAF	Friend Of A Friend
FTP	File Transfer Protocol
GL	Gossip Learning
gRPC	Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
IDS	International Data Spaces
IDSA	International Data Spaces Association
IEEE	Institute of Electrical and Electronics Engineers
INX	IOTA Node Extension (INX) interface
ISC VM	IOTA Smart Contracts Virtual Machine
ISG CIM	Industry Specification Group for Context Information Management
JSON-LD	JavaScript Object Notation for Linked Data
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport

Abbreviation / Acronym	Description
MPC	Multi-Party Computation
NGSI-LD	Next Generation Service Interfaces for Linked Data
ODRL	ODRL
ONNX	Open Neural Network Exchange
OWL	Web Ontology Language
POI	Proof of Inclusion
PSI-CA	Private Set Intersection Cardinality Protocol
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
SDM	Smart Data Models
SPARQL	SPARQL Protocol and RDF Query Language
SSE-C	server-side encryption with customer-provided keys
URI	Uniform Resource Identifier
WASM	WebAssembly
WLAN	Wireless Local Area Network
WP	Work Package
XML	eXtensible Markup Language

Executive Summary

SEDIMARK_D3.3 “Enabling tools for data interoperability, distributed data storage and training distributed AI models” is a report produced by the SEDIMARK project. It aims at providing insights in relation to progress made for:

- Management of interoperability in data flows by proper definition and usage of metadata and semantic technologies
- Distributed storage considering interoperability and scalability issues of such storages.
- Finally, on the use of distributed AI and analytics, considering in particular federated learning,

Description is purposedly made short as this report is meant to accompany the implementations developed in the first period of the project and guide the user in understanding the rationale behind the tools selections and configurations as they exist in the project git repository.

The content of the report is built as follows:

- Section 1 provides introduction to the document, describing how the different sections relates to the elements of the SEDIMARK architecture.
- Section 2 describes the interoperability approach chosen for assets. It provides an overview of the related specification (NGSI-LD from ETSI ISG CIM) as well as the community driven smart data models initiative used for the data asset and describes the models used for the Marketplace offers and assets descriptions.
- Section 3 discusses how data points and datasets get annotated to report their quality level or encompassed processes.
- Section 4 dives into the development and training of AI models focusing on distributed models training using federated learning methods. Two frameworks (Fleviden and shamrock) deployed in the project are presented.
- Finally, sections 5 and 6 discuss the storage layer with section 5 describing an IOTA based distributed storage layer and section 6 focused on storage meant for scalability and semantic interoperability.

1. Introduction

1.1 Purpose of the document

Data quality is considered to be of the highest importance for companies to improve their decision-making systems and the efficiency of their products. In this current data-driven era, it is important to understand the effect that “dirty” or low-quality data (i.e., data that is inaccurate, incomplete, inconsistent, or contains errors) can have on a business. Manual data cleaning is the common way to process data, accounting for more than 50% of the time of knowledge workers. SEDIMARK acknowledges the importance of data quality for both sharing and using/analysing data to extract knowledge and information for decision-making processes.

Common types of dirty or low-quality data include:

1. Missing Data: Incomplete datasets where certain values are not recorded.
2. Inaccurate Data: Data that contains errors, inaccuracies, or typos. This can happen due to manual data entry errors or system malfunctions.
3. Inconsistent Data: Data that is inconsistent across different sources or within the same dataset. For example, the same entity may be represented in different ways (e.g., "Mr. Smith" vs. "Smith, John").
4. Duplicate Data: Repetition of the same data in a dataset, which can distort analyses and lead to incorrect results.
5. Outliers: Data points that deviate significantly from the majority of the dataset, potentially skewing the analysis.
6. Bias: Data that reflects systematic errors due to a particular group's overrepresentation or underrepresentation in the dataset.
7. Unstructured Data: Data that lacks a predefined data model or organization, making it difficult to analyse.

Thus, one of the main work items of SEDIMARK is to develop a usable data processing pipeline that assesses and improves the quality of data generated and shared by the SEDIMARK data providers.

1.2 Relation to another project work

This deliverable presents the work related to the components meant to support interoperability, distributed storage, and system intelligence. These are three main pillars within SEDIMARK to support the project objectives for enabling seamless data sharing between consumers and providers. Interoperability is a key part of SEDIMARK to enable the efficient and easy reuse of datasets, models, and services across the whole network of SEDIMARK participants, aiming i.e., to support them in integrating data from different sources to train more advanced and robust models or to enable the distributed training of machine learning models on compatible datasets.

1.3 Structure of the document

Figure 1 presents the SEDIMARK functional architecture that was described in Deliverable SEDIMARK_D2.2 in detail. With orange highlights are the functional components that are part of this deliverable. As it is evident, these components are split between the data layer and the

Intelligence layer of SEDIMARK, with one component from the Service layer. More details about these components and their mapping to sections in the deliverable are given below:

- Data formatting: this component is described in Section 2.2 and is related to discussing the ontologies and the data models that will be used for the internal (within the data quality pipeline) and external format of the data (the format to be used during data sharing).
- Offering description: this component is part of section 2.3, discussing the format of the offerings that will be used to describe the assets to be shared within the marketplace.
- Semantic enrichment, Data quality annotation, Model annotation, ML-oriented data quality annotation, AI model quality annotation, AI model validation, data validation/certification: these components are related to functionalities presented in section 4, discussing the interoperability enabler.
- Data analytics, Model inference, Local model training, AI orchestration: these are components of the AI layer described in section 4.1, showing how SEDIMARK supports the providers and the consumers to train their own models and use them for inference and analytics.
- Distributed model training: this component is split into two parts and presented in sections 4.2 and 4.3, discussing the two different and complementary frameworks developed within SEDIMARK to allow joint training of ML models between distant participants.
- Interaction, transactions: these components are related to the Interaction layer of SEDIMARK and are described in section 5.
- Local storage, distributed storage: these are components of the Storage layer of SEDIMARK and are described in section 6, discussing the various options available within SEDIMARK for supporting distributed storage of data and assets.

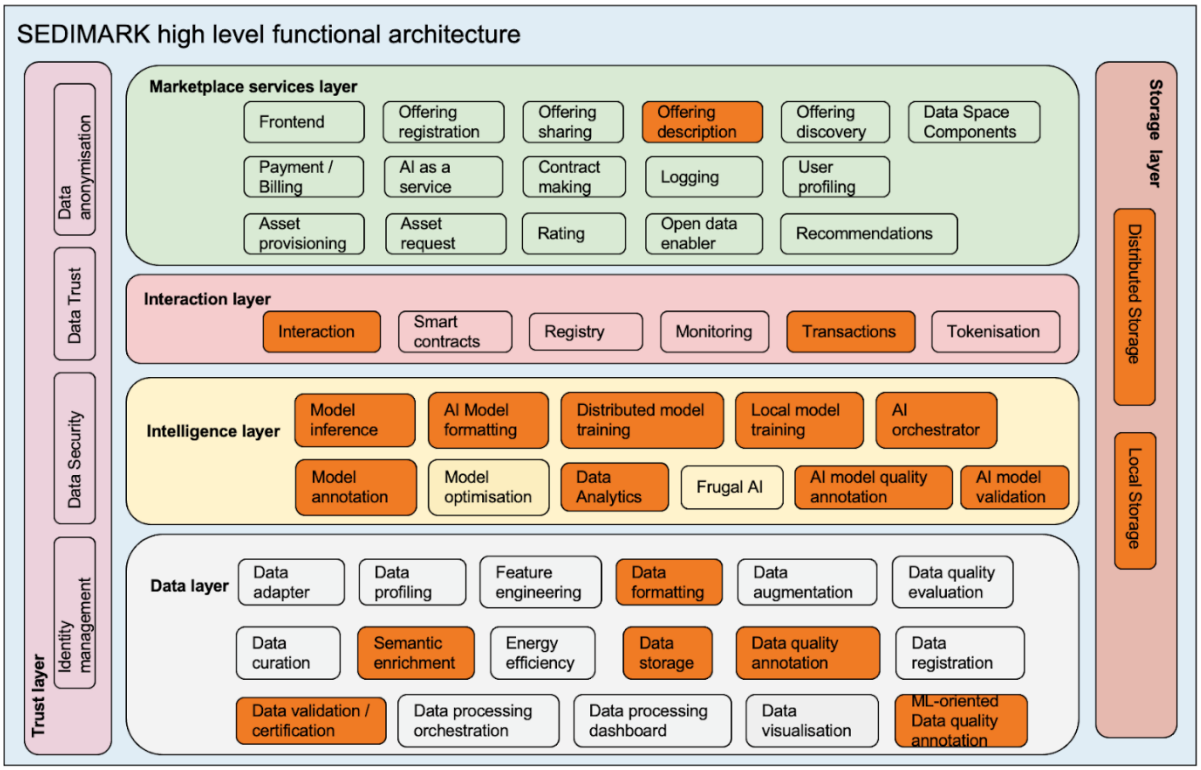


Figure 1: SEDIMARK Functional Architecture: orange highlights functional components that are being part of this deliverable

2 Interoperability assets

2.1 Introduction

Interoperability is a crucial facet of modern information management, enabling seamless communication and exchange of data, models and services across diverse systems, platforms, and applications. In a world characterized by an abundance of data sources and formats, achieving interoperability ensures that disparate systems can understand, interpret, and effectively use shared data. This capability facilitates collaboration, integration, and synergy among organizations and technologies, breaking down silos and promoting a more interconnected digital ecosystem.

This introduction explores the significance of data interoperability in overcoming the challenges posed by data heterogeneity, promoting standardization, and ultimately unlocking the full potential of interconnected data landscapes. Embracing data interoperability not only enhances operational efficiency but also lays the groundwork for advanced analytics, artificial intelligence, and the seamless flow of information in our interconnected, data-driven world.

Before initiating any data processing pipeline within the SEDIMARK platform, data thus need to be formatted so to be usable by the pipeline. In its initial version, it has been agreed that the data processing pipeline would consumes and produces data organised along the NGSI-LD information model [1].

2.2 Assets information models

2.2.1 NGSI-LD as base format

NGSI-LD is represented in JSON-LD and thus should have a grounding in RDF. It is mainly based on RDF standards to capture high-level relations between entities (representing or not a real-world asset) and properties of entities, as shown below. The core concept in the NGSI-LD data model is the “Entity” which can have properties and relationships to other entities. An entity is equivalent to an OWL class. The assumption is that the world consists of entities, which can be physical entities like a car or a building, but also more abstract entities like a company or the coverage area of WLAN access points. Entity instances are identified by a unique URI and a type, e.g., a sensor with identifier *urn:ngsi-ld:Sensor:01* and of type Sensor. Different from *rdf:Properties*, NGSI-LD properties (and relationship) are also considered as OWL classes also. Properties and relationships can be annotated by properties and relationships themselves, e.g., a timestamp, the provenance of the information or the quality of the information can be provided. The *hasObject* and *hasValuein* the NGSI-LD metamodel are defined to enable RDF reification, based on the blank node pattern, to leverage the property graph model.

The NGSI-LD cross domain ontology extends the NGSI-LD metamodel to cover several general contexts presented below [2]:

- Mobility defines the stationary, movable or mobile characteristics of an entity;
- Location differentiates and provides concepts to model the coordination based, set based or graph-based location;
- Temporal specification includes property and values for temporal property definitions;

- Behavioural system includes properties and values to describe system state, measurement and reliability;
- System composition and grouping provides a way to model system of systems in which small systems are composed together to form a complex system following specific patterns.

The NGSI-LD cross domain ontology is presented in Figure 2.

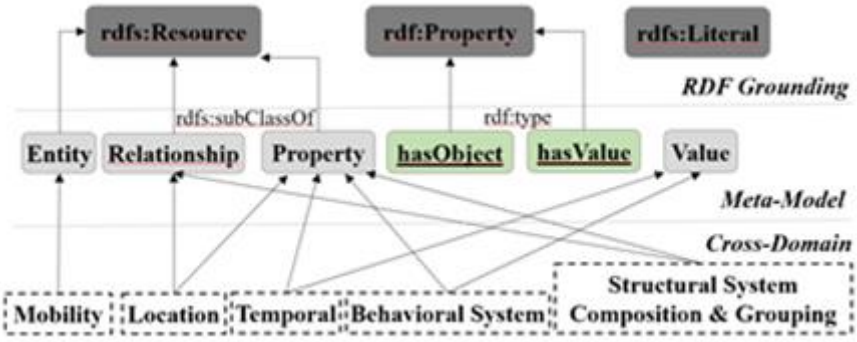


Figure 2: RDF standards to capture high-level relations between entities in NGSI-LD

Below we present a use case example for modelling data and context using the NGSI-LD. The example consists of a station that returns the measure of the level and flow of a river. This station has an id which is *urn:ngsi-ld:Hydrometric-Station:X061000201*.This station is located in a river identified by *urn:ngsi-ld:River:La_Durance.* This is defined by the relationship (*isLocatedOn*).

To model this example, Figure 3 presents the main symbols signification used in the medialisation task.

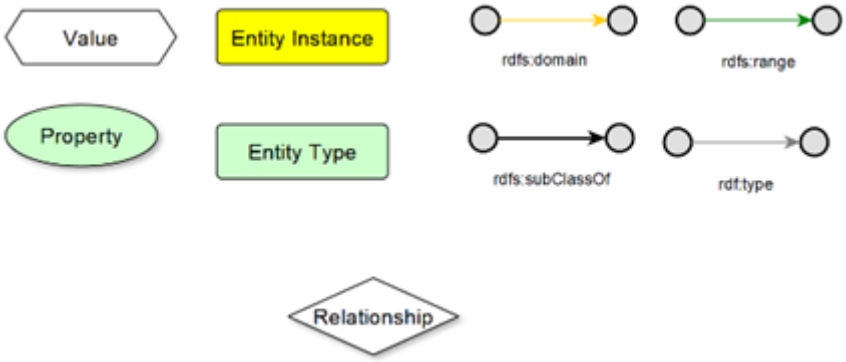


Figure 3: Main Symbols Definition

The Entity river (since it is a subclass of NGSI-LD Entity) is instantiated with the identifier *urn:ngsi-ld:River:La_Durance*. Several relationships are defined in this example: the first (*isAffluentOf*) describes the hierarchy between the rivers, to be used later on for graph-based data processing. The relationship *hasWeatherInformation* provides weather related information for the river.

