2024

UPCAST PROJECT

Draft Document

DELIVERABLE 1.3

THIS DOCUMENT IS IN DRAFT FORM AND PENDING OFFICIAL APPROVAL. IT IS SUBJECT TO REVIEW AND MAY BE UPDATED.

D1.3: Updated Project Concept and Architecture

This project has received funding from the European Union’s Horizon Research and Innovation Actions under Grant Agreement № 101093216.

Title:	Document version:
Updated Project Concept and Architecture	0.50

Project number:	Project Acronym	Project Tittle
101093216	UPCAST	Universal Platform Components for Safe Fair Interoperable Data Exchange, Monetisation and Trading

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M18 (June 2024)	M19 (July 2024)	Other-PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other; ORDP: Open Research Data Pilot; E: Ethics.

**Security Class: PU: Public; PP: Restricted to other program participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Lead Authors (organization):
Sofoklis Efremidis, Dimitrios Verlekis, Kostas Kalamboukas (MAGGIOLI)
George Lioudakis, Mariza Koukovini, Eugenia Papagiannakopoulou (ABOVO)
Shanshan Jiang, Gøran Svaland (SINTEF)
Soulmaz Gheisari, George Konstantinidis, Luis Daniel Ibanez, Semih Yumusak, Jaime Osvaldo Salas (SOTON)
Aditya Grover, Hanene Jemoui (CEDAR)
Paraskevi Tarani (MDAT)
Charalampos Bratsas, Lazaros Ioannidis (OKFN)
Fernando Perales (JOT)
Majid Ektesabi (NOKIA)
Evangelos Kotsifakos (LSTECH)
Nenad Stojanovich, Milan Vuckovic (Nissatech)
Olga Papadodima (NHRF)
Anestis Stamatis (CACTUS)
Jeremy Decis (Dawex)

Abstract:

Deliverable D1.3 reports on the updated UPCAST concept and MVP. An update of a typical workflow is given along with an updated version of the architecture Moreover, the document provides updates of the designs of the plugins and an overview and technical details of the two available marketplaces. Finally, updates to the project' pilots are given.

Keywords:

Software architecture, platform architecture, plugin integration, APIs, requirements, process workflow, Legal Framework.

REVISION HISTORY

Revision:	Date:	Description:	Author (Organization)
v0.10	8/01/2024	ToC	Sofoklis Efremidis (Maggioli)
v0.11	22/01/2024	ToC update	Sofoklis Efremidis (Maggioli)
v0.12	08/04/2024	ToC update	Sofoklis Efremidis (Maggioli)
v0.13	13/05/2024	Update of Architecture	Sofoklis Efremidis (Maggioli)
v0.20	9/7/2024	Incorporation of contributions	Sofoklis Efremidis (Maggioli)
v0.25	22/7/2024	Incorporation of contributions	Sofoklis Efremidis (Maggioli)
v0.50	30/7/2024	Update after internal review	Sofoklis Efremidis (Maggioli)

This project has received funding from the European Union's Horizon Research and Innovation Actions under Grant Agreement Nº 101093216.

More information available at https://upcastproject.eu/

COPYRIGHT STATEMENT

Table of Contents

1 Introduction ... 8
1.1 Purpose of the Document ... 9
1.2 Scope of the Document ... 9
1.3 Structure of the Document ... 9
2 UPCAST Workflow ... 10
3 Background Technologies and Architectures ... 14
3.1 Open Technologies and Architectures ... 14
3.2 UPCAST Marketplaces ... 17
4 UPCAST Architecture ... 22
5 UPCAST Plugins and Interfaces ... 24
5.1 Resource Specification and Profiling ... 24
5.2 Semantic Profiling ... 32
5.3 Resource Publishing ... 33
5.4 Resource Discovery ... 34
5.5 Privacy and Usage Policy Plugin ... 34
5.6 Environmental Impact ... 44
5.7 Pricing ... 45
5.8 Data Processing Workflow Modelling ... 49
5.9 Federated Machine Learning ... 53
5.10 Data Integration and Exchange ... 55
5.11 Negotiation and Contracting ... 57
5.12 Secure Data Exchange ... 67
5.13 Monitoring ... 71
5.14 Compliance ... 75
5.15 Safe and Secure Execution ... 75
6 UPCAST Pilot Descriptions ... 80
6.1 Biomedical and Genomic Data Sharing ... 80
6.2 Digital Marketing Data and Resources ... 81
6.3 Digital Marketing Data and Resources ... 84
6.4 Sharing Public Administration for Climate ... 85
6.5 Health and Fitness Data Trading ... 86
7 UPCAST Integration with Marketplaces ... 88
7.1 Nokia Marketplace ... 88
7.2 Dawex Data Exchange ... 89

8 Conclusions 90

9 References 91

ACRONYMS 93

List of Figures

Figure 1: Core functionality of UPCAST MVP. 11
Figure 2: CKAN Architecture. 15
Figure 3: IDSA Reference Architecture Model. 17
Figure 4: Overview of the NOKIA Data Marketplace. 18
Figure 5: Nokia Data Marketplace Architecture. 18
Figure 6: Nokia Data Marketplace access workflow. 19
Figure 7: Dawex marketplace environment architecture. 19
Figure 8: Conformity of the Dawex API to standards. 20
Figure 9: Summary of Dawex features. 21
Figure 10: Typical uses of the Dawex marketplace. 21
Figure 11: UPCAST Architecture. 22
Figure 12: Provider Dashboard. 24
Figure 13: Provider Dashboard, List of Datasets. 25
Figure 14: Provider Dashboard, Dataset Creation, Step 1. 27
Figure 15: Provider Dashboard, Dataset Creation, Step 2. 27
Figure 16: Provider Dashboard, Dataset Creation, Step 3. 28
Figure 17: Provider Dashboard, Dataset menu. 29
Figure 18: Provider Dashboard, Dataset annotation plugins. 29
Figure 19: Provider Dashboard, List of dataset negotiations. 30
Figure 20: Provider Dashboard, Negotiations menu. 30
Figure 21: Provider Dashboard, List of datasets. 31
Figure 22: Provider Dashboard, New Dataset, Step 1. 31
Figure 23: Provider dashboard, Updated list of datasets. 32
Figure 24: Data Profiling API. 32
Figure 25: Data Model of Data Profiling. 33
Figure 26: Sequence Diagram of Data Profiling plugin. 33
Figure 27: Privacy and Usage Policy GUI, snapshot 1. 35
Figure 28: Privacy and Usage Policy GUI, snapshot 2. 35
Figure 29: Privacy and Usage Policy GUI, snapshot 3. 36
Figure 30: Data model of Rules showing modelling purposes as constraints. 39
Figure 31: Rule Creation form, general rule information and definition of applicable actions. 42
Figure 32: Rule Creation form - defining preActions and constraints upon action entities. 42