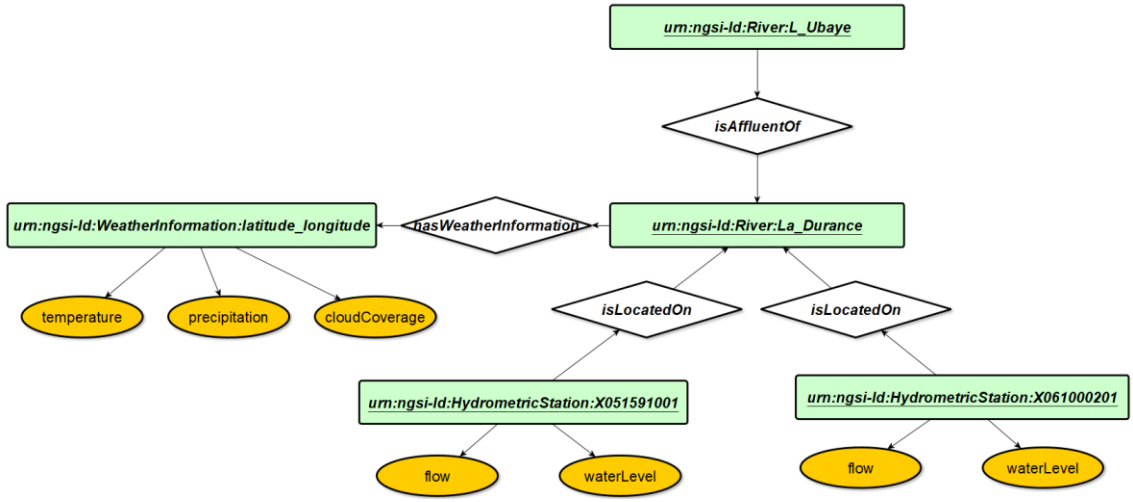


Figure 4: Illustration of NGDI-LD usage (extracted from water use case)

2.2.2 NGSI-LD Smart Data Models

The Smart Data Models (SDM) initiative, aims to offer a standardized approach to data representation across different domains. It aims to enhance interoperability between diverse systems and applications, thus enabling seamless communication. Although initiated by FIWARE, the Smart Data Models are open source and are developed through constant efforts from the community. The community contributes to creating or updating the existing data models.

Within SEDIMARK, implementing Smart Data Models for data assets aims to establish a homogeneous approach for participants to utilize the reusable common tools proposed by SEDIMARK, including AI modelling and data processing. Therefore, it complements the like NGSI-LD semantic-enabled APIs with NGSI-LD data models. The implementation of Smart Data Models ensures that providers reveal a consistent taxonomy and ontology. This enables SEDIMARK participants to both sell enhanced data and expand the pool of potential customers and data providers within the Marketplace for service providers.

Smart Data Models offer a customisable framework suitable for diverse domains, allowing for the creation of multiple domain-specific data models that cater to applications or datasets.

SEDIMARK advocates for the Practical use of Smart Data Models in data assets, despite the possibility of needing to adjust proposed models with new attributes and properties. Several data models have been identified from the domains supported by the initiative, including Smart Mobility, Smart Cities and Smart Energy. Additionally, Smart Data Models, such as the Data Quality model, can be used to enrich the content of existing datasets with the output of the data processing pipeline.

2.2.3 NGSI-LD API

2.2.3.1 Introduction

The NGSI-LD API supports several operations, with messages expressed in JSON-LD. The API is the standard for management of context information (which can be summarised as being any piece of information associated with a context such as time-location information). The Overall NGSI-LD API operations include:

Table 1: NGSI-LD operations

General Operations	Batch Operations
Entity create	Batch Entity Creation
Entity update	Batch Entity Create/Update (Upsert)
Entity partial update	Batch Entity Update
Entity delete	Batch Entity Delete
Entity retrieval	
Queries	**Temporal Operations**
Subscriptions	Create/Update Temporal Entity Representation
	Add Attributes to Temporal Entity Rep.
Registry Operations	Delete Attribute from Temporal Entity Rep.
CSRegistryEntry create	Modify Attribute Instance in Temporal Entity Rep.
CSRegistryEntry update	Delete Attribute Instance from Temporal Entity Rep.
CSRegistryEntry partial update	Delete Temporal Entity Representation
CSRegistryEntry delete	Retrieve Temporal Entity Evolution
CSRegistryEntry retrieval	Query Temporal Entity Evolution
CSRegistryEntry query	
CSRegistryEntry subscription	

This API relies on the NGSI-LD data model introduced earlier. In short, this model makes use of the JSON-LD serialisation format which adds linked data capabilities to the JSON format. The core of the model builds upon the concept of **Entity,** where an entity can have **Properties** and **Relationships** with other entities, building a property graph model.

The JSON-LD format allows to create a network of standards-based machine interpretable data across different sources. The JSON-LD format includes an @Context clause used to map short terms used in the serialization to URIs uniquely identifying concepts and mapping to specific types (e.g., Date Time).

In the following, we present the modelling process of the previous example using the NGSI-LD API based on JSON-LD messages for creating and querying instances of Sensor and Station.

2.2.3.2 Creating an instance of entity

An entity can be created using the following endpoint (among others):

POST {gatewayServer}/ngsi-ld/v1/entities

The payload must contain at least an id and a type for the entity. Any other attribute can also be added to the entity when creating it.

An example of payload used for the creation of a hydrometric station entity for the water use case is given below:

```
{
"id": "urn:ngsi-ld:HydrometricStation:X031001001”,
"type": "HydrometricStation”,
"location": {
            "type": "GeoProperty",
            "value": {
                "type": "Point",
                "coordinates": [
                6.2727640,
                44.4709131
            ]
        }
    }
}
```

Figure 5: example of NGSI-LD payload

2.2.3.3 Creating an instance of an attribute in an entity

An instance of an attribute can be added to an entity using the following endpoint (among others):

PATCH {gatewayServer}/ngsi-ld/v1/entities/urn:ngsi-ld:HydrometricStation:X031001001

The payload can contain an instance for any attribute (already existing or not), if an attribute does not exist, it will be created with the new instance.

An example of payload used to add some flow and water level measurements to a hydrometric station for the water use case is given below:

```
{
       "flow": {
           “value" : 138000.0,
           "observedAt" : "2023-12-04T10:15:00Z",
           "type" : "Property",
           "unitCode" : "G51"
       },
       "waterLevel": {
           "value" : 1237.0,
           "observedAt" : "2023-12-04T10:15:00Z",
           "type" : "Property",
           "unitCode" : "MMT"
       }
}
```

Figure 6: payload to add instance of attributes to an entity.

2.2.3.4 Retrieving an Entity by Id query

An entity can be retrieved using the following endpoint (among others):

GET {{gatewayServer}}/ngsi-ld/v1/entities/{{entity_id}}

An example of the response given for the entity used in the previous example is given below:

```
{
    "id": "urn:ngsi-ld:HydrometricStation:X031001001",
    "type": "HydrometricStation",
    "flow": {
        "type": "Property",
        "value": 139000.0,
        "observedAt": "2023-12-04T07:45:00Z",
        "unitCode": "G51"
    },
    "waterLevel": {
        "type": "Property",
        "value": 1238.0,
        "observedAt": "2023-12-04T07:45:00Z",
        "unitCode": "MMT"
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [
                6.49800996,
                44.55535641
            ]
        }
    }
}
```


Figure 7: response given for an entity request.

Figure 7 show the current state of the entity (i.e., only the last instances for each attribute are displayed).

The history of the entity can be retrieved using this endpoint (among others):

{{gatewayServer}}/ngsi-ld/v1/temporal/entities/{{entity_id}}?timerel=after&timeAt={{datetime}}&options=temporalValues

An example of the response given for the entity used in the previous example is given below:

```
{
    "id": "urn:ngsi-ld:HydrometricStation:X031001001",
    "type": "HydrometricStation",
    "flow": {
        "type": "Property",
        "values": [
            [
                80500.0,
                "2023-12-01T00:15:00Z"
            ],
            [
                82800.0,
                "2023-12-01T00:30:00Z"
            ],
            [
                85100.0,
                "2023-12-01T00:45:00Z"
            ], …
      }, …
}
```


Figure 8: getting history (timeseries) of an entity attribute.

2.3 Marketplace information models

The Marketplace Information Model is an RDFS/OWL-ontology covering the fundamental concepts of SEDIMARK needed for the registration of participants and the discovery and exchange of offerings and assets. This model establishes a common framework to ensure interoperability within a SEDIMARK-based Marketplace and includes the terms defined in Deliverable SEDIMARK_D2.2 [3] to enable participants to discover and exchange assets in the form of offerings. This common ontology is meant to serve as a shared language, fostering seamless communication and interoperability among the users of SEDIMARK. Therefore, the use of this information model is enforced for any participant or component that wants to join the Marketplace based on SEDIMARK guidelines. The main goal of this model is to ease the search and discovery of Participants and their offers, describing accurately their information.

The creation of this model is supported by existing proposals by similar initiatives and is built upon well-known ontologies such as Open Digital Rights Language (ODRL) [4], Data Catalog vocabulary (DCAT) [5], Friend Of A Friend (FOAF) [6] or the Dublin Core Terms (DCT) [7]. In particular, the model has its foundations in the proposal shared by the International Data Spaces Protocol [8], to try to be compatible as much compatible as possible with such an

initiative, although including new terms introduced by SEDIMARK (e.g., the concept of Offering; the additional type of assets that can be part of the marketplace; or the data quality information that is part of SEDIMARK).

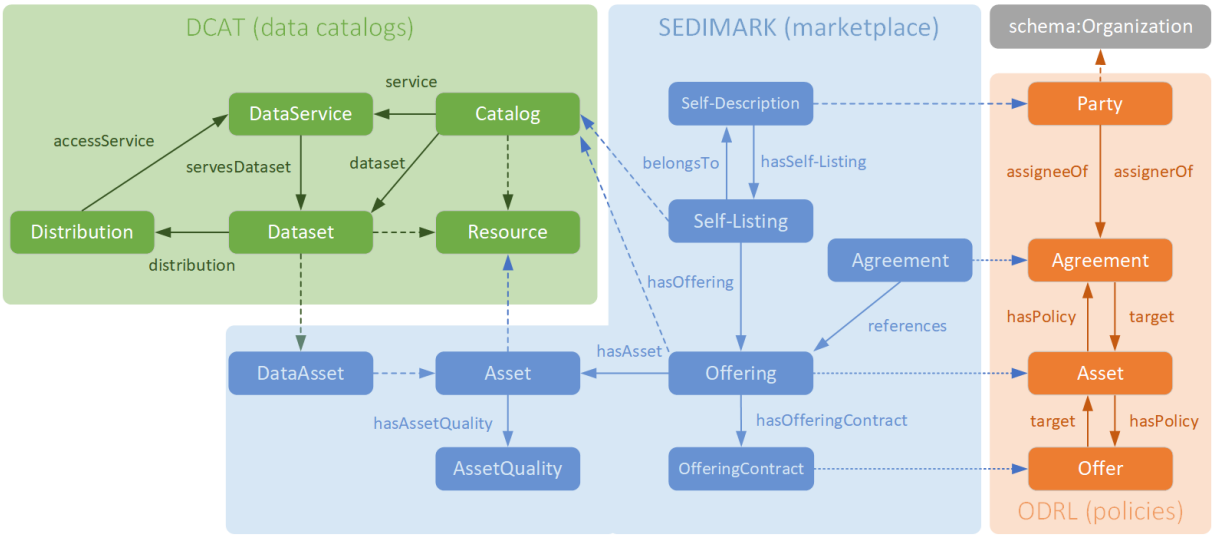


Figure 9: High-level view of the Marketplace Information Model

The current version of the Marketplace Information Model is depicted in Figure 9. The ontology is being built using Protégé version 5.6.3 [9] and will be updated in the forthcoming releases of the whole platform. There are three main concepts that can be highlighted in this information model: Self-Description, Offering and Asset, which are described below.

2.3.1 Self-Description

As defined in Deliverable SEDIMARK_D2.2 [3], Self-Description is a machine-interpretable document providing all the information about a Participant. In this case, it can be considered the main class within the Marketplace Information Model. This concept is also a core part of other information models, such as the ones from Gaia-X and IDS. Any Participant in a Marketplace must provide a Self-Description.

There are several concepts that are part of the Self-Description, including the information about the Participant (name, description, and the timestamps where this information was updated or created). Besides, the Self-Description can also link to a Self-Listing concept, which lists the set of Offerings from a Participant acting as a Provider.

```
{
    "@id": "https://connector.eu/",
    "@type": "sedimark:self-description",
    "dct:issued": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "dct:modified": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "foaf:name": "SEDIMARK Participant A",
    "dct:description": "Participant located in Europe...",
    "dct:language": {
        "@id":
"http://publications.europa.eu/resource/authority/language/ENG"
    },
    "sedimark:hasSelf-listing": {
        "@id": "https://connector.eu/self-listing"
    },
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-
model/0.1/"
    }
}
```


Figure 10: Self-Description JSON-LD example

2.3.2 Offering

Offering is a concept introduced by SEDIMARK and describes the set of assets that are part of an offer, along with their terms and conditions. This concept is conceived essentially as a subclass of a DCAT Catalog (as well as the Self-Listing concept) with additional properties to link to other SEDIMARK concepts such as Assets and Offering Contracts. As mentioned, only Participants acting as a Provider has a Self-Listing along with a set of Offerings.

The Offering concept is also a key difference between the SEDIMARK Marketplace Information Model and the IDS Protocol. In the IDS protocol, every offer is composed of a single Asset, while in SEDIMARK they can be grouped in an Offering, thus containing multiple assets per transaction.

Finally, one important aspect of Offerings is contracting. Each Offering contains a mandatory Contract object and, possibly, an Agreement. Both concepts, Contract and Agreement, are subclasses of ODRL Offer and Agreement concepts, respectively. While a single Contract object is mandatory (even if there are no particular restrictions) in each Offering, Agreement

objects are only required per transaction. Agreements are similar to contracts but add specific properties (i.e., assigner and assignee), which specify the Participants tied to the policies that are part of the Offering Agreement.

```
{
    "@type": "sedimark:self-listing",
    "@id": "https://connector.eu/self-listing",
    "sedimark:belongsTo": {
        "@id": "https://connector.eu/"
    },
    "sedimark:hasOffering": [
        {
            "@id": "https://connector.eu/offering/offeringID"
        }
    ],
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-
model/0.1/"
    }
}
```

Figure 11: Self-Listing JSON-LD example

```
{
    "@id": "https://connector.eu/offering/offeringID",
    "@type": "sedimark:offering",
    "sedimark:participantId": "https://connector.com/",
    "dct:title": "offeringName",
    "dct:description": "University from the North of Spain...",
    "dct:issued": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "dct:modified": {
        "@type": "xsd:dateTime",
        "@value": "2023-11-06T16:54:48.577964"
    },
    "dcat:keyword": [
        "keyword 1",
        "keyword 2"
    ],
    "odrl:hasPolicy": {
        "@id": "https://connector.eu/policy/policyID",
        "@type": "sedimark:Contract",
        "odrl:permission": [],
        "odrl:prohibition": [],
        "odrl:obligation": []
    },
    "sedimark:hasAsset": [
        {
            "@id": "https://connector.eu/asset/assetID"
        }
    ],
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-
model/0.1/"
    }
}
```


Figure 12: Offering JSON-LD example

2.3.3 Asset

Assets are the resources being offered in each of the Offerings. There have been three different asset concepts considered in SEDIMARK depending on the type of the resource they are representing. In this sense, the assets defined are datasets (either static or streaming data), AI Models, Services (e.g., IT infrastructure) and other assets, such as containers or

virtual machines. This is also another difference with the IDS Protocol proposal, as assets are only related to data, either streaming or static datasets.