Figure 33: goodFlows access and usage control rule data model. 43
Figure 34: The environmental plugin swagger interface. 44
Figure 35: Data model of the Environmental plugin. 45
Figure 36: UPCAST Static pricing plugin architecture. 46
Figure 37: OpenAPI3.0 of the Static Pricing plugin. 46
Figure 38: UPCAST Static Pricing interface. 47
Figure 39: UPCAST Static Pricing functions to obtain similar datasets in the market. 48
Figure 40: DPW modelling. 49
Figure 41: Dataset discovery 50
Figure 42: Properties of the discovered dataset. 50
Figure 43: Conflict identification prior to negotiation. 51
Figure 44: DPW API. 51
Figure 45: DPW Data model. 52
Figure 46: Interactions with the DPW plugin. 53
Figure 47: Federated Machine Learning plugin high level design. 55
Figure 48: Resource Specification. 58
Figure 49: Negotiations GUI. 59
Figure 50: Consumer Dashboard for datasets available for negotiations. 59
Figure 51: Consumer Dashboard for datasets available for negotiations. 60
Figure 52: Negotiations plugin API specification. 61
Figure 53: Negotiations plugin API specification. 62
Figure 54: Negotiation plugin Data Model. 63
Figure 55: One to one data exchange scenario. 69
Figure 56: One to many data exchange scenario. 69
Figure 57: Safe data delivery Plugin Architecture. 71
Figure 56: Table containing the user request for data set generation. 77
Figure 57: User interface showing the status of the user request. 78
Figure 58: Frontend interface elements of SIMPIPE. 79
Figure 61: Cancer Genomics workflow 81
Figure 62: “In-vitro gene expression” workflow. 81
Figure 63: Digital Marketing data specifications. 82
Figure 64: Business Case workflow model. 82
Figure 65: Digital Marketing workflows. 84
Figure 66: NDM REST API. 88

List of Tables

Table 1: Additional requirements for Negotiation. 67
Table 2: Dawex Secure Data Delivery capabilities. 69

1 Introduction

UPCAST, Universal Platform Components for Safe Fair Interoperable Data Exchange, Monetisation and Trading, provides a set of universal, trustworthy, transparent and user-friendly data market plugins for the automation of data sharing and processing agreements between businesses, public administrations and citizens. The UPCAST plugins will enable actors in the common European data spaces to design and deploy data exchange and trading operations guaranteeing:

- automatic negotiation of agreement terms;
- dynamic fair pricing;
- improved data-asset discovery;
- privacy, commercial and administrative confidentiality requirements;
- low environmental footprint;
- compliance with relevant legislation;
- ethical and responsibility guidelines;
- Accountability, auditing, and compliance of dataset executions.

UPCAST will support the deployment of Common European data spaces by consolidating and acting upon mature research in the areas of data management, privacy, monetisation, exchange and automated negotiation, considering efficiency for the environment as well as compliance with EU and national initiatives, AI regulations and ethical procedures. Five real-world pilots across Europe will exercise a set of working platform plugins for data sharing, monetisation and trading, deployable across a variety of different data marketplaces and platforms, ensuring digital autonomy of data providers, brokers, users and data subjects, and enabling interoperability within European data spaces. UPCAST aims at engaging SMEs, administrations and citizens by providing a transferability framework, best practices and training to endow users to deploy the new technologies and maximise impact of the project.

The work reported in this deliverable has been carried out in Work Package 1, UPCAST concept and MVP definition, which addresses the following project objectives of the project:

- Objective 1: Apply models and standards to easily specify data processing requirements in the context of common European data spaces,
- Objective 4: Enable interoperability of data sharing across different entities, platforms and marketplaces,
- Objective 5: Provide a legal and ethical framework for automated contracts, and,
- Objective 8: Pilot and evaluate the platform in Real Market Dataspaces.

These project objectives will be achieved by WP1 through the following sub-objectives:

- Sub-objective 1.1 Establish the vision and direction for the project by defining a Minimum Viable Product (MVP) and agreeing the technical and pilot requirements and usage scenarios to achieve sustainability of the UPCAST set of tools. A methodology to define the requirements and the MVP will be used to drive the process.
- Sub-objective 1.2 Define the data model and vocabularies for expressing the UPCAST workflows, preferences and other features based on extending existing efforts in GAIA-X and IDS.
- Sub-objective 1.3 Provide a legal framework based on European and National regulations, best practices and ethics guidelines for UPCAST.
- Sub-objective 1.4 Define the UPCAST system architecture using best practices on architecture specification in compliance with the data spaces, along with legal and ethical aspects.

1.1 Purpose of the Document

This document reports on the updated UPCAST concept and the updated architecture after its elaboration and refinement following the analysis of the pilot and legal requirements, details and technical capabilities of the marketplaces that will be used in the project and updates in the project plugin technologies. D1.3 is a follow-up of Deliverable D1.1 and Deliverable D1.2, which presents the first version of the UPCAST architecture and the UPCAST MVP. D1.3 documents the technical design of the refined UPCAST architecture and pilot demonstrations through the use of semiformal models and formal specifications. It also describes the final version of vocabularies and data models that will be used by the UPCAST plugins and the pilot demonstrations.

1.2 Scope of the Document

Deliverable D1.3 is a follow-up of Deliverable D1.1 that reports on the UPCAST concept and requirements setup and D1.2 that reports on the UPCAST MVP and the UPCAST Architecture. Similar to D1.2 it reports on the work that has been carried out in the following tasks:

- Task 1.1 (MVP Definition and Requirements for the Data Value Chain): definition of the main features to be delivered in the UPCAST MVP;
- Task 1.2 (Pilot Design and Functionalities): the final pilot design and functionalities based on elaboration of the initial pilot use cases and requirements;
- Task 1.3 (Vocabulary and Data Model): the initial input related to the vocabulary and data model to be used in UPCAST MVP and pilots.
- Task 1.5 (UPCAST Tools Design and Architecture): the initial UPCAST architecture and interface to develop the MVP.

D1.3 documents the update on the UPCAST pilot design and functionality using technical models and diagrams that have been presented in D1.2.

1.3 Structure of the Document

The document is structured as follows: Chapter 2 gives an update of the UPCAST MVP. Chapter 3 gives an overview of background technologies and architectures on which UPCAST is based. Chapter 3.2.2.1 gives the technical specifications of the interfaces and data models of the UPCAST plugins. Chapter 6 gives the descriptions and requirements of the UPCAST pilots. Chapter 4 gives the updated UPCAST architecture and Chapter 7 gives the technical details for integration of the UPCAST plugins to the two marketplaces of the project. Finally, Chapter 8 concludes the document.

2 UPCAST Workflow

UPCAST provides support for the management, negotiation, and exploitation of resources through a set of plugins that can be installed in Data Marketplaces or other data sharing platforms that can mediate data transactions between providers and consumers. A resource can be a dataset, a data operation or an artefact (such as a machine learning model). This chapter gives an overview of a typical workflow by a dataset provider and a dataset consumer when they interact through the UPCAST platform.

The UPCAST Minimum Viable Product (MVP) is an implementation of the minimum functionality of the UPCAST plugins (described in [1]) and integrated into a platform that satisfies the prioritised requirements that have been selected based on the pilots' needs and project vision. The MVP will serve to gather valuable feedback for further development of the UPCAST platform. The architecture of the UPCAST platform is given in chapter 4.

This chapter presents the interactions with the UPCAST MVP (UPCAST plugins and platform) from a user perspective. In the context of this presentation, users of UPCAST are either dataset providers or dataset consumers. Figure 1 illustrates the interactions with the core functionality that is offered by the UPCAST MVP as a set of functions performed by or provided to either the Dataset (Resource) Provider, the Dataset (Resource) Consumer, or in some cases to both. The figure shows a typical sequence of actions the dataset provider and the dataset consumer take as well as the components (plugins) that support these actions.

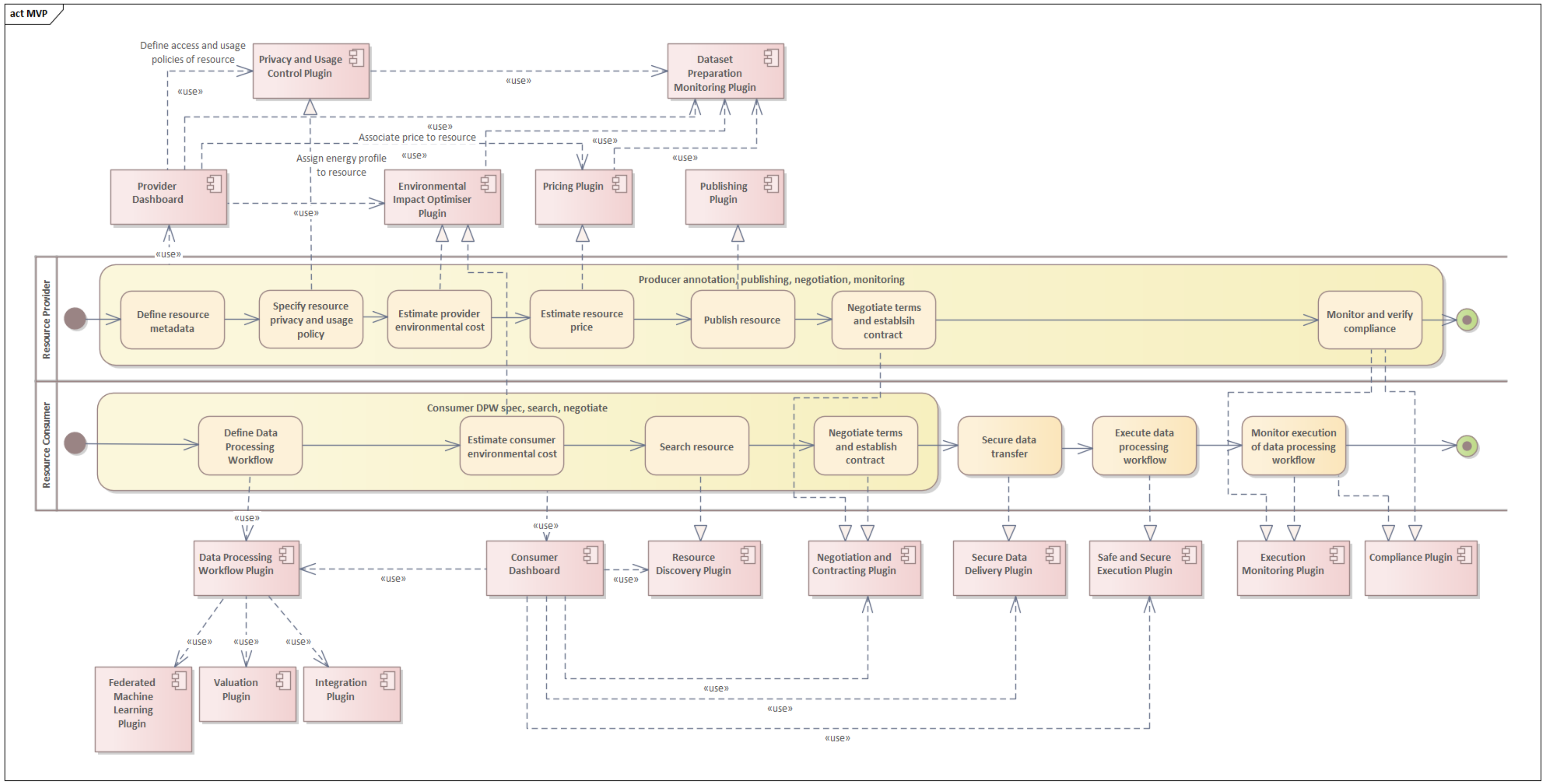

act MVP
Define access and usage policies of resource
Privacy and Usage Control Plugin
«use»
Dataset Preparation Monitoring Plugin
Associate price to resource
Assign energy profile to resource
Provider Dashboard
Environmental Impact Optimiser Plugin
Pricing Plugin
Publishing Plugin
Resource Provider
Producer annotation, publishing, negotiation, monitoring
Define resource metadata
Specify resource privacy and usage policy
Estimate provider environmental cost
Estimate resource price
Publish resource
Negotiate terms and establsih contract
Monitor and verify compliance
Resource Consumer
Consumer DPW spec, search, negotiate
Define Data Processing Workflow
Estimate consumer environmental cost
Search resource
Negotiate terms and establish contract
Secure data transfer
Execute data processing workflow
Monitor execution of data processing workflow
Data Processing Workflow Plugin
Consumer Dashboard
Resource Discovery Plugin
Negotiation and Contracting Plugin
Secure Data Delivery Plugin
Safe and Secure Execution Plugin
Execution Monitoring Plugin
Compliance Plugin
Federated Machine Learning Plugin
Valuation Plugin
Integration Plugin

Figure 1: Interactions with the UPCAST MVP.

For the UPCAST MVP definition the focus is on datasets, but some plugins are applicable for other types of resources. The UPCAST plugins are modules that can be deployed on a data marketplace (or other data sharing platforms) offer well-defined functionalities that enhance or complement those in the host marketplace. Plugins interact with each other and with the marketplace in which they are deployed through well-defined APIs. The users, i.e., resource providers or resource consumers, can use those plugins that suit their needs and invoke them through the provided interface (depicted as provider and consumer dashboard in Figure 1).