Besides, a new concept has been defined within the Marketplace Information Model to represent the quality of the asset, thus giving an idea of the asset composing an Offering, to foster the exchange and represent what SEDIMARK tools through the Data Processing Pipeline can provide as an added value to providers which enhance their data through SEDIMARK.

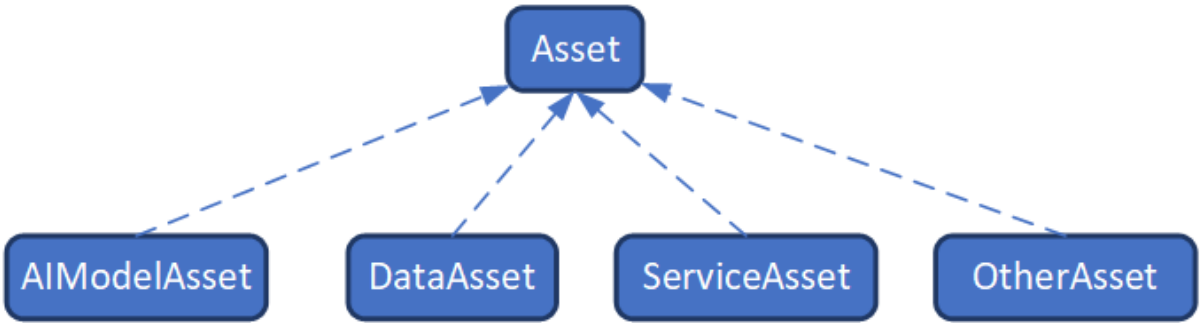


Figure 13: Types of Assets in the Marketplace Information Model

Data Assets

Datasets are represented in the Marketplace Information Model as a subclass of the dcat: dataset class, a well-known class from the DCAT ontology. DatasAsset includes object properties such as dcat: distribution and dcat:dataService to represent how these kind of asset can be accessed.

```
{
    "@id": "https://connector.eu/asset/assetID",
    "@type": "sedimark:Dataset",
    "dct:description": "data asset description",
    "dct:language": {
        "@id": "http://publications.europa.eu/resource/authority/language/ENG"
    },
    "sedimark:hasDataQuality": {
        "@type": "sedimark:dataQuality",
        "@id": "https://connector.eu/dataquality/dataQualityID"
    },
    "dcat:distribution": [{
        "@id": "https://connector.eu/dataquality/distributionID",
        "@type": "dcat:Distribution",
        "dct:format": {
            "@id": "HttpProxy"
        },
        "dct:issued": {
            "@type": "xsd:dateTime",
            "@value": "2023-11-06T16:54:48.577964"
        },
        "dct:modified": {
            "@type": "xsd:dateTime",
            "@value": "2023-11-06T16:54:48.577964"
        },
        "dcat:mediaType": {
            "@id": "https://www.iana.org/assignments/media-types/application/ld+json"
        },
        "dcat:accessService": {
            "@id": "https://connector.eu/serviceID",
            "@type": "dcat:DataService",
            "dcat:endpointDescription": "NGSI-LD API",
            "dcat:endpointURL": {
                "@id": "https://connector.eu/assetID/protocol"
            }
        }
    }],
    "@context": {
        "dct": "https://purl.org/dc/terms/",
        "dcat": "https://www.w3.org/ns/dcat/",
        "odrl": "http://www.w3.org/ns/odrl/2/",
        "dspace": "https://w3id.org/dspace/v0.8/",
        "sedimark": "https://sedimark.eu/marketplace-information-model/0.1/"
    }
}
```

Figure 14: DataAsset JSON-LD example

AI Model Assets

AI model assets are meant to represent exchangeable AI Models for both, distributed AI learning techniques and existing AI Models. Therefore, AI Model Asset reflects a number of aspects of AI models which will be exploited by search and discovery mechanisms to retrieve relevant offerings for Consumers. The following properties have been identified to become part of this class:

- ID (Model ID, as opposed to artifice ID. Model ID can be shared among multiple artifacts which have different serializations)
- Description
- Category
- Method
- Serialization (model serving format)
- Input (parameters)
- Output (parameters)
- Version
- Architecture
- Size
- Parameters
- Compute (resources)
- Execution (centralised, federated etc,)

ServiceAssets

ServiceAsset class is expected to cover service assets, such as the provision of data storage or computation resources. Each of these assets will represent the information describing the particularities of each service (e.g., number and type of processor cores, storage type, etc.).

OtherAssets

Other type of assets, such as Virtual Machines or Containers are included here, which will include additional properties to define their characteristics (e.g., computation requirements, operating system, etc.).

3 The Interoperability enabler

3.1 Data Annotations

In the realm of data processing and analytics, the utilization of smart data models within the NGSI-LD data format has emerged as an approach for data annotation and enrichment. This section explores the dual facets of data annotations: global annotations, applied at the dataset level, and local annotations, which focus on individual data points. Leveraging NGSI-LD's semantic capabilities and the richness of smart data models, this methodology ensures meaningful and interoperable annotations for improved comprehension and utilization of data.

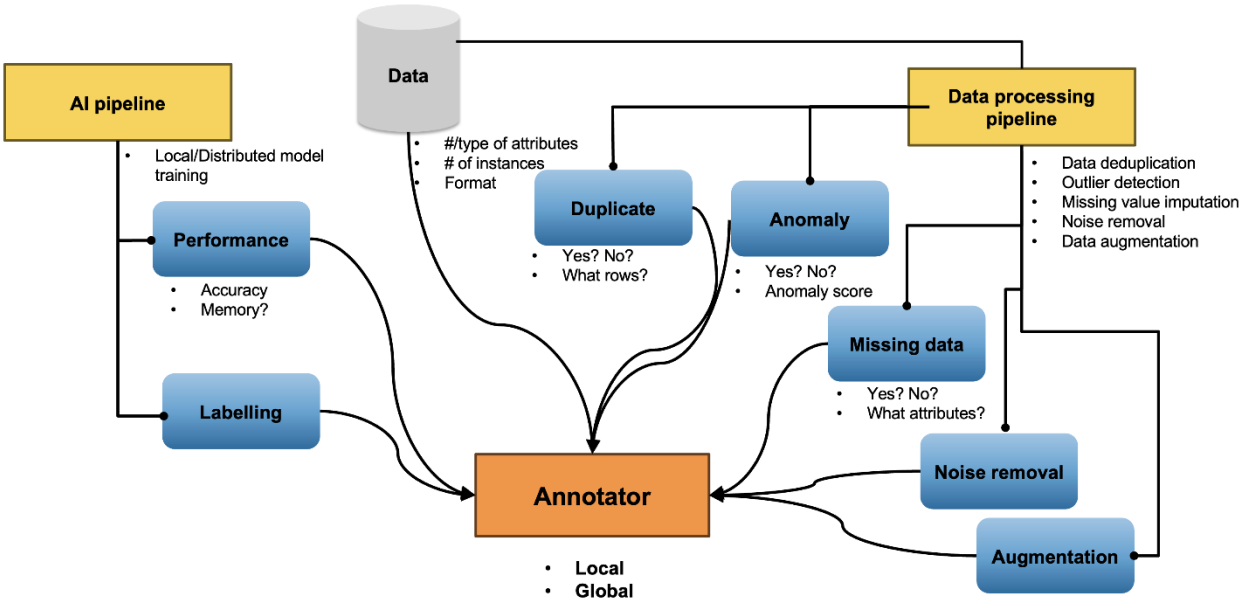


Figure 15: flow of local and global annotations

Annotations, whether at the global level, providing global information regarding datasets, or at the local level, enhancing individual data points with context-specific information, contribute to a more meaningful and interoperable SEDIMARK ecosystem.

3.1.1 Local annotations: enhancing individual data points metadata

Local annotations play a crucial role in enriching the metadata of individual data points with, *inter alia*, specific labels derived from data processing outcomes, and specifically incorporating data quality models. This includes categorizing data points based on predefined criteria, enabling users to identify patterns, missing values, or anomalies. For instance, in SEDIMARK, we will use the anomaly scores and other data quality measures to support local annotations by adding metadata to mark data points that deviate significantly from the expected patterns. These annotations are crucial for identifying potential errors, anomalies, or noteworthy events that may require special attention. This information will be obtained from the Data processing and AI pipelines.

Local annotations also consider temporal aspects, capturing changes in individual data points over time. This temporal context enhances the understanding of the dataset dynamics, supporting applications that require historical analysis or real-time monitoring. In addition to

standardized metadata, local annotations enable the inclusion of custom metadata tailored to specific SEDIMARK use cases. This flexibility allows users to embed domain-specific information, enhancing the richness of annotations for individual data points.

The integration of a data quality model to enrich the data within SEDIMARK refines this process by emphasizing the accuracy and completeness of individual data points and including information about outliers, missing data, and other anomalies. As shown in Figure 15, this metadata will be mainly generated from the Data deduplication, Outlier detection, and Missing value imputation components. For this to happen, the Smart data model "Data Quality" will be used to enrich the content of data points by matching the output of the data processing and AI pipelines to the properties of the Data Quality model. The existing specific properties for the different quality aspects that will be considered within SEDIMARK are provided in SEDIMARK_D3.1 [10]. For example:

- Accuracy
- Completeness (considering the missing values: isMissing, whatAttribute)
- Outlier (isOutlier, outlier score)
- Duplication (isDuplicate, whatInstance)

3.1.2 Global annotations: enhancing datasets/data streams metadata

Global annotations involve enriching the metadata associated with an entire dataset or data stream, providing a holistic view of the underlying information. This process is important for establishing a contextual foundation that facilitates a comprehensive understanding and utilization of the data as a whole. Smart data models with their domain-specific ontologies offer a structured semantic context for datasets, encapsulating the essential characteristics of the data.

Global annotations contribute contextual information to the dataset, offering insights into the overall purpose, source, and relevance that illuminate the overall data quality. Metrics such as completeness, accuracy, precision, and timeliness are essential components of global annotations, enabling users to assess the reliability of the dataset as a whole. This metadata enrichment facilitates efficient data discovery, sharing, and utilization in applications and analytics. Global annotations encompass general properties related to datasets or data streams and are presented in Sections 3.5 and 3.6 in the deliverable D3.1. For instance, we cite:

- Accuracy
- Precision
- Completeness
- Statistics extracted from data (data format, number of attributes, number of instances)
- Information regarding the dataset usage (e.g., with which ML task this data can be used, isLabeled)
- If data is curated (information on how outliers are identified and handled, how missing values are handled)

In this context of global metadata, DCAT is used within SEDIMARK as an integral element of the Offering description. Its role is to augment information about the Offerings, providing descriptions of datasets and any pertinent information required for enhanced data

discoverability. Consequently, global annotations will be integrated within the Offering description component.

3.2 Formatting

Data formatting defines the process of converting diverse data into a standard, widely recognised, and easily manageable data format like NGSI-LD. This element takes the data obtained in prior stages (e.g., Data Annotation) as its input, and presents NGSI-LD data as its output. Due to its diverse nature, the input can be presented in several data formatting standards like CSV, JSON, or XML. Moreover, the names of properties and their respective values can be highly variable. This is due to the presence of heterogeneous data sources, language policies, internal rules, units, and other contributing factors. The primary purpose of this component is to consolidate diverse data types into a singular format, namely NGSI-LD. Subsequently, this format can be further processed by other pipeline components.

We have identified several methods that achieve this objective. These methods grow in complexity as their flexibility and scalability increase.

- The first method is manual formatting, which involves a module that maps a specific type of input data directly to NGSI-LD. This module requires a full understanding of the input data since its mechanism is dependent on factors such as the input format or the specific words used in the data. The main disadvantage of this method is that it requires a separate formatter for each information source and type. As a result, the number of parallel modules required can quickly become unmanageable. However, its benefits include a more straightforward implementation, as well as high accuracy and minimal data loss due to the availability of a dedicated formatter for each input.
- The template-based formatting utilises a set of pre-designed templates that vary based on the data type and format used by the data sources. These templates extract and compile information from various sources to generate NGSI-LD compliant output. As with the manual formatting approach, the limitations include the prerequisite knowledge of the data source and the need for a distinct template for each output data type. However, this methodology offers multiple benefits, including improved reusability and more efficient implementation. The program which executes the formatting is generic, and templates are generated solely when a new source is handled. This eliminates the need for source code modification, making the process more user-friendly and automated compared to manual formatting.
- AI-based formatting employs Machine Learning methods to automate the formatting process. It consists of three stages: type identification, template selection, and transformation. During the type of identification stage, a previously trained AI model categorises the input data text into multiple classifications that match various output data models. Once identified, the corresponding template is selected from the pool of existing templates (which can be reused from the previous approach). In the third and final phase, the input data is automatically matched with the template, resulting in the NGSI-LD output data. This approach offers significant advantages in terms of scalability because once trained, the formatting process to NGSI-LD is almost fully automatic. The main disadvantage is that, while AI-trained models can handle the syntactic aspect of Data Formatting, we are uncertain whether this approach can achieve the semantic side. For instance, two data sources may include a particular property name (e.g., *dataset*), and while the first source is referring to a dataset name, the second may be using it to

represent a dataset identifier. This would result in different properties in the output data model (e.g., *name* and *id*) which the AI model may not be able to learn how to choose from. In addition, it is essential to ensure the appropriate training and retraining of AI models is performed when new types of input data are introduced into the pipeline.

Two options based on manual (based on the use of an ETL – Extract Transform Load – tool) and on template--based methods are described hereunder in the following subsections.

3.2.1 NiFi opensource ETL

The interface between a NGSI-LD broker and eternal data sources and data consumers can be achieved by the use of ETL (Extract Transform Load) tools which provide a bidirectional connectivity with several different protocols. The main used connectors in the context of data platforms targeting supervision are HTTP, MQTT, FTP and .CSV connectors. In the context of the water use case, the Apache NiFi open-source component has been used. This collection and validation module provides tools and an environment for:

- The creation and visual management of a directed graph of processes, generally by composing a set of processes that already exist in the catalogue (which contains nearly 400 connectors to data sources).
- A complete existing catalogue of processing allowing native integration of a REST API (with or without authentication), processing of CSV, Excel, JSON, or XML files, or even access to files stored on FTP servers.
- The versioning of the processing graphs with the traceability of the modifications made to each graph, as well as the author of each modification.
- The possibility of importing and exporting processing graphs.
- The possibility of automatically executing, at predefined regular intervals, processing flows.
- The development of coherent and loosely coupled components that can simply be reused by different processing tools.
- Management of errors during processing, with the possibility of configuring specific actions when they occur (subsequent test, sending a notification, etc.).
- The possibility, during processing, of querying a database for the detection of duplicates and of applying specific processing depending on the case.
- Automatic and detailed monitoring of all actions performed during processing.
- Securing exchanges and the possibility of defining detailed authorizations.
- The possibility of applying validations to the data processed, in particular via the definition of JSON or CSV schemas.
- Monitoring of all processing via a dashboard.
- Sending alerts triggered on the lack or invalidity of data.

Moreover, it is a tool designed natively for extensibility and therefore offers all the functionalities necessary for the development and integration of specific processing components.

Via the processing chain administration interface made available by NiFi (Figure 16), the customer also retains autonomy in their daily adaptation and in the addition of new file processing.

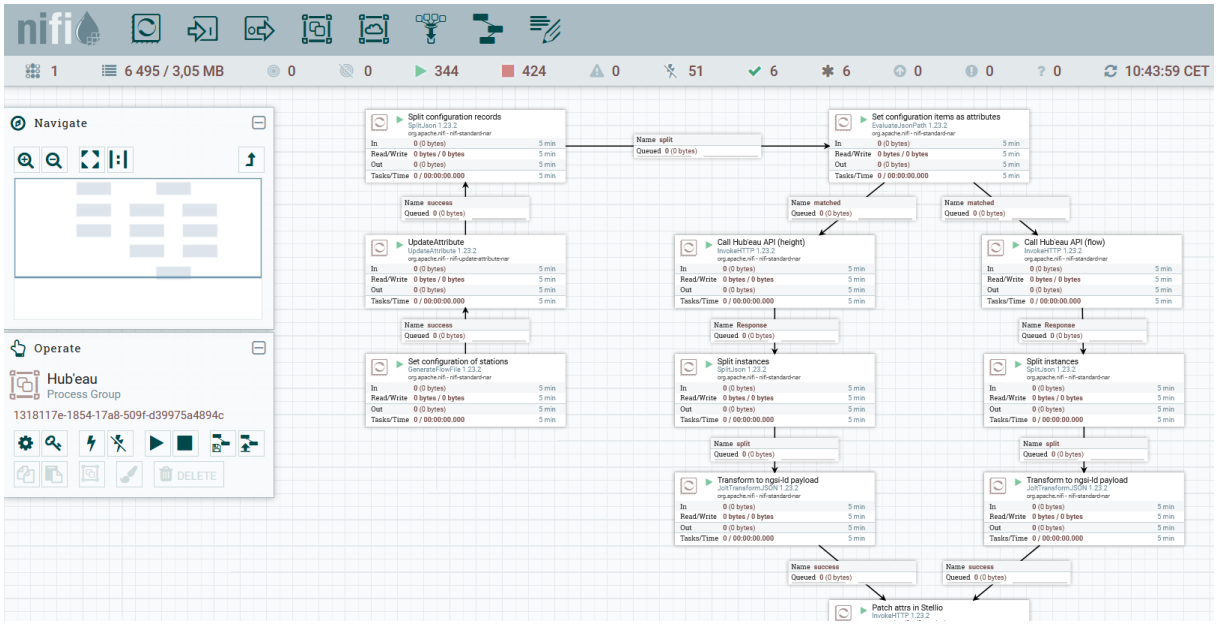


Figure 16: Example of the definition of a complete processing chain in Apache NiFi to collect data from Hubeau API (rivers flow and heights) (Water SEDIMARK use case)

3.2.2 Data Model Mapper

One of the options under consideration is the Data Model Mapper, which follows the template-based formatting approach. This module relies on a semi-automatic mechanism that only requires the Data Provider to fill in a configuration file (i.e., a template) at the time of providing a new Data Asset to the Marketplace. The result is a high-accuracy NGSI-LD output that is a lossless transformation of the input, provided that the template is complete. A fully functional first version of the module can be found in the SEDIMARK GitHub repository, along with an example of its usage. Figure 17 depicts an example of an input to this module on the left side and its output on the right side.

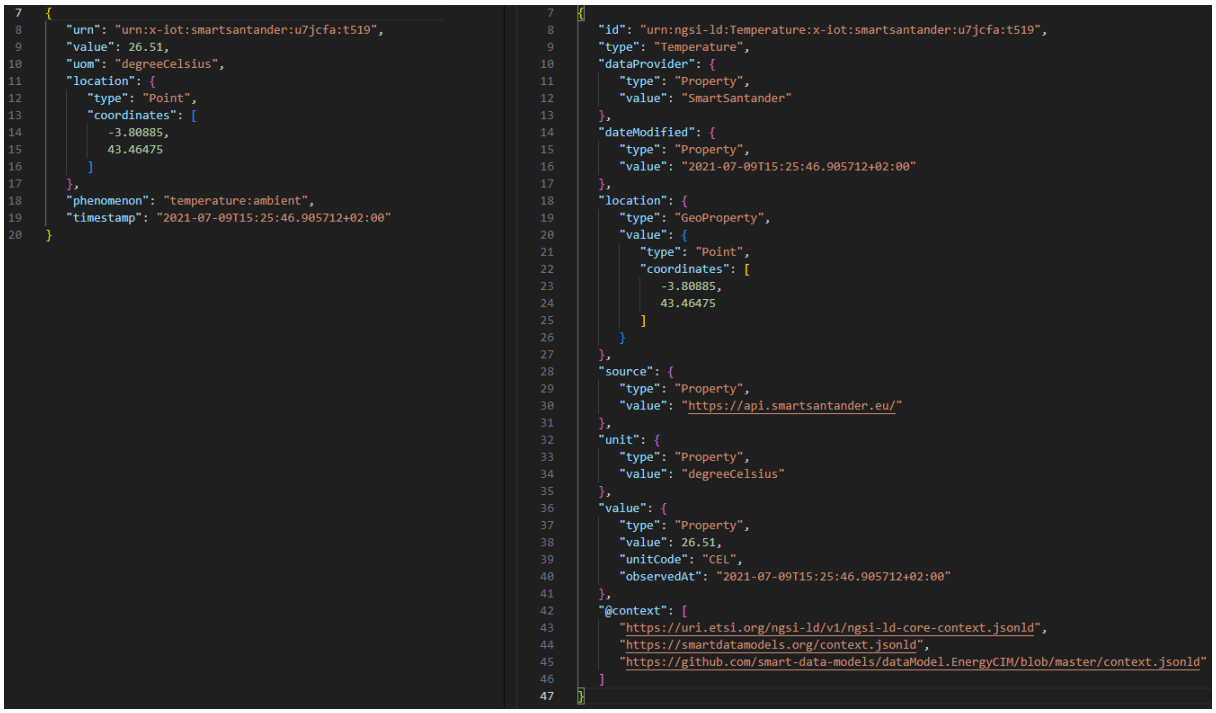


Figure 17: Data Model Mapper input and output example

3.3 Data validation / certification

Validation is required to ensure that the formatting applied to data assets and Marketplace self-descriptions is valid and complies with their respective information models (Table 2)

For both types of artefacts, validation is done through a set of stages.

- Format: the format that is used for representation complies with an acceptable serialization format and variant within that format.
- Syntax: the syntax applied to the annotation of the artefact complies with the classes and properties defined in the corresponding information model.
- Semantic: the axioms defined in relation to the relationships between instantiations of the concepts defined in the corresponding information model are compliant. This would include relationships regarding properties, class hierarchies, cardinalities etc.
- Domain-specific: the literal values that represent qualitative and quantifiable properties are valid in terms of ranges and states.

Table 2: Validation approach of platform assets

Artefact	Information model	Format Validation	Syntax Validation	Semantic Validation	Domain Validation
Self-Description	*Marketplace Information model*	JSON-LD/RDF schema validator	RDF model validator	Ontology compliance checker	Not applicable
Data Asset	*NGSI-LD, Smart Data Models*	JSON-LD schema validator	JSON-schema based validator, NGSI-LD model validator	SHACL validator (for graph-based validation)	Domain ontology + taxonomy validator

4 The AI enabler

4.1 Interoperable Federated Learning for SEDIMARK

4.1.1 Introduction

SEDIMARK is a complex data marketplace with several distinct pieces. Recent standardization efforts increase the need for interoperability and the importance of creating components that can be mixed and matched without affecting the entire offering pipeline. According to the Institute of Electrical and Electronics Engineers (IEEE) [11], interoperability is a characteristic of a product or system to work with other products or systems.

SEDIMARK distributed artificial intelligence pipeline includes several components, such as local model inference, local and distributed model training, and so on, englobed in what has been called the Artificial Intelligence Enabler. Due to the complexities of such systems, we need an approach for syntactical interoperability among those different components with a focus on model formats and service interoperability. SEDIMARK makes special emphasis on providing and using machine learning models and services throughout the entire marketplace and other marketplaces, hence, the interoperability requirement for model and artificial intelligence services is not only constrained to the Artificial Intelligence Enabler.

Syntactical interoperability at the model and service level can be achieved, to the extent of our knOWLedge, in two ways, namely, 1) providing support for the heterogeneity of formats that can come from other systems, and 2) providing common standard syntactical constructs and enforcing all components to support that standard. In SEDIMARK we propose to follow these approaches for model interoperability and service interoperability respectively.

SEDIMARK aims to provide interoperability capabilities to the SEDIMARK Federated Learning (FL) offering. Furthermore, we also provide tools for interoperability at the service level.

Regarding the first, we exploit the ONNX open standard for ML interoperability, and additionally provide support for handling Keras and PyTorch models, so that they can work in a variety of frameworks, and compilers. We follow the first approach to interoperability considering that third-party services can use a myriad of different formats to define machine learning models.

4.2 Local model training

The AI SEDIMARK pipeline will first allow the building and training of classical models. This is illustrated here with a model defined for energy consumption prediction.

In the electricity consumption prediction endeavour, we harness a week's worth of time-series energy consumption data, preceding our decision-making juncture, to forecast subsequent daily consumption in hourly intervals (Figure 18). Utilizing the advanced DeepAR model, we aim to construct a universal framework capable of accurately predicting consumption patterns across facilities and buildings of diverse magnitudes (Figure 19).

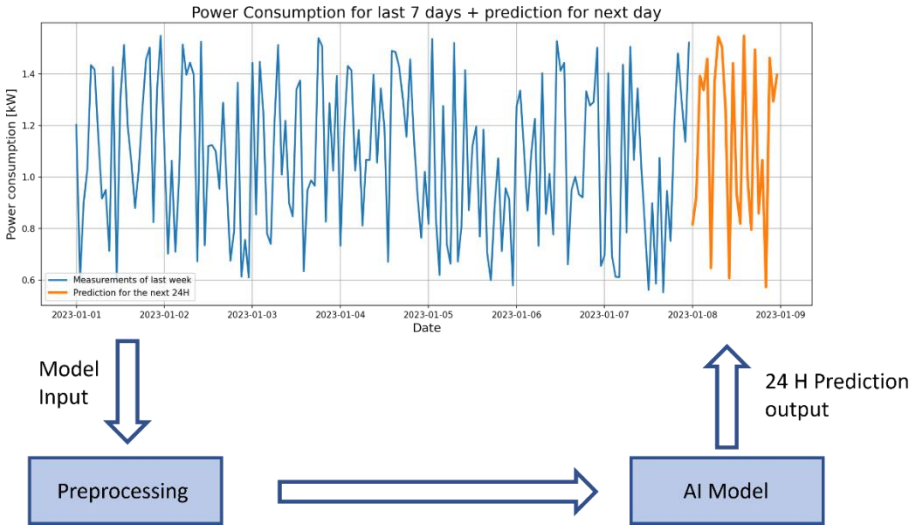


Figure 18: general principals of the locally trained predictive module

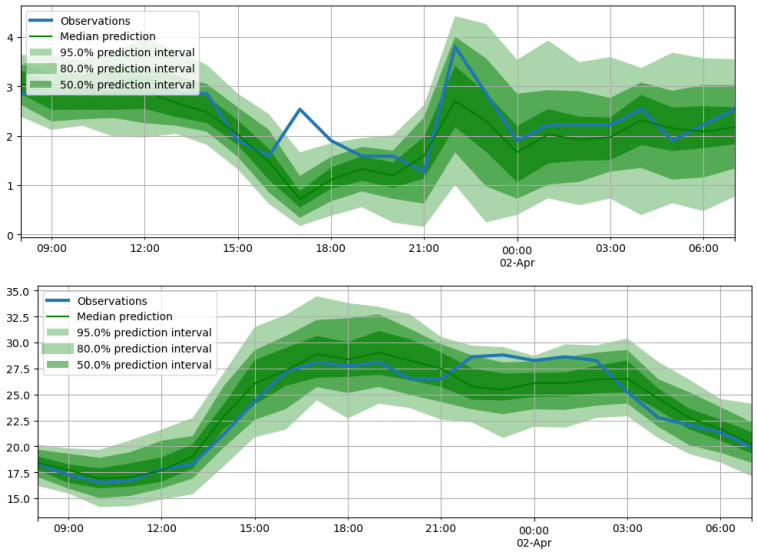


Figure 19: comparing deepAR based predictions with observations

Another example of a locally trained model relates to customer segmentation and churn prediction. In this initiative, we meticulously pre-process and sanitize datasets encompassing electricity consumption patterns, payment histories, geographical metrics, and behavioural indicators like complaints. Employing state-of-the-art ensemble decision tree algorithms such as LightGBM, Catboost, or XGBoost, our objective is to segment our customer base and forecast churn propensity, culminating in a calculated churn probability for each individual customer (Figure 20).

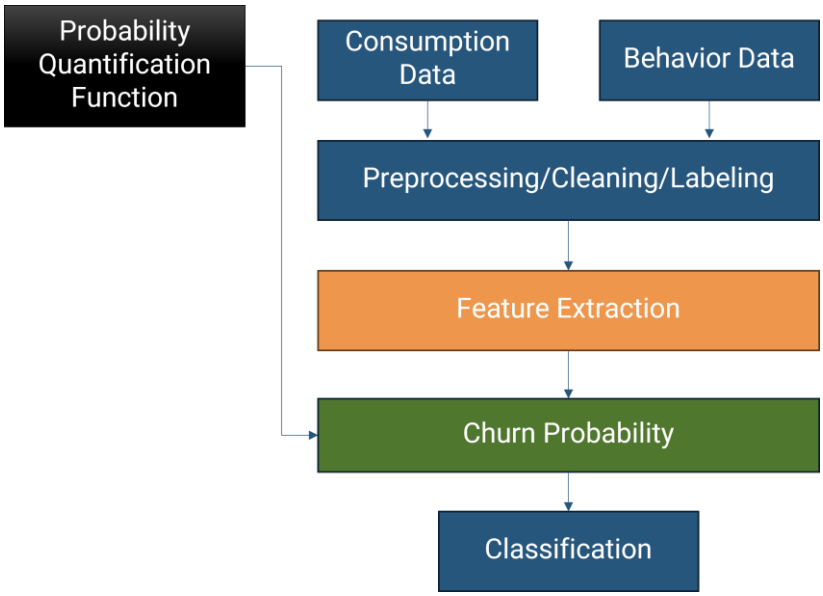


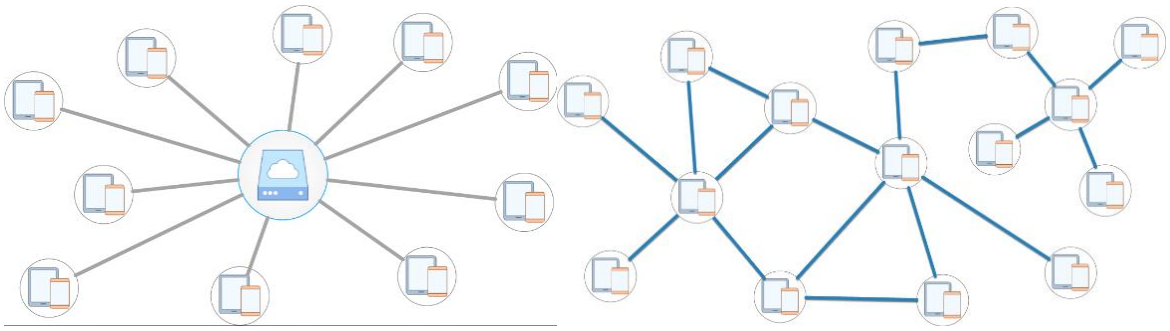
Figure 20: customer segmentation and churn prediction.