Figure 1 shows a representative user journey with activities that involve all of the UPCAST plugins. The upper part of Figure 1 illustrates the actions of a resource provider who wants to annotate and publish a dataset[1] using UPCAST plugins. The preparation of a dataset (collection of data, cleaning, and preprocessing) is a necessary action any provider needs to take but it falls outside the scope of UPCAST. Therefore, the provider actions start with the dataset annotation by which the provider describes the resource using basic metadata or semantic metadata and defines access and usage policies. The provider may also assign an environmental profile to the resource that relates to the energy and carbon consumption of its generation and storage and may also associate a price or price range to it to facilitate its monetisation. A typical sequence of actions by a provider who uses the UPCAST plugins is as follows:

RP1. *Define resource metadata*: Using the Provider Dashboard, the provider creates a resource specification and annotates the resource with basic and semantic metadata using UPCAST vocabulary and domain-specific vocabularies.

RP2. *Specify resource privacy and usage policy*: Using the Privacy and Usage Control Plugin through the Provider Dashboard the provider can define the privacy and usage control policies for the resource.

RP3. *Estimate resource environmental cost*: Using the Provider Dashboard, the provider can create the environmental profile for the resource. For datasets, this relates to the collection and storage cost. For data operations, this relates to the cost of executing the operation with the support of the Environmental Impact Optimiser Plugin.

RP4. *Estimate resource price*: Using the Provider Dashboard, the dataset provider can assign a price or price range to the resource. The functions of the Pricing plugin may be used to generate an informed price suggestion.

RP5. *Publish resource*: The dataset provider publishes the resource annotated with the resource specification in a data marketplace or a data catalogue provided by a broker so that potential consumers can discover the resource. This functionality is provided by a broker or a marketplace. UPCAST Publishing/Discovery plugins may be used for this functionality.

RP6. *Negotiate terms and establish contract*. The dataset provider and the dataset consumer may need to negotiate terms of the policies expressed by the provider and the usage intentions expressed by the consumer. Negotiation is an iterative process supported by the Negotiation Plugin, which, if successful, will result into a contract that forms the basis for a data processing workflow execution and verification for compliance.

RP7. Monitor dataset execution and verify compliance. The dataset provider receives in their dashboard continuous monitoring data from the execution of the dataset. Monitoring data are used for checking compliance of the execution against the agreed terms of the contract and any violations are notified to the provider. The same

[1] This chapter focuses on datasets, but some plugins are applicable to other types of resources.

monitoring data may be fed to supporting modules for generating analytics.

The lower part of Figure 1 illustrates the actions of a resource consumer who wants to make use of a dataset resource. The first action of a consumer is typically to Define a Data Processing Workflow (DPW, RC1) which may utilise one or more datasets (possibly from several providers) to do the processing they need. The DPW may involve generic actions, like transformations or aggregations on datasets, and also specialized actions like performing Federated Machine Learning (FML) on datasets that are not allowed to be transferred outside the domain of a dataset provider, valuation of a dataset, and integration of several datasets collected possibly from multiple providers. These actions are implemented by respective components as shown in Figure 1, and are not represented in the overall activities of the dataset consumer, as they are special steps of the DPW the consumer models.

RC1. *Define Data Processing Workflow*: The consumer defines the processing workflow for the dataset as a series of actions that pertain to the pre-processing and actual processing of datasets using the Data Processing Workflow plugin. A DPW model is defined, and the intended usage and the access and usage policies for the DPW are specified.

RC2. *Estimate consumer environmental cost*. The consumer makes an estimate of the environmental cost that will be incurred when processing the dataset. The cost is estimated based on the workflow, and the characteristics of the processing environment that will be used.

RC3. *Search resource*: The consumer searches and discovers resources to include in the DPW by searching or browsing a Dataset Catalogue or getting suggestions on relevant resources using the Resource Discovery plugin.

RC4. *Negotiate terms and agree on contract*: The dataset consumer negotiates with resource providers regarding the terms of access, usage and pricing of the datasets. The result of the negotiation, if successful, is a contract that states the terms of access and usage, as well as the pricing of the dataset under negotiation. The negotiation and contracting tasks are supported by the corresponding plugin that facilitates and automates the negotiation process and can be used by the dataset producer (see RP6) and consumer.

RC5. *Secure data exchange*: With the use of the Secure Data exchange plugin, the dataset contracted will be transferred securely to a trusted environment, which in the case of the UPCAST pilots is the consumer one, for processing.

RC6. *Execute data processing workflow*: Using Safe and Secure Execution Plugin, the consumer starts the DPW execution for the processing of the dataset subject to the terms of access and usage policies that have been negotiated and agreed between the provider and the consumer and are expressed in the negotiation contract.

RC7. *Monitor execution of data processing workflow*: The UPCAST Execution Monitoring plugin monitors the execution of the DPW. The collected monitoring data are used for generating analytics for the provider and also for checking the compliance with the agreed contract. The compliance plugin receives monitoring data and notifies the dataset provider in case of any breaches of the contract (RP6), such as any access or usage rule violated during the DPW execution.

The interactions with the UPCAST MVP of Figure 1 involve an overarching set of activities that are foreseen by UPCAST that relate to the actions of the dataset provider and the dataset consumer. Chapter 6 gives details of interactions with the UPCAST MVP for each of the project pilots.

3 Background Technologies and Architectures

This chapter presents open technologies and architectures that are relevant to UPCAST and are key to the developments of the project. OpenAPI, an open formalism for specifying HTTP interfaces, CKAN, an open-source Data Management Service, and IDSA Reference Architecture, an open dataspaces architecture, are given in section 3.1. Moreover, the two data marketplace platforms that are available to the project are presented in section 3.2.

3.1 Open Technologies and Architectures

3.1.1 OpenAPI

The OpenAPI[2] is a widely used API description language. The OpenAPI Specification, formerly known as Swagger, is an open specification for building machine-readable APIs for web services. It provides a standardized way to describe REST APIs in a programming-language agnostic manner, allowing humans and computers to discover and understand the capabilities of the services without accessing the source code.

An OpenAPI document provides the interface description according to the OpenAPI specification and describes the endpoints, methods (such as GET, POST, PUT and DELETE), and data schema definitions of the API. It specifies the operations that an API can perform, the parameters and data models it accepts, and the expected responses. Typically, the OpenAPI documents are written in YAML or JSON.

The OpenAPI plays a key role in the API lifecycle. There is a wide range of OpenAPI tools[3] that facilitate the automatic generation of code, documentation, and test cases from OpenAPI documents, such as OpenAPI editors, code generators, validators, and testing tools. For example, tools are available to for automatically generate service- and client-side codes in different programming languages from the OpenAPI documents, both accelerating the interface implementation and ensuring the consistency between interface design and the implementation.

3.1.2 CKAN

CKAN[4] (Comprehensive Knowledge Archive Network) is an open-source (Data Management System (DMS) for powering data hubs and data portals. CKAN is an evolution of the Debian Linux package management capabilities, which is used by public institutions seeking to share their data with the general public.

The CKAN technology shown in Figure 2[5] involves several core components:

- Routes and Views: Routes map URLs to views, which process requests and render responses using Jinja2 templates. Views interact with the backend through action functions.
- Logic Layer: This includes action functions, auth functions, and validation schemas. Action functions handle CRUD operations and are exposed via the API.
- Models: SQLAlchemy is used for database interactions, but access should be through model methods, not directly from other packages.

[2] https://www.openapis.org/what-is-openapi

[3] https://tools.openapis.org/

[4] https://ckan.org/

[5] https://docs.ckan.org/en/2.9/contributing/architecture.html

- Plugins: CKAN's functionality is extensible through plugins, which can define routes, views, and override core functions.

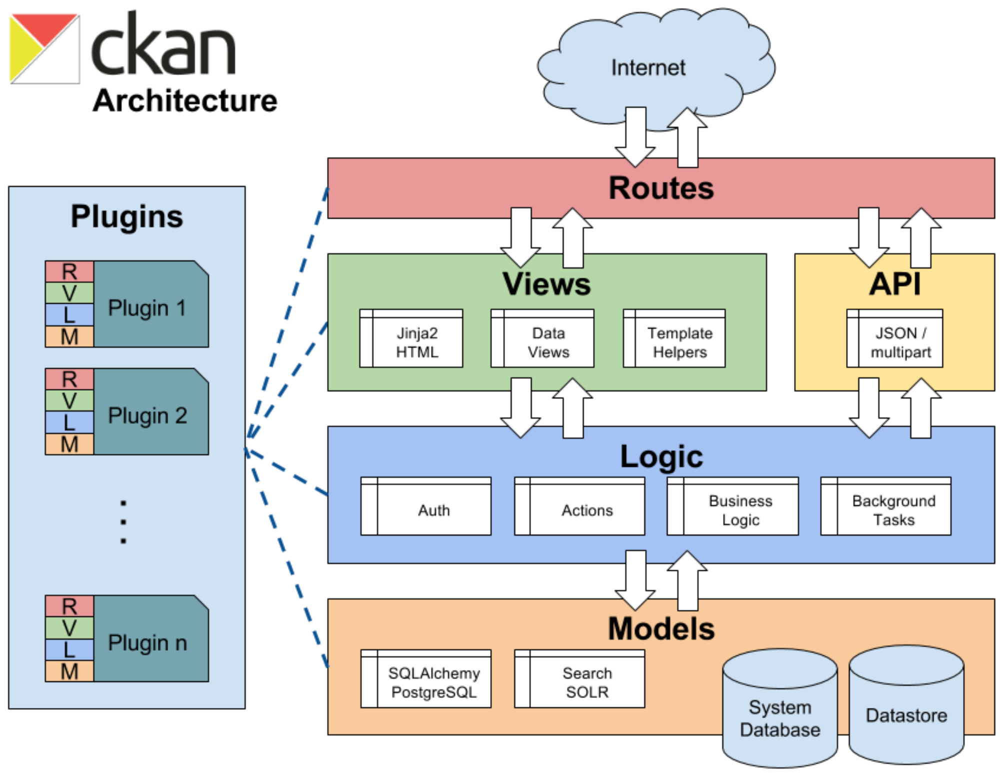


Figure 2: CKAN Architecture.

CKAN provides a powerful API for accessing and managing datasets. The API allows users to interact with CKAN programmatically, enabling the creation, updating, and querying of datasets and resources. Key Features of the CKAN API include:

- Action API: This is the primary API for interacting with CKAN. It uses a JSON-based protocol to perform various actions like creating datasets, updating metadata, and retrieving data. The actions are categorized into core groups:
- Dataset and Resource Management: Create, update, and search for datasets and resources.
- User and Organization Management: Manage user accounts, organizations, and memberships.
- Tag and Group Management: Manage tags and groups associated with datasets.
- RESTful Interface: CKAN's RESTful interface allows for straightforward HTTP operations (GET, POST, etc.) on datasets, resources, and other objects. This makes it easy to integrate CKAN with other web services and applications.
- Search API: The search functionality is robust, allowing for detailed queries on datasets using Solr's powerful search capabilities. This includes full-text search, faceted search, and filtering based on metadata fields.
- Authentication: The API supports multiple authentication mechanisms, including API keys and session-based authentication, ensuring secure access to the data and operations.

- Extensions and Plugins: CKAN’s API is extensible, allowing developers to create custom endpoints and integrate additional functionality through extensions.

3.1.3 IDSA Reference Architecture Model

The International Data Spaces Association[6] (IDSA) is a coalition of more than 150 member companies from 28 countries that share the vision by which companies self-determine data usage rules and realize the full value of their data in secure, trusted, equal partnerships. The goal of IDSA is the definition of a global standard for sovereign data spaces and interfaces, as well as fostering the related technologies and business models that will drive the data economy of the future across industries.

IDSA focuses on the value of future data-driven global, digital economies whose functions are expected to be based on the use of International Data Spaces (IDS). IDSs provide secure environments for data sharing, which ensure that the self-determined control of data use (data sovereignty) remains in the hands of data providers, and in which all participants can realize the full value of their data.

The IDSA Reference Architecture Model[7] is depicted in Figure 3. The model is made up of five layers:

- The Business Layer specifies and categorizes the different roles which the participants of the International Data Spaces can assume, and it specifies the main activities and interactions connected with each of these roles.
- The Functional Layer defines the functional requirements of the International Data Spaces, plus the concrete features to be derived from these.
- The Process Layer specifies the interactions taking place between the different components of the International Data Spaces; using the BPMN notation, it provides a dynamic view of the Reference Architecture Model.
- The Information Layer defines a conceptual model which makes use of linked-data principles for describing both the static and the dynamic aspects of the International Data Space’s constituents.
- The System Layer is concerned with the decomposition of the logical software components, considering aspects such as integration, configuration, deployment, and extensibility of these components.

In addition, the Reference Architecture Model comprises three perspectives that need to be implemented across all five layers: Security, Certification, and Governance.