4.3 Distributed model training

Distributed model training within SEDIMARK can be divided into two main concepts:

1. **Federated learning (FL):** this concept employs a server and a set of worker nodes. The role of the server is to orchestrate the overall training process through model aggregation and model parameter redistribution approaches. The role of the worker nodes is to train a local model based on the local data and the updated parameters received from the server. Here the server has a complete view of the worker nodes, while the individual worker nodes are only aware of the server.
2. **Gossip learning:** this concept only contains worker nodes. Here the worker nodes are connected with a subset of other worker nodes, where they share the model parameters. Through gossiping of model parameters between worker nodes all worker nodes in the network will eventually agree on a global model. Differently from the previous setting, here workers are aware of a subset of other workers, but generally, no worker has a full view of the complete network of workers.

The differences between federated and gossip learning are illustrated in Figure 21. The clear advantage of the Gossip approach is that it can avoid the single point of failure by eliminating the server from the computation. However, this comes at the expense of the gossip approach taking longer to converge as it takes longer for the model updates to propagate through the communication network.



a) Federated Learning b) Gossip Learning

Figure 21: the difference between the architecture of a) federated and b) gossip learning.

Within SEDIMARK, two frameworks for distributed learning have been proposed and developed to cater for different scenarios and different user preferences.

1. Shamrock: this is a dynamic framework that is used for scenarios when data providers share through the marketplace either a model for training or a training process. This scenario is dynamic with participants being able to join or leave the training process at any given time.
2. Fleviden: is an extensible tool to define computational graphs representing the FL agents and the operations therein. We put special emphasis on tools that improve interoperability at the AI/ML models level, acknOWLedging that not all data providers/sources will use the exact same software to train/run the models from a federated learning point of view.

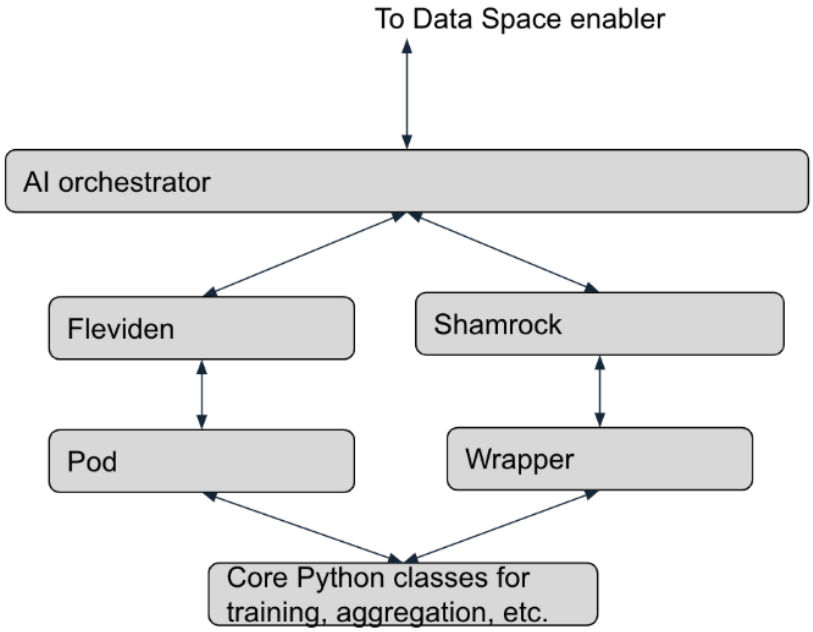


Figure 22: works for distributed learning developed within SEDIMARK.

As shown in Figure 22, the two frameworks are both designed to be modular and adaptive so that any project modules (i.e. models, aggregation mechanisms, privacy modules, etc.) can be developed in a framework agnostic way so that they can be used within both networks by exploiting their pods/wrappers.

When developing a FL solution, we need to consider the security requirements as a two-step process:

- The solution developer must develop the details of a federated algorithm by combining several techniques.
- The data provider must run a script representing the FL algorithm. This script must provide guarantees that it will follow the federated learning protocol strictly, e.g., keep the data and inference artifacts inside the provider infrastructure. This is what we call the minimum compliance requirement for any federated learning deployment.

There are several ways in which compliance can be accomplished. One alternative is that the data provider trusts the solution developer, which is not interesting from our perspective as in such a case we can fall back to more traditional machine learning solutions. Another approach is for the data provider to come up with their trusted review protocol to ensure the federated script provided by the solution developer is compliant. However, this is a difficult and expensive task that requires expertise and the capabilities to read through code and its dependency tree.

A proposal to be considered in future evolutions is the introduction of a third agent in the development process: the platform provider. The platform provider would create tools for the solution developer, such as automatic compliance checks, and demands the data provider to trust the platform provider but not the solution developer. This way, three-sided marketplace on top of federated learning:

- The data provider side, with private data and infrastructure offerings.
- The solution developer side, with novel algorithmic offerings and advanced implementations.
- The platform provider side, with their tooling offerings.

4.3.1 Shamrock

SEDIMARK will provide a flexible, fully decentralised model training framework. Titled “Shamrock”, this component offers a modular fully decentralised, asynchronous machine learning training solution. This component is based upon an extension of the popular open-source federated learning framework Flower [12]. Flower is a novel framework specifically developed for Federated learning research, allowing heterogeneous workloads at scale. Shamrock extends Flower, removing the need for always having a server coordinating the distributed learning process and enabling also completely serverless learning scenarios. Additionally, Shamrock is a dynamic framework does not require the set of training participants to be known ahead of time.

Shamrock moves beyond a client/server model, and instead makes *nodes* the first-class citizens within the distributed learning environment. While Shamrock primarily targets fully decentralised model training, an FL paradigm can be easily recovered by arranging the nodes within a star topology. As an extension of Flower, Shamrock inherits all the benefits of Flower, namely being scalable, allowing participation of heterogeneous clients running on different platforms, being framework agnostic (the group of clients can use TensorFlow, PyTorch, etc. according to their group decision), etc.

Within SEDIMARK, a node can be instantiated by any participant wishing to collaboratively train a machine learning model and can discover participants with similar compatible datasets. A training process, specifying a model architecture, a dataset specification, and a number of hyperparameters will then be advertised within the wider SEDIMARK trust infrastructure. New

participants with compatible datasets can discover this advertised training process, and then launch their own Shamrock nodes. Nodes follow a broader Gossip Learning (GL) protocol whereby they train locally their own version of the ML model and then send updates to other nodes that they select via a chosen sampling protocol. Shamrock is developed in a modular approach, such that tools developed within the SEDIMARK project in other tasks, i.e., model types, sampling strategies, quantization, etc. can be easily deployed within Shamrock, without the need of rebuilding those tools from scratch, but only with the use of simple wrappers.

4.3.1.1 Shamrock node implementation

In Flower, there are two types of nodes that participate in the federated learning training process: (i) a client and (ii) as server. The client is the node that does the local training process, running the ML model on top of the local data, periodically sending the model parameters to the server and receiving the updated parameters for the next round. The server is the node that at each round samples the clients to run the local training, receives the parameters from the clients, runs the process for aggregating the parameters and sends the aggregated parameters back to the clients for the next round.

In Shamrock, the two types of Flower nodes have been merged into a single Shamrock node, whose internal structure is depicted in Figure 23 below. Inspired by the Gossip Learning approach where all clients are of a similar type, Shamrock generalises the notion of a Flower node in a way that it can cover multiple distributed learning scenarios (as discussed below).

As depicted in Figure 23, a Shamrock node consists of four main threads of operation:

- Receive thread, which handles the reception of weights from the rest of the nodes participating in the learning process. As discussed above, Shamrock inherits from Flower the “communication-agnostic” feature, allowing multiple communication protocols between the nodes. However, currently, only gRPC (Remote Procedure Calls) is tested/supported, while in the future other protocols (i.e., REST) will be fully supported.
- Aggregation thread, which is the main thread that runs the sampling of the fellow nodes with which the node will communicate in the current round, and the runs rounds of aggregation of the parameter received from the fellow nodes.
- Training thread, which is the main thread that runs the local training process of the model based on the local data.
- Send thread, which takes the output weights from the training process and forwards them to the fellow nodes that participate in the current round.

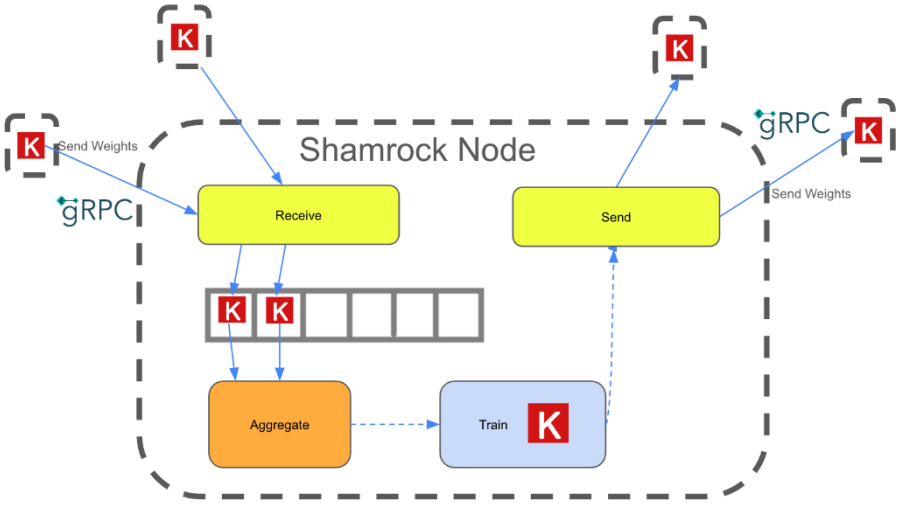


Figure 23: the internal structure of a Shamrock node.

In its current state of development, Shamrock is using Flower's gRPC as the communication protocol for sending and receiving model parameters defined within the Flower framework. The communication "receive" component hosts a gRPC server that continuously listens for incoming connections from other nodes, receives weights, and places them within a multiprocessing queue to be later aggregated. The use of a multiprocessing queue is critical to ensure that no weights are lost due to congestion, when receiving weights from many other nodes.

The main operational thread runs a local training process using Keras-Core for interoperability, such that the user can choose from either Pytorch, TensorFlow or Jax as their backend deep learning framework (more detail on model interoperability is given in Deliverable SEDIMARK_D4.3).

Execution alternates between rounds of aggregating any model updates that have been collected in the aggregation queue, running the local training procedure, and then launching connections to communicate model updates to other nodes, selected via a sampling strategy.

The dynamic nature of the Shamrock training process allows nodes to enter or leave the process at any given moment, without any special requirements, apart from following the SEDIMARK procedures for participating in the training as a service process.

4.3.1.2 Interaction of Shamrock framework with the rest of the SEDIMARK components

To implement the distributed learning process, Shamrock interacts with several other layers and components within the SEDIMARK architecture. These interactions are provided in the figures below.

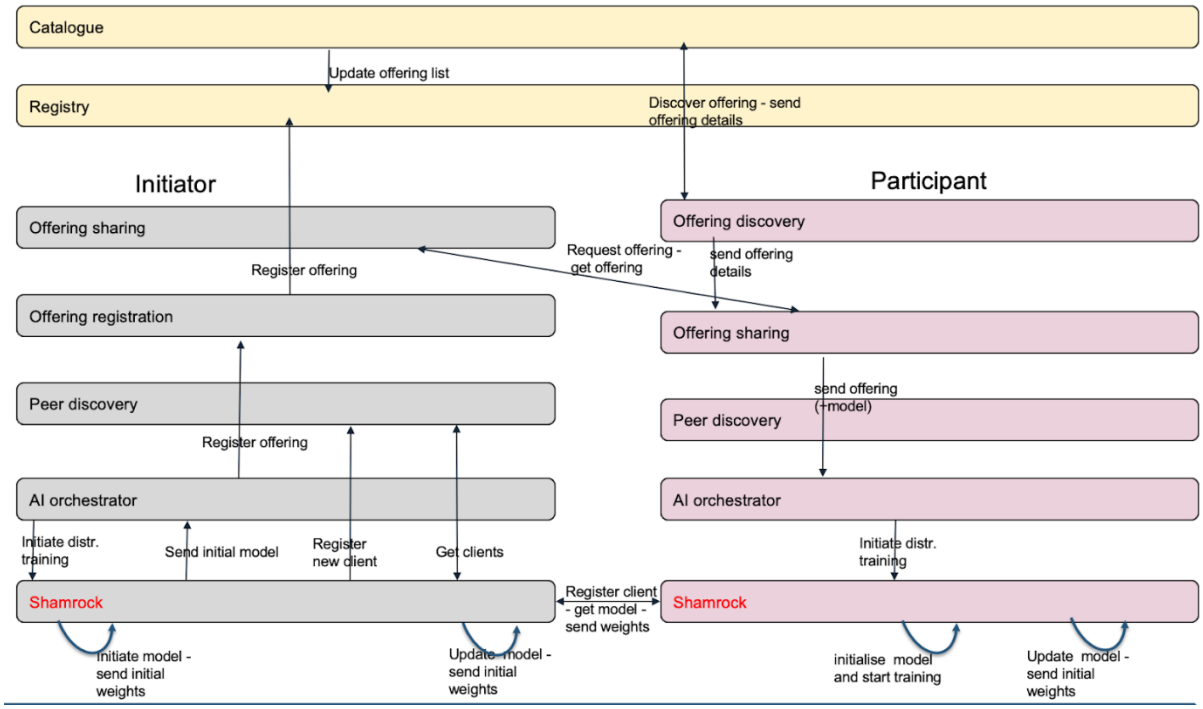


Figure 24: Shamrock-based Federated Learning process.