[6] https://internationaldataspaces.org/

[7] https://internationaldataspaces.org/wp-content/uploads/IDS-Reference-Architecture-Model-3.0-2019.pdf

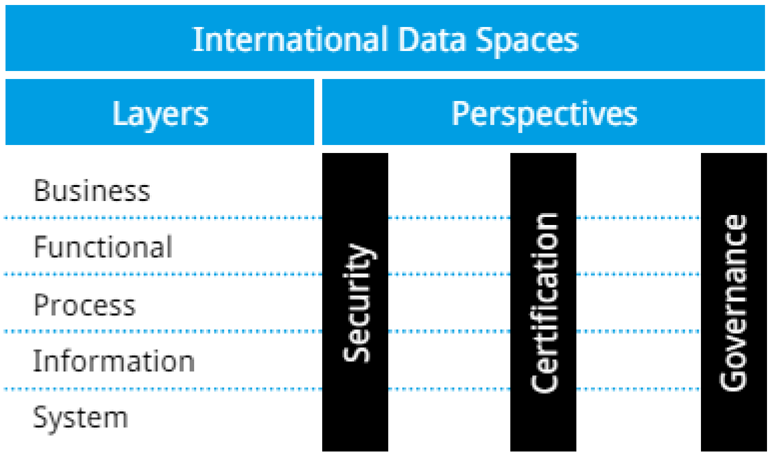


Figure 3: IDSA Reference Architecture Model.

3.2 UPCAST Marketplaces

This section gives technical details of the two marketplaces that are available in the project, the NOKIA Marketplace and the Dawex Marketplace.

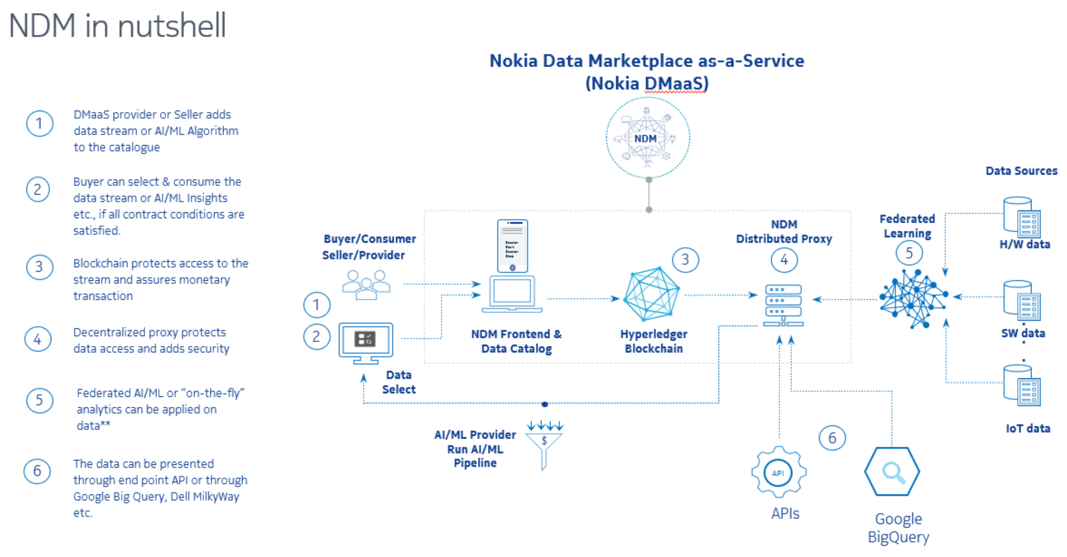
3.2.1 Nokia Open Analytics Exchange

The Nokia Data Marketplace NDM, also referred to as Nokia Open Analytics Exchange, solution (Figure 4) is ideally suited for connecting organizations that want to exchange data with each other to optimize both internal and value chain processes. These collaborating organizations define a business ecosystem.

The Nokia Data Marketplace is a decentralized data marketplace, powered by Blockchain technology, for the safe and automated exchange of digital assets in the form of data streams. The exchanged data streams are monetized and supported by technical and policy-based data verification. The unique functions of the Nokia Data Marketplace, both for generating revenue and for the exchange of data streams between interested parties in a business ecosystem, are

- A private and permissioned Blockchain-technology that ensures network security, data integrity, the use of smart contract, for fast and automated transactions (data streams) and the use of micropayments based on a token economy
- Data verification, technical and policy-based via a hardware certification program and with Nokia partnerships.

The Nokia Data Marketplace can be used to stream any type of data from any source type such as administrative, IoT devices, physical assets, autonomous cars, drones, and many more. Furthermore, it enables the business ecosystem to integrate 3rd party data and to monetize it through the same marketplace. Analyses and reporting are provided on the basis of the Nokia Data Marketplace computing capabilities, about the continuous transformation in both the business ecosystem and the data assets. The Nokia Data Marketplace integrates various 3rd party or Nokia products for orchestration of machine learning or Artificial Intelligence solutions.



NDM in nutshell

1. DMaaS provider or Seller adds data stream or AI/ML Algorithm to the catalogue
2. Buyer can select & consume the data stream or AI/ML Insights etc., if all contract conditions are satisfied.
3. Blockchain protects access to the stream and assures monetary transaction
4. Decentralized proxy protects data access and adds security
5. Federated AI/ML or "on-the-fly" analytics can be applied on data**
6. The data can be presented through end point API or through Google Big Query, Dell MilkyWay etc.

Nokia Data Marketplace as-a-Service (Nokia DMaaS)
NDM
Buyer/Consumer Seller/Provider
NDM Frontend & Data Catalog
Hyperledger Blockchain
NDM Distributed Proxy
Federated Learning
Data Sources
H/W data
SW data
IoT data
Data Select
AI/ML Provider Run AI/ML Pipeline
APIs
Google BigQuery

Figure 4: Overview of the NOKIA Data Marketplace.

The architecture of the NDM is built based on microservice architecture as shown in Figure 5 and it facilitates the integration of additional feature on demand.

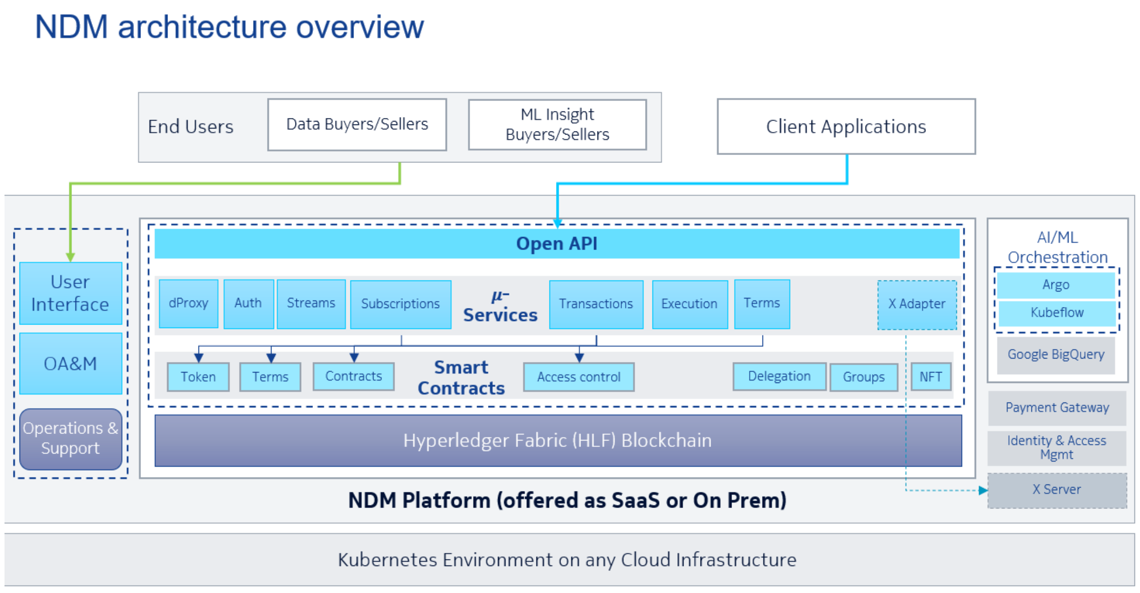


Figure 5: Nokia Data Marketplace Architecture.

There is no data storage on NDM and it never stores or processes the provider's data. The access to the data from the consumer is done through a hashed URL which is going through NDM dProxy service as shown in Figure 6.

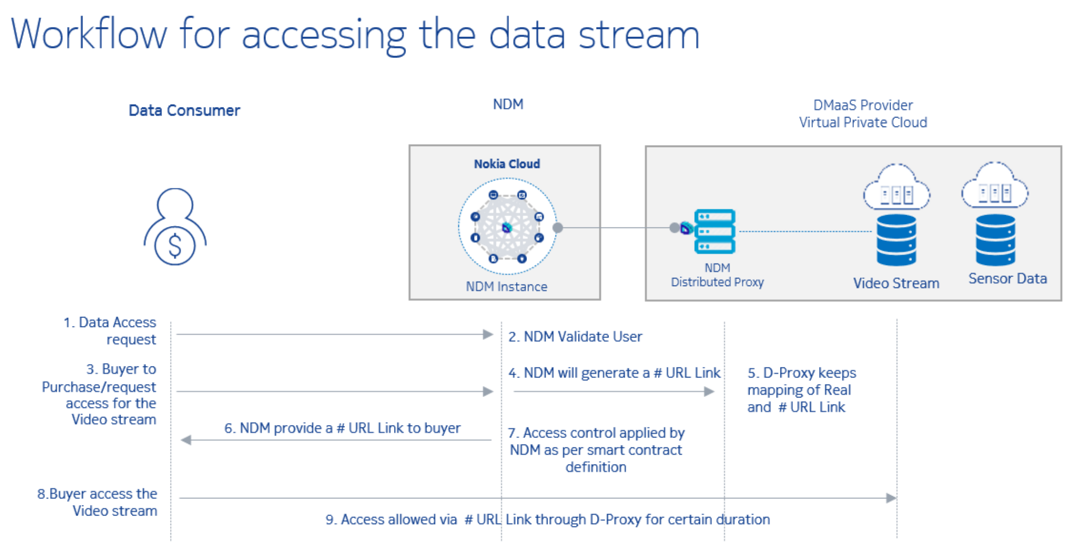

Workflow for accessing the data stream
Data Consumer
NDM
DMaaS Provider
Virtual Private Cloud
Nokia Cloud
NDM Instance
NDM
Distributed Proxy
Video Stream
Sensor Data
1. Data Access request
2. NDM Validate User
3. Buyer to Purchase/request access for the Video stream
4. NDM will generate a # URL Link
5. D-Proxy keeps mapping of Real and # URL Link
6. NDM provide a # URL Link to buyer
7. Access control applied by NDM as per smart contract definition
8.Buyer access the Video stream
9. Access allowed via # URL Link through D-Proxy for certain duration

Figure 6: Nokia Data Marketplace access workflow.

For the needs of UPCAST, a dedicated instance of NDM has been deployed and is available in the Nokia cloud through the link https://upcast.dataexchange.nokia.com/.

3.2.2 Dawex Data Exchange Platform

This section give an overview of the Dawex Data Exchange platform[8]. The architecture of the platform and its main features are presented.

3.2.2.1 The Dawex Data Marketplace environment architecture

The Dawex Data Marketplace environment architecture is shown in Figure 7.

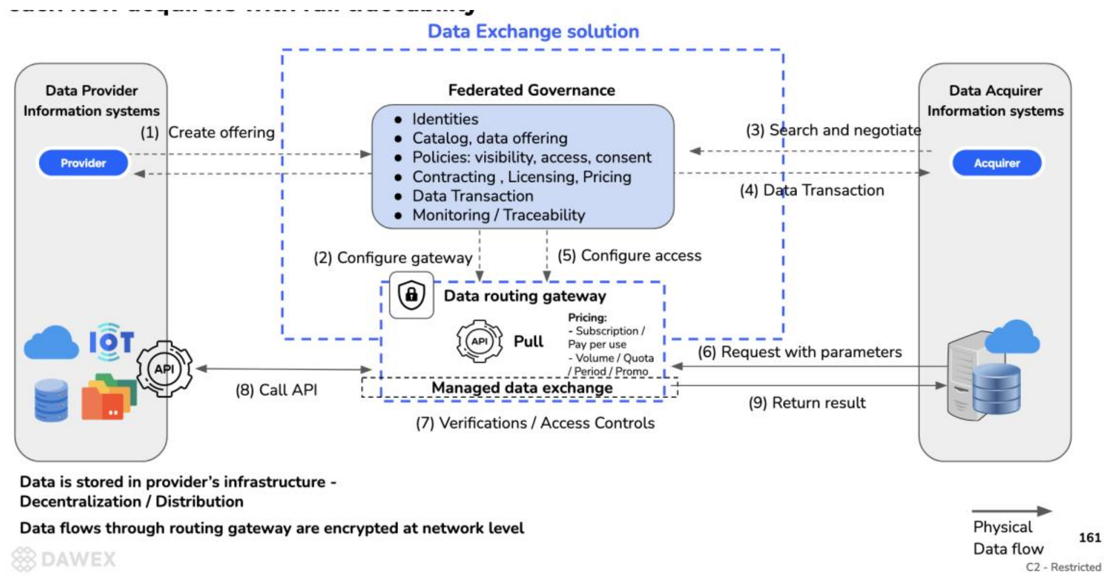
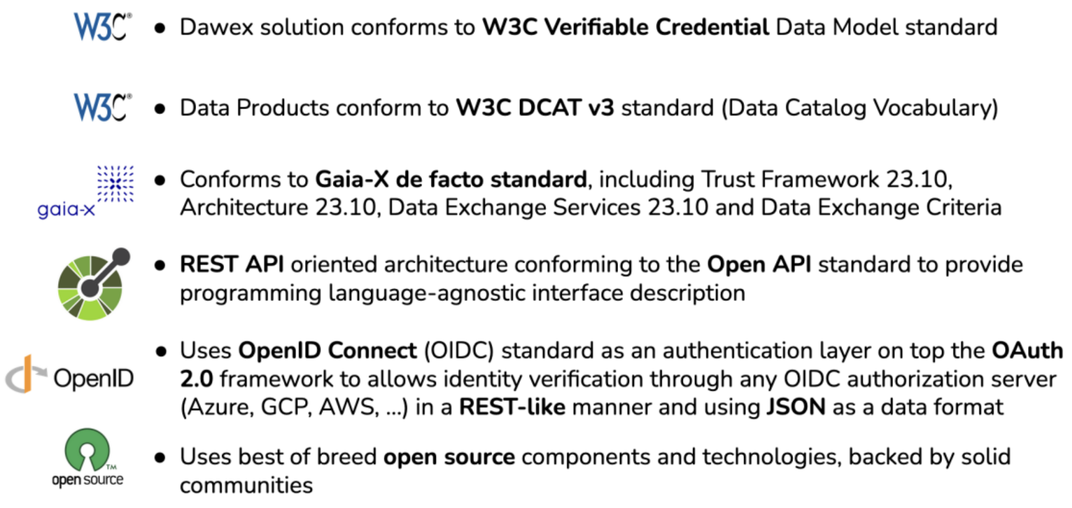