Figure 24 shows the interactions between Shamrock and the rest of the SEDIMARK components during the initiation and the execution of a Federated Learning process. It is clear that Shamrock basically interacts with the AI Orchestrator, which is the main component that handles the training process. In this scenario, the assumption is that an "Initiator", which is a SEDIMARK user (i.e., a provider) wants to start training a ML model on their data and then start a Federated Learning process, so that more participants join and help to train a better model. In this respect, the AI Orchestrator provides Shamrock with the user preferences and settings, i.e., the framework to use, the model to train, etc. Shamrock then initiates the training process, by initialising the model structure and its weights and forwards them to the Offering registration (through the AI Orchestrator) so that the training process is registered to the marketplace.

An Interesting participant in the process can discover the training process in the marketplace, finding the respective offering and receiving it from the "Initiator" via the Offering sharing component. The details of the distributed process are forwarded to Shamrock through the AI Orchestrator. The Shamrock module on the Participant contacts the respective module of the Initiator to register as a client and receive the latest version of the model weights. Then Shamrock initialises the local model and starts the local training process, updating the model and sharing the model updates to with the "Initiator".

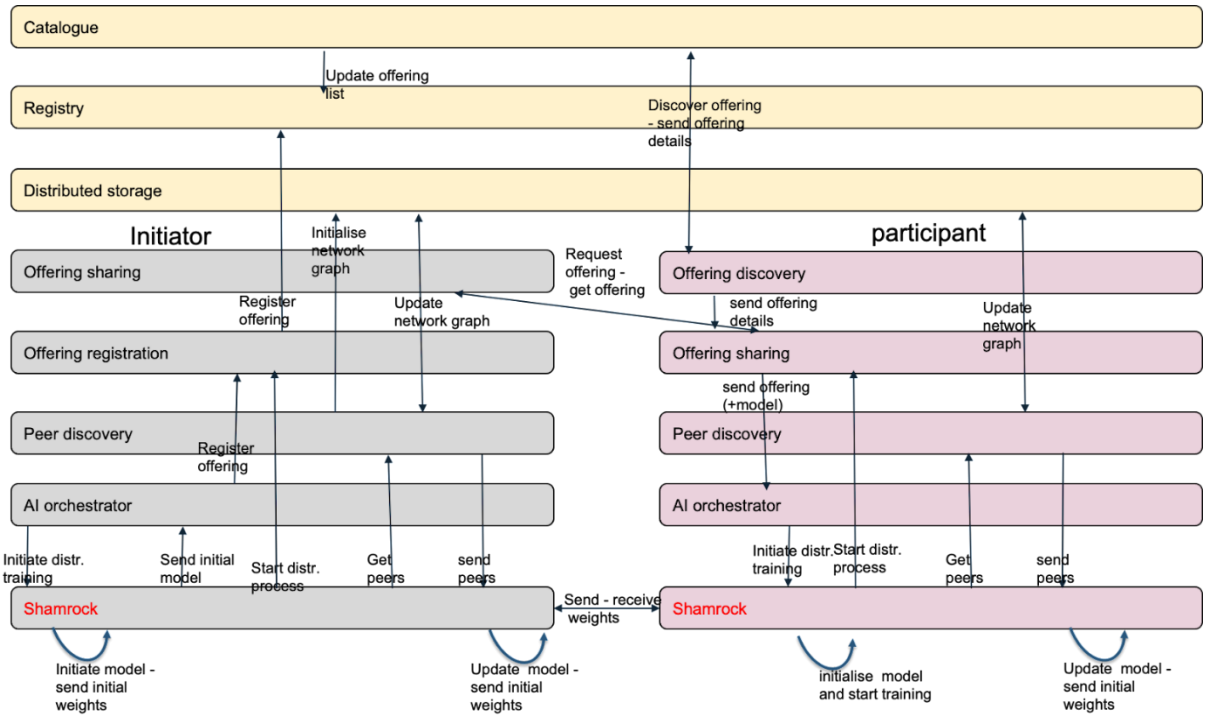


Figure 25: Shamrock-based Gossip Learning process.

Figure 25 shows the interactions between Shamrock and the rest of the SEDIMARK components during the initiation and the execution of a Gossip Learning process. The initialisation of the process is similar to that of Federated Learning (discussed above). The difference here is that there is not a server that holds a registry of the connected clients. To allow nodes to know the participants in the process, we exploit the distributed storage component of SEDIMARK, storing a "network graph", which is updated any time a node enters or leaves the training process. This is done by the "Peer discovery" component, which gets information from Shamrock regarding the training process id, etc. When a "participant" discovers and wants to join the training process, the Shamrock component initialises the received model and contacts the Peer discovery module to find out the fellow nodes participating in the current round. Then, Shamrock executes the next round, training the local model, updating the weights, and sending the weights to the fellow participants, while at the same time received the updates from its neighbours, performs the weight aggregation and continues to the next iteration of the training process.

In the current implementation, the communication between the Shamrock nodes takes place directly through the Shamrock component. In future versions, Shamrock will be extended to use the interaction and communication protocol of SEDIMARK.

In a future version of SEDIMARK, we will continue to iterate on the development of Shamrock. We intend to test its robustness when run across a greater number of machines, with larger datasets and larger models. We intend to further modularise Shamrock, to allow for the simple composition of the modules for aggregation, sampling and quantization that will be developed elsewhere within SEDIMARK. Initial evaluation results are given in SEDIMARK_D3.1, where the trade-offs between communication and performance are provided.

4.3.2 The Fleviden tool

The Fleviden library is an alternative and extensible tool to define computational graphs representing the FL agents and the operations therein. The main architectural pattern of Fleviden is pipes and filters. Federated learning is, at its very core, the distributed optimization of the parameters of a neural network.

The parameters being optimized are sent back and forward in different ways as messages between federated actors (clients, servers, etc.). This situation can be naturally modelled as pipes, as the connections among actors, and filters, as the actors themselves or, even more specifically, the operations they perform on the parameters being optimized, e.g., optimization, secure-sum, differential privacy, etc.

The core component of the Fleviden library is the Pod, with input and output wires that must be connected to define a federated computational graph. The minimal set of abstract operations for a Fleviden graph is:

- **create** new pods, i.e., nodes in the graph,
- **link** different pods,
- **wait** in a given input interface for a message not coming from a pod, e.g., from a web server.
- **bridge** through an output interface a message not sent to another pod, e.g., to a web server or the standard output.

The internal logic of a pod is to receive messages from input wires, process them and trigger the corresponding output wires. This logic and the pod itself can be implemented in various languages and using different technologies, as long as the wires interface logic and abstract operations are kept.

4.3.3 Handling of model interoperability in Fleviden

The Fleviden model interoperability strategy is to offer support for the most widely used frameworks for machine learning neural network models, that is, Keras with TensorFlow backend and Torch. In addition, we also provide support for ONNX models as a possible future standard for neural network exchange. Figure 26 shows a general overview of the Fleviden library's main components and packages. It shows the core Pod class which all the Fleviden pods inherit from. The arch package contains the available federated learning communication protocols, including a peer-to-peer communication approach that complies with the decentralized spirit of SEDIMARK. However, Fleviden flexibility allows us to also use centralized, hierarchical, asynchronous and swarm protocols. Special attention is deserved by the package *trainers*, which contains several pods that allow us to choose the framework with which to train the models, namely, Keras, Torch and ONNX.

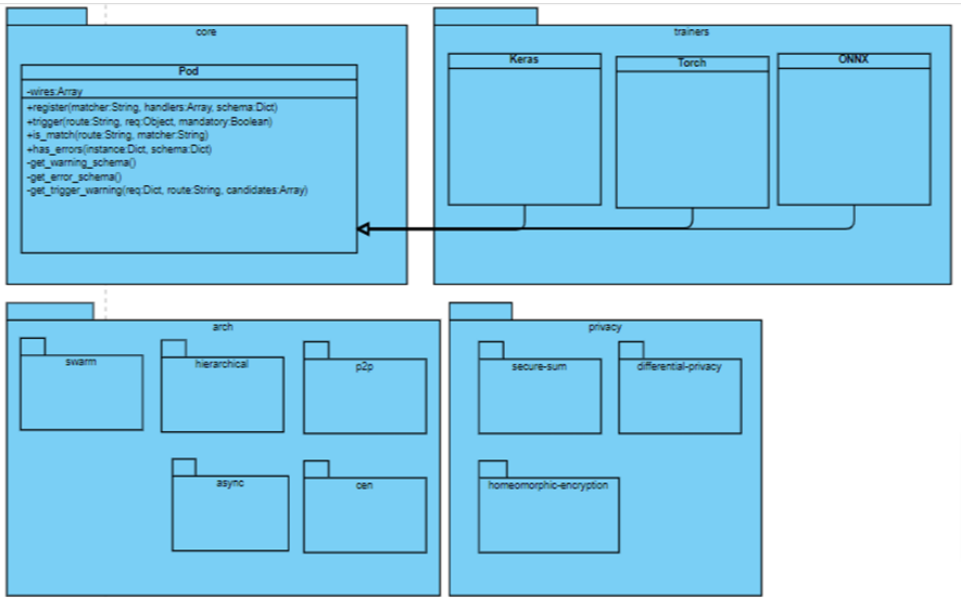


Figure 26: General overview of the Fleviden packages and classes.

The high-level functionalities defined at a trainer level are:

- Loading a model from a serialized file. One of the most important features of our trainer pods is the capacity to load a model specification from a file. In the case of a Keras model, we are making use of the Keras v5 model serialization format, in the case of Torch we are using TorchScript. The importance of loading serialized versions of the models instead of binary executables is to provide compliance guarantees and to prevent running arbitrary and potentially malicious code on the agent's infrastructure.
- Initializing a model that has been loaded at random.
- Given a parameter vector, performing a specified number of training iterations, and output the optimized / trained parameters to be shared with the other federated learning agents.
- Given a parameter vector and a dataset, evaluate the performance of the model.

4.3.4 Handling of service interoperability in Fleviden

For service interoperability we introduce a new scripting language and interpreter called Fleviscript, a simplifying tool to define Fleviden graphs by creating, linking, waiting and bridging pods. First, users do not need to get involved with the complexities of Python and Fleviden to develop federated algorithms. Secondly, it answers a business need: how can we run a script on a client site that is safe to execute without breaking compliance?

This script must guarantee that it will follow the federated learning protocol strictly, e.g., keeping the data and inference artifacts inside the client infrastructure. This is what we call the minimum compliance requirement for any federated learning deployment.

As Fleviscript (see annex) is not a general-purpose language, it can only do a limited number of operations, say create, link, wait and bridge pods. Though control statements can be added in the future, such statements are to support the previous actions required to build Fleviden graphs.

4.3.5 Asynchronous Federated Learning in Fleviden

Federated Learning is an increasingly broad topic and has been presented as a fundamental framework in which a set of agents can jointly train machine learning models without sharing their sensitive data. In SEDIMARK, we propose an asynchronous federated learning approach in which agents only need to perform a single communication round with the central server. This is useful in a variety of scenarios, but especially in the case of heterogeneous agents with different computing and connectivity capabilities. We conduct a thorough Bayesian performance analysis of the empirical results on a benchmark set of classification problems showing the strong performance of our approach when compared with several state-of-the-art alternative algorithms.

Motivating examples

We are particularly interested in situations in which the communication with the agents is 1) unpredictable or 2) sporadic up to the point where data transfers only happen once:

- Sensors network example: unpredictable communication. If we cannot predict when an agent goes offline (e.g., network communication failure or power irregularities, as in sensor networks), the computation performed in that agent is wasted without the possibility of reusing it. The reason why the computation cannot be leveraged is explained by synchronous nature of Federated Learning: after a certain time, the global model is updated with the currently available local models. Having to wait indefinitely for every model to be shared would prevent healthy clients from continuing their training. However, being able to reuse these partial computations is an appealing solution in this kind of situations because they represent valuable power and time.
- Video surveillance networks example: sporadic transfers. If an agent can only be reached once, we should be able to get its local model information in a single data transfer. The complexity of this situation in the general federated learning framework becomes evident again considering the synchronous federated learning rounds. Scenarios like this one are common in video surveillance systems, that operate with extremely sensitive data deployed in fully isolated networks that are not reachable from the outside world. Hence, getting data from these agents is a manual and carefully audited operation.

The bottom-line idea is that, either by sending small batches of information scattered over time (synchronous FL) or by relying on sharing large batches in a single round (asynchronous FL), eventually enough data would have become available to aggregate and to train the global model.

In our work, we present a novel federated learning algorithm that only needs to synchronize the different agents at the beginning of the optimization process, thus minimizing the number of federated rounds to only one. Our algorithm allows different agents to collect their private data and train their local models asynchronously, reducing the drawbacks of heterogeneity in computing capabilities and connectivity. As stated in the examples above, this is of particular importance for applications in which the data is behind an isolated network or behind a network of devices that are eventually available but not always.

The way we achieve asynchronous FL is by allowing each agent to fully train several local models before collecting them securely in a central server. Then, we aggregate a global model taking just one local model from each agent, provided that the distance between the models in the aggregated subset is minimum. As a downside effect, the amount of data synchronized in

the single communication round is considerably larger than the typical federated learning approach.

4.3.5.1 Asynchronous federated algorithm

In this section, we describe in depth our asynchronous federated learning algorithm. We give additional details on the way the global model aggregation is performed, and finally, we discuss some needed security mechanisms to ensure privacy preservation.

The way we achieve asynchronous FL is simply by fully training many local models using the agent's private data without any intermediate aggregation step. Algorithm 1 (Figure 27) describes in detail the steps of the proposed algorithm. It takes as input the number of federated agents *n* and the number of local models *m* trained by each of them. First, we initialize a global model from which every agent starts optimizing their *m* local models asynchronously. Each agent creates a list where the local models are stored after being trained in steps 6 to 8. The agent's local models are shared with the central server in step 11 in a secure way. Secure means that the central server does not have access to the agent's local models' parameters directly, but it can yet calculate the Euclidean distance between them (step 12) and then aggregate them (step 13).

Algorithm 1: Asynchronous Full Model Training Federated Learning.

input : Number of agents: n
input : Number of models per agent: m

1 Initialize the global model parameters θ_0 in the central server
2 Broadcast the global model parameters to each agent
3 **for** $i := 1$ **to** n **do**
4 Initialize local set of models $M_i := \{\}$
5 **for** $j := 1$ **to** m **do**
6 Initialize local model parameters θ using θ_0
7 Optimize local model θ for a given number of epochs
8 Store in the local list of models: $M_i := M_i \cup \{\theta\}$
9 **end**
10 **end**
11 Send all local model sets $M_1, \ldots, M_n$ to the central server
12 Find the set of nearest models from each agent:

$$(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n) = \underset{\{(\theta_1,\ldots,\theta_n):\theta_i \in M_i\}}{\arg\min} \sum_{i,j} \left\| \theta_i - \theta_j \right\|_2$$

13 Aggregate the set of nearest models to obtain the global model:

$$\theta_* = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\theta}_i$$

14 **return** Global model θ_*

Figure 27: steps for asynchronous federated learning algorithm

Private distance and model aggregation

The model's aggregation step of our algorithm (Algorithm 1, step 12) requires that we compute the Euclidean distance between model parameters in the central server. Considering that the model parameters are private to the agents, we need an approach to obtain such distance privately. To achieve this, we 1) transform the Euclidean distance into a scalar product between vectors, 2) estimate the scalar product from the size of the intersection of two sets and, 3) compute the intersection using a private set intersection cardinality protocol (PSI-CA).

Secure sum protocol

The secure sum protocol employed in this work is partially based on the work by [13]. The problem in hand is to obtain $\theta_* = \theta_1 + \cdots + \theta_n$ in the central server without sharing the values of the local model parameters θ_i. First, assume there is a total ordering between the agents such that each pair of agents (i, j), $i < j$ agrees on some random mask $s_{\{i,j\}}$. The observation made by [3] is that if agent *i* adds this mask to θ_i and agent *j* subtracts it from θ_j, then, the mask will be cancelled out and the actual model parameters will not be revealed. This way, each agent computes:

$$\hat{\theta}_i = \theta_i + \sum_{j=i+1}^{n} s_{i,j} - \sum_{j=1}^{i-1} s_{j,i} \tag{26}$$

and sends the result to the central server. The central server then computes:

$$\theta_* = \sum_{i=1}^{n} \hat{\theta}_i \tag{27}$$

$$= \sum_{i=1}^{n} \left(\theta_i + \sum_{j=i+1}^{n} s_{i,j} - \sum_{j=1}^{i-1} s_{j,i} \right) \tag{28}$$

$$= \sum_{i=1}^{n} \theta_i \tag{29}$$

without never having access to the actual values of the model parameters.

Sharing the common masks $s_{\{i,j\}}$ between agents naively would require a quadratic communication overhead. Instead, [3] proposes to share random seeds, by using Diffie-Hellman public keys, such that each agent can generate the random masks from these seeds using a pseudorandom number generator. Algorithm 2 summarises the steps required by the employed secure sum protocol (Figure 28).

The input of Algorithm 2 is the number of agents *n*, the number of models trained by each agent *m,* and the trained model's parameters $\theta_{\{i,j\}}$. Step 1 of the algorithm represents the processing that each node performs asynchronously. First, in steps 2 to 4, each agent *i* initializes a random seed that is shared with agents *j* > *i*. Afterwards, in step 6, each agent loops through all its local models. For each model, in steps 8 to 12, in node *i* we receive the random seeds shared by all agents *j* < *i*, generate the corresponding masks and subtract that mask from the local model parameters. Afterwards, in steps 13 to 16, in agent *i* we add the masks that all nodes *j* > *i* will subtract at a later stage.

Algorithm 2: Secure sum protocol for aggregating local models.

input : Number of agents: n
input : Number of models per agent: m
input : Trained models per agent: $\theta_{i,j}$ for $i \in [1,n]$ and $j \in [1,m]$

for $i := 1$ **to** n **do**
  for $j := i+1$ **to** n **do**
    Initialize random seed $x_{i,j}$
    Send random seed $x_{i,j}$ from agent i to agent j
  end
  for $k := 1$ **to** m **do**
    Initialize the $\hat{\theta}_{i,k} = \theta_{i,k}$
    for $j := 1$ **to** i **do**
      Receive random seed $x_{j,i}$ in agent i from agent j
      Generate the random mask $s_{j,i}$ using seed $x_{j,i}$
      Aggregate the mask $\hat{\theta}_{i,k} = \hat{\theta}_{i,k} - s_{j,i}$
    end
    for $j := i+1$ **to** n **do**
      Generate the random mask $s_{i,j}$ using seed $x_{i,j}$
      Aggregate the masks $\hat{\theta}_{i,k} = \hat{\theta}_{i,k} + s_{i,j}$
    end
    Send $\hat{\theta}_{i,k}$ to the central server
  end
end

Figure 28: Secure sum protocol

- Classic federated learning in multi-party computation scheme

In our pioneering endeavour, we intend to architect a framework dedicated to distributed learning leveraging classic federated learning methodologies. Central to our strategy is the deployment of a multi-party computation scheme, specifically designed to eschew the need for a central server, ensuring secure computation of the requisite updates for model parameters.

Federated learning in a multi-party computation (MPC) scheme offers a decentralized approach to model training. Each participating entity retains its data locally, ensuring data privacy and reducing centralization risks. Through MPC, parties collaboratively compute model updates without explicitly sharing their raw data. The aggregated model updates are then securely integrated, allowing for global model enhancement without compromising individual data integrity. This synergy of federated learning with MPC safeguards both privacy and efficiency in distributed learning scenarios.

- Meta-learning of ensemble model weights

Our aim is to design a sophisticated framework enabling the aggregation of an ensemble model from a diverse set of individual models present in a distributed network. Each node within this network retains its distinct model; however, through the application of meta-heuristic methods, these are harmoniously combined into a cohesive global ensemble model. A salient feature of this approach is the ability to integrate varying model architectures from different nodes, obviating the need for uniformity in their design, and thus enhancing the versatility and robustness of the resulting ensemble.

5 The DLT Infrastructure

This section describes the functionality of the DLT Infrastructure employed as the underlying foundation for the SEDIMARK Marketplace. The distributed ledger is able to provide trustworthy, non-repudiable and immutable information about Participants and Offerings. Moreover, this infrastructure is interconnected to other enablers and mechanisms allowing:

- identity management of the participants;
- offering metadata management for catalogue;
- storage of Trust Metadata;
- tokenization of assets;
- interactions with user's wallet;
- trading of assets among participants.

From an architectural point of view, the DLT Enabler is composed of two overlapped and interacting levels, namely:

- the Layer 1 (L1) which is the IOTA Tangle, and
- the Layer 2 (L2) which is the IOTA Smart Contract (ISC) chain.

These two layers are analysed in the SEDIMARK_D4.1. The following subsections analyse instead the implementation of the underlying infrastructure from the hardware and software point of view.

5.1 Software stack

The L1 layer provides the distributed ledger which is the underlying structure for storing transactions. The adoption of IOTA as DLT dictates the software to implement the network of nodes composing the Tangle. The HORNET software is a lightweight and powerful IOTA full-node software written in Go that allows to use IOTA DLT in an efficient way. This node software is the backbone of the distributed network. The node software allows to be flexibly configured. This aspect allows to tailor an instance of the node according to specific deployment requirements.

The architecture of a HORNET node supports an additional layer of flexibility with IOTA Node Extension (INX) interface. The functionality of a node can be extended with plugins based on INX. An example is the dashboard of the HORNET module which allows to graphically interact with the service and monitor the transactions received in real-time on a specific node.

IOTA Wasp software serves the smart contract functionality of Layer 2. Wasp is a software, written in Go, able to build, test, deploy and interact with smart contracts.

As specified before, the ISC VM is, in general, language-agnostic. Nevertheless, the most updated release of ISC currently supports Ethereum Virtual Machine (EVM)/Solidity smart contracts, as well as WebAssembly (Wasm) smart contracts. A Wasp node can deploy multiple chains of smart contracts. Every deployed ISC chain automatically includes a core contract (called *evm*). This core contract is responsible for running EVM code and storing the EVM state without additional software.

Given the necessity to anchor the state of a chain onto the Tangle (L1), an instance of Wasp requires at least a Tangle to connect to.

The Wasp node also provides a standard JSON-RPC service, which allows to interact with the EVM layer using existing wallet, like Metamask or tools (e.g., Remix or Hardhat). Deploying EVM contracts is as easy as pointing your tools to the JSON-RPC endpoint.

5.2 Current architecture instance

The direct installation of the software stack prepares a node able to interface and connect with the public network (i.e., the mainnet) of the IOTA Foundation. As a consequence, an instance of a HORNET node would be consistent with the content of the ledger public network, holding data and transactions not related to SEDIMARK Marketplace. Also, the hardware running the nodes software would be employed to become a peer of the decentralized public network.

For the scope of the SEDIMARK project, the architecture differs. In particular, the underlying structure needs to be reserved and adapted for the project target. Thus, the SEDIMARK Marketplace requires a private instance of the entire DLT, as well as the necessary smart contracts engine.

As previously described, the DLT Enabler is composed of multiple components. While maintaining the two levels necessary for the implementation of the other functionalities of the marketplace, the internal interconnections among the various services differ. These two layers of this enabler are mapped onto the physical hardware, as shown in Figure 29.

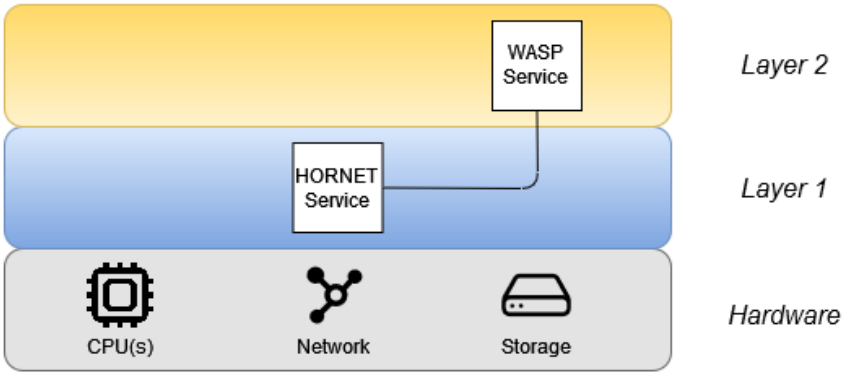


Figure 29: Layered architecture for DLT infrastructure

The initial decentralised network for L1 is composed of four instances of HORNET. The L1 services are deployed on different physical nodes. These nodes are interconnected to provide improved network resiliency. Each node contains a copy of the distributed ledger database.

Each instance of HORNET is complemented with a set of INX plugins. The following extensions are enabled: coordinator, indexer, spammer, dashboard, MQTT, faucet, participation and POI.

- The INX-coordinator performs the core functionalities of a node. It generates and issues the Milestones transactions, which are a special kind of transactions employed as markers of the progress and for providing timestamps for different points in the Tangle. Any transaction points, directly or indirectly, to at least one Milestone. The coordinator decides which transactions to approve. Moreover, it prevents double-spending issues and ensures that transactions cannot be reversed. The coordinator helps new nodes join the decentralized network by providing checkpoints for history, promoting faster synchronization. This ensures that new nodes have a starting point for validating the Tangle.

- The INX-spammer is a client application, running locally, which sends dummy transactions to the Tangle to provide a constant flow of transactions. This happens for performance reasons: a new transaction must be indeed referenced by at least three blocks. The spammer transactions increase the reference and confirmation rates of the DLT.
- INX-Indexer is an indexing tool to provide structured data that can be searched and utilised by wallets and other applications. The indexer maintains its own database separate from that of the node.
- INX-Participation is an extension for nodes to enable on-tangle voting. The extensions maintain its own database separate from that of the node and provides means to track events and votes.
- INX-MQTT provides an event-based real-time streaming node API. The built-in MQTT broker offers a list of topics clients can subscribe to, to receive the latest blocks and outputs attached to the tangle.
- INX-POI enables you to generate and verify Proof-of-Inclusion of blocks in the Tangle. Given a piece of data or transaction and the proof, it is possible to verify whether it was included in the Tangle at any given time.
- Finally, the faucet is employed for dispensing native tokens. For development purposes, two faucets are deployed respectively in L1 and L2.

For any instance of HORNET, a corresponding set of instances of Wasp are instantiated. These instances communicate directly only with the specific HORNET node. Multiple Wasp services are instantiated.

Three physical servers host the software stack for both L1 and L2. Each Server contains an instance of HORNET and one instance of Wasp. The three Wasp nodes are interconnected to each other for the purpose of validation of incoming *Requests*. In such a way, a consensus must be reached with an agreement of this whole set of validators. The current architecture is shown in Figure 30.

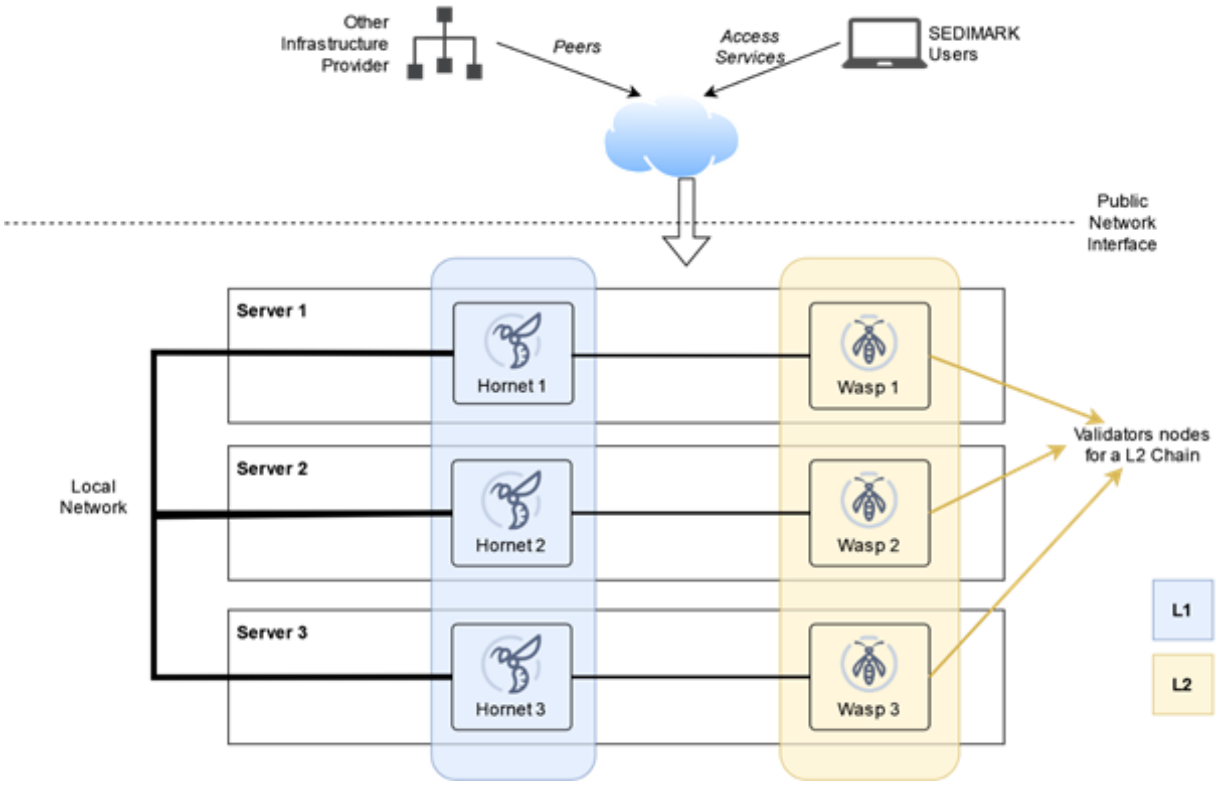


Figure 30: instantiation of DLT Layers onto physical hardware machines

The servers are interconnected to each other in a local network. These three machines are the peers composing the DLT and the Smart Contract chains for the SEDIMARK Marketplace. Incoming connections related to the digital identity are managed at L1 level, where the transactions store (partially) the elements of the SSI. Smart contract applications are deployed at L2 with the ISC allowing the trading of assets between SEDIMARK users and implementing the Marketplace business logic.

The infrastructure exposes a public interface that allows the interactions with remote users. SEDIMARK users are able to connect and interact with the services detailed resorting to the toolbox and the applications developed during the other WPs. The partners who want to enforce the capability of the SEDIMARK Marketplace can provide their own computational capabilities and storage facilities by deploying their own instances. The software stack is containerised for the ease of deployment on an external physical infrastructure. A newly deployed infrastructure can be linked to the existing one, thereby extending the capability of the whole system. In such a way, the partners' infrastructures become members of the ledger by acting as peers of the decentralized network and/or of network of validators.

6 The Storage enabler

In the digital age, data stands as one of most important components of any modern business ecosystem. Its value is especially magnified in the realm of distributed marketplaces, which serve as hubs of vast and diverse data exchanges across various regions.

These systems go beyond traditional storage paradigms by spreading data across multiple physical locations, be it within a single data centre location or across countries. Such an approach is not just a matter of scalability, but a pivotal strategy to ensure data availability, fault tolerance, and efficient distribution.

As these platforms deal with heterogeneous data – from city traffic information and user profiles to transaction records and user-generated content – the need for a robust, scalable, and interoperable storage mechanism becomes implicit.

Furthermore, as AI and machine learning continue to play a more significant role in data analysis and decision-making processes within these marketplaces, the integration between storage and computational resources gains even more prominence.

6.1 Significance of data storage

There are three pivotal attributes one must consider when choosing the data storage solution for these systems: scalability, fault tolerance and data interoperability.

- **Scalability** refers to the system's ability to handle increased load or demand by adding more resources or nodes, without affecting the system's performance or architecture. Distributed storage systems, unlike traditional systems, don't require massive fine-tuning or downtime to scale. As the need arises, new storage nodes can be incorporated seamlessly.
- **Fault tolerance** is the property that enables a system to continue operating seamlessly in the event of the failure of some of its components. Distributed storage systems typically replicate data across multiple nodes. This means if one node encounters a failure, the system can retrieve the data from another node. This redundancy always ensures data availability.
- **Data interoperability** is the ability of systems and services that create, store, and exchange data to have clear, shared expectations for the contents, context, and meaning of that data. In a distributed marketplace, data might originate from various sources - different vendors, platforms, or services. Distributed storage solutions can store diverse data types and structures, offering a unified access point irrespective of the data's origin. For marketplaces that involve multiple stakeholders, from vendors to third-party service providers, data interoperability ensures that all parties can access and understand the shared data, facilitating smoother collaborations and transactions.

6.2 Storage Enabling software

There are currently a few solutions that cover the above aspects, and we will analyse them in what follows.

6.2.1 MinIO

MinIO [14] is a high-performance, distributed object storage server, designed for large-scale data infrastructures. It is S3 compatible, built for the cloud-native world, supports object versioning, encryption, and event notifications. MinIO provides scalable storage for marketplace assets, supports multi-tenancy, and ensures high availability. MinIO is often used for storing unstructured data such as photos, videos, log files, backups, and container images.

There are two main operations that MinIO exposes: Bucket and Object operations. From these, we will depict only a few of them which are of most importance:

Bucket Operations:

- make_bucket - Create a bucket with region and object lock.
- list_buckets - List information of all accessible buckets.
- bucket_exists(bucket_name) - Check if a bucket exists.
- remove_bucket(bucket_name) - Remove an empty bucket.

Object Operations:

- get_object - Gets data from offset to length of an object. The returned response should be closed after use to release network resources.
- put_object - Uploads data from a stream to an object in a bucket.
- copy_object - Create an object by server-side copying data from another object. In this API maximum supported source object size is 5GiB.
- stat_object - Get object information and metadata of an object.
- select_object_content - Select content of an object by SQL expression.

MinIO supports bucket event notifications to a variety of targets such as AMQP, Elasticsearch, Redis, Kafka, Webhooks, and more. One can set up event triggers on object creation, deletion, etc.

MinIO supports S3-compatible server-side encryption with customer-provided keys (SSE-C).

MinIO provides SDKs for a wide range of languages, including Go, Java, JavaScript, Python, and .NET. This allows developers to easily integrate MinIO into their applications.

6.2.2 Triple Store

Triple stores [15] are a type of database optimized for storing and querying RDF data, which is often used in semantic web and linked data projects. It offers support for SPARQL query, reasoning capabilities, and efficient storage mechanisms. It also facilitates semantic searches, data linkage, and offers rich querying capabilities.

Apache Jena is an open-source Java programming framework for building applications using Semantic Web and Linked Data paradigms, through the provision of tools and libraries for storing models based on RDF and OWL graphs, and in turn provides a SPARQL query engine, ARQ2.

6.2.3 NGSI-LD brokers

NGSI-LD brokers implementing the temporal API described in section 2.2 also act as a storage. As an example, the Stellio context broker embeds a PostgreSQL database empowered with TimescaleDB and PostGIS extensions to handle time series as well as geographic information. Data exchange within the Stellio broker are made over a high-speed exchange bus built on Apache Kafka which allows to scale while a spring boot-based API gateway ensures the conformity to the NGSI-LD specification. Such an implementation provides interoperability while allowing fast ingestion rates. As visible in Figure 31, ingestion rates of more than 20k events/s have been demonstrated on machine with 8 vcore, 32 Go RAM and 4 To disks. Based on NGSI-LD specification, deployment architecture includes centralised, distributed, and federated options.

Figure 31: example of scaling capacity of Stellio context broker (number of inserted items per second over time)

6.2.4 Integration with AI Models

Distributed storage solutions are not just repositories for raw data; they play a vital role in AI ecosystems. With the ability to store vast amounts of data, they can be directly integrated with AI tools and platforms, enabling:

- Cleaning, transformation, and normalization of data before training.
- Using distributed data sources directly for training AI models.
- Storing evaluation metrics, logs, and results for AI models.
- Serving AI models directly from distributed storage solutions.

6.2.5 Challenges and Solutions

While distributed storage offers numerous benefits, it's not without challenges:

- Ensuring data remains consistent across nodes.
- Retrieving data from distributed nodes can introduce latency.
- Protecting data in a distributed environment.
- For each challenge, various strategies and tools can mitigate the risks and optimize performance.

7 Conclusions

This report has been drafted based on long discussions involving the different project members. It thus reflects a consensus reached among different organisations with different profiles. The encountered heterogeneity increased difficulty to reach a consensus but ultimately end up with a technology proposal armed for wider acceptance.

The different facets related to interoperability of the assets and services, have been described. While the internal of data pipelines models remain to be defined, it has been agreed that external interfaces would rely on the NGSI-LD interface and data model as defined by ETSI ISG CIM. With respect to marketplace offerings, ta RDFS/OWL ontology built by integrating well-known ontologies and models (ODRL, DCAT, FoaF, DCT) is proposed to allow for discovery and serving of the different marketplace assets.

Discussion for annotations of dataset (global) and datapoints (local) have been discussed with the aim of augmenting the data with quality related metrics. These annotations as well as related certification mechanisms, will be further refined when variety of situations will have clarified from the use cases implementation.

With respect to AI processing, emphasis has been placed on federated learning approaches with 2 frameworks proposed in the project (Fleviden and Shamrock). Both will be able to prove trained models under ONNX representation, to achieve interoperability at inference level. However, at training level, it is unclear yet if interoperability could be achieved across 2 different frameworks or if they would remain to be selected as 'black-boxes' by the marketplace users.

Finally, storage options considered for the different components are presented, including a DLT infrastructure used to provide trustworthy, non-repudiable and immutable information about Participants and Offerings.

8 References

[1] E. I. CIM, “ETSI GS CIM006 - 3NGSI-LD Information model",” ETSI, 2023.

[2] W. Li, G. Tropea, A. Abid, A. Detti and F. Le Gall, “Review of Standard Ontologies for the Web of Things,” in *Global Internet of Things Summit (GIoTS)*, Aarhus, 2019.

[3] SEDIMARK, “D2.2 - SEDIMARK Architecture and Interfaces. First version.”.

[4] R. Iannella and S. Villata, “ODRL Information Model 2.2,” 15 February 2018. [Online]. Available: https://www.w3.org/TR/odrl-model/.

[5] R. Albertoni, D. Browning, S. Cox, A. González Beltrán, A. Perego and P. Winstanley, “https://www.w3.org/TR/vocab-dcat-2/,” 4 Ferbruary 2020. [Online]. Available: https://www.w3.org/TR/vocab-dcat-2/.

[6] D. Brickley and L. Miller, “Friend Of A Friend - version 0.99,” 14 June 2014. [Online]. Available: http://xmlns.com/foaf/0.1/.

[7] DCMI Usage Board, “DCMI Metadata Terms,” 20 January 2020. [Online]. Available: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.

[8] Internatonal Data Spaces Association, “International Data Space Protocol - Working Draft,” 2023. [Online]. Available: https://docs.internationaldataspaces.org/ids-knowledgebase/v/dataspace-protocol/.

[9] Stanford University, “Protégé,” [Online]. Available: https://protege.stanford.edu/.

[10] S. p. (. 101070074), “D3.1 Energy efficient AI-based toolset for improving data quality. First version,” 2023.

[11] Institute of Electrical and Electronics Engineers, “IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries,” New York, NY, 1990.

[12] D. J. B. a. T. T. a. A. M. a. X. Q. a. J. F.-M. a. Y. G. a. L. S. a. K. H. L. a. T. P. a. P. P. B. d. G. a. N. D. Lane, *Flower: A friendly federated learning research framework.",* arXiv 2007.14390, 2022, p. v.

[13] V. I. B. K. A. M. H. B. M. S. P. D. R. A. S. a. K. S. K. Bonawitz, “Practical secure aggregation for privacy-preserving machine learning,” in *ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '17*, New York, NY, 2017.

[14] N. A. M. P. M. Ian F. Adams, “Enabling near-data processing in distributed object storage systems,” vol. Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems, 2021.

[15] Y. Z. P. Y. H. J. L. L. Buwen Wu, “SemStore: A Semantic-Preserving Distributed RDF Triple Store,” vol. CIKM '14: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge, 2014.

[16] n. (. A. F. Lead author (surname, “Project name, deliverable number and title (i.e. WITDOM. D2.2 - Functional analysis and use cases identification),” Year (i.e. 2015).

[17] J. D. S. G. W. C. H. D. A. W. M. B. T. C. A. F. a. R. E. G. Fay Chang, “Bigtable: A Distributed Storage System for Structured Data,” Vols. ACM Trans. Comput. Syst. 26, 2, Article 4 , 2008.

[18] SEDIMARK, “D2.1 - Use Cases Definition and Initial Requirement Analysis”.

[19] W. W. C. a. S. E. F. P. Ravikumar, “A secure protocol for computing string distance metrics,,” in *Fourth IEEE International Conference on Data Mining* , Brigthon, 2004.

[20] M. F. D. G. S. J. M. S. a. J. T. R. Egert, “Privately computing set-union and set-intersection cardinality via bloom lters,” in *Information Security and Privacy*, E. F. a. D. Stebila, Ed., Springer International Publishing, 2015, pp. 413-430.

Annexes

Fleviscript details

A Fleviscript program can be specified in plain text with the following structure:

- Import pods: Any Fleviscript program starts with zero or more import statements and is the only part in which they are allowed in a program. An import statement is used to indicate what pods we want to use and create during the script execution. Its structure is given by the keyword import followed by the Fleviden package structure in which the pod is located, the reserved keyword as and the pod name itself. For example, to import the pod Server in package fleviden/arch/cen/server.py we would use the following import:

  import fleviden->arch->cen->server as Server

- Registering wires: Any Fleviscript program runs encapsulated in a Fleviden pod. In other words, a Fleviscript program is a pod whose logic is defined by the computational graph of the pods created within. This conforms a clear modular system in which by default everything that is created inside a Fleviscript is private. Anything intended to be public has to be connected through an input or output wire.

  Input and output wires corresponding to the Fleviscript enclosing pod must be declared after the import statements and before any other statement. This is the only place where it is allowed to use registering statements.

  There are two types of wire registering instructions: the input statement and the output statement. To declare an input wire, we use the keyword input followed by an identifier representing the name of the wire. Output wires are registered in the same way but using the keyword output. This is an example of a registering wire block:

  input /update
  output /updated

- Variables declaration and literals: The Fleviscript language allows us to define variables and to perform several manipulations on top of them. The variable declaration and assignation ops are done with the same instruction using the keyword |< followed by an identifier that represents the variable name and followed by the operator = and then the value to be set to that variable. For example:

  |< continent = Europe

  In addition to simple variables like the one above, we can define variables that contain lists and dictionaries whose syntactical constructs are similar to those we find in Python, for example:

  |< person = {name: john, lastname: doe}
  |< continents = [africa, europe, asia, america, australia]

A special feature of Fleviscript is that everything is an identifier, even variable names and values. That means that using a literal as a variable name will disable using that identifier in other parts of the program as a value. For instance:

|< name = john
|< person = {name: john, lastname: doe}

will result in person variable value being {john: john, lastname: doe} as the name identifier used in the dictionary key was evaluated to its value. A recommended best practice to avoid this kind of confusion is to prefix every variable declaration with an uncommon symbol, for example:

|< $name = john
|< $person = {name: $name, lastname: doe}

The only place in which this identifier evaluation process does not happen is in the variable declaration instruction itself, that is:

|< $name = john
|< $name = maria

will result in overwriting the variable $name value instead of creating a variable named john.

- Pod declaration: Pods can be declared by using the >| instruction followed by the pod identifier, the = symbol, the pod type previously imported and finally a dictionary containing the arguments required to instantiate the pod. For example:

|> server = Server {
num_rounds: 4,
clients: [{id: client-one}, {id: client-two}] }

- Wait instruction: A wait sentence is created by using the << instruction followed by an identifier representing the pod name, the symbol -> and an identifier representing the wire name. Optionally, a dictionary can be provided in case additional arguments are needed while waiting. For example:

<< $http -> rest.updates

- Bridge instruction: A bridge sentence is created by using the >> instruction followed by an identifier representing the pod name, the symbol -> and an identifier representing the wire name. Optionally, a dictionary can be provided in case additional arguments are needed while bridging. For example:

>> $http -> broadcasted {host: localhost, port: 9090}

- Link instruction: A link sentence is created by using the -> instruction followed by an identifier representing the pod name, the symbol -> and an identifier representing the wire name. Optionally, a dictionary can be provided in case additional arguments are needed while linking. It follows an identifier representing the target wire name the symbol -> and an identifier representing a target pod name. For example:

-> $http -> rest.update update -> $server
-> $server -> updated aggregate -> $aggregator