Data Exchange solution
Data Provider Information systems
Provider
(1) Create offering
Federated Governance
Identities
Catalog, data offering
Policies: visibility, access, consent
Contracting , Licensing, Pricing
Data Transaction
Monitoring / Traceability
(3) Search and negotiate
(4) Data Transaction
Data Acquirer Information systems
Acquirer
(2) Configure gateway
(5) Configure access
Data routing gateway
Pull
Pricing:
- Subscription / Pay per use
- Volume / Quota
/ Period / Promo
Managed data exchange
(6) Request with parameters
(8) Call API
(9) Return result
(7) Verifications / Access Controls
IOT
API
Data is stored in provider's infrastructure - Decentralization / Distribution
Data flows through routing gateway are encrypted at network level
DAWEX
Physical Data flow
161
C2 - Restricted

Figure 7: Dawex marketplace environment architecture.

Dawex APIs rely on the use of common standards, as shown in Figure 8, to ease the interoperability with third party services and technologies providers. By conforming to these

[8] https://www.dawex.com/en/data-exchange-platform/

standards, the plugins will be able to easily interact with the Dawex Data Marketplace environment provided to UPCAST.



- Dawex solution conforms to **W3C Verifiable Credential** Data Model standard
- Data Products conform to **W3C DCAT v3** standard (Data Catalog Vocabulary)
- Conforms to **Gaia-X de facto standard**, including Trust Framework 23.10, Architecture 23.10, Data Exchange Services 23.10 and Data Exchange Criteria
- **REST API** oriented architecture conforming to the **Open API** standard to provide programming language-agnostic interface description
- Uses **OpenID Connect** (OIDC) standard as an authentication layer on top the **OAuth 2.0** framework to allows identity verification through any OIDC authorization server (Azure, GCP, AWS, ...) in a **REST-like** manner and using **JSON** as a data format
- Uses best of breed **open source** components and technologies, backed by solid communities

Figure 8: Conformity of the Dawex API to standards.

3.2.2.2 Data Exchange Platform technology

Dawex Data Exchange technology delivers all necessary capabilities to manage data exchanges under various business models, meeting compliance, security and privacy requirements. A data exchange platform addresses the needs of the orchestrator operating the platform as a trusted intermediary, as well as the needs of the data providers and data acquirers engaged in trusted data transactions.

The Data Marketplace environment will integrate four universes to cover the overall data exchange lifecycle:

- **Explore**: All exploration and search capabilities reunited in one place, from a fully configurable homepage to the exploration tools like the data offerings and services marketplace, the spatio-temporal exploration tool or the organization index.
- **Publication**: The participants can create, publish and update all their contents available through the Data Marketplace (organization public profile, data offerings, data services and data themes).
- **Transactions**: From discussion to data transaction, in this universe participants can discuss and negotiate in the forum and follow all their past data transactions both as a data provider and as a data acquirer to obtain the data and relevant documents and metrics associated with these data transactions.
- **Administration**: The administration universe integrates the participant and organization settings.

The Data Marketplace environment provided by Dawex covers hundreds of capabilities, and thousands of detailed features. An overview is shown in Figure 9.

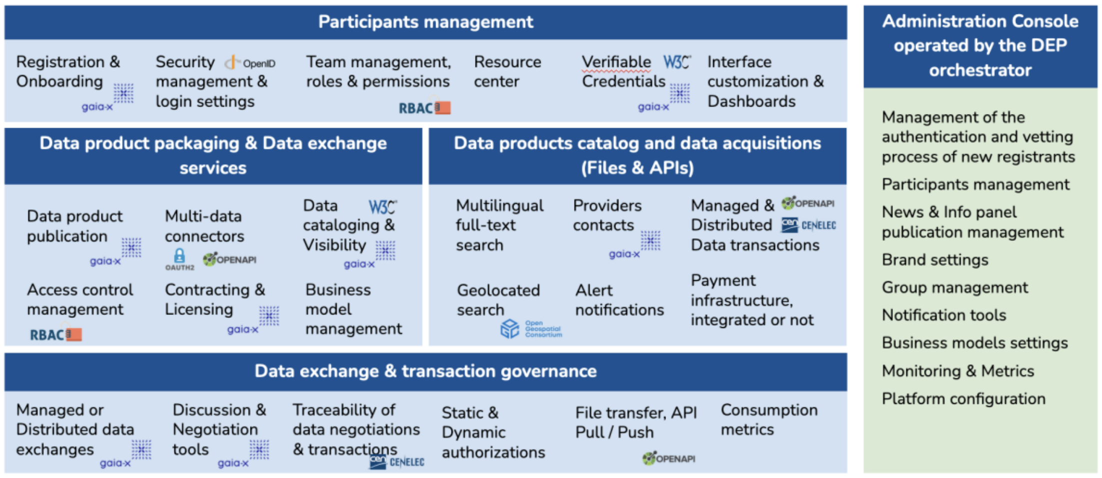

Participants management
Registration & Onboarding
Security management & login settings
Team management, roles & permissions
Resource center
Verifiable Credentials
Interface customization & Dashboards
Data product packaging & Data exchange services
Data product publication
Multi-data connectors
Data cataloging & Visibility
Access control management
Contracting & Licensing
Business model management
Data products catalog and data acquisitions (Files & APIs)
Multilingual full-text search
Providers contacts
Managed & Distributed Data transactions
Geolocated search
Alert notifications
Payment infrastructure, integrated or not
Data exchange & transaction governance
Managed or Distributed data exchanges
Discussion & Negotiation tools
Traceability of data negotiations & transactions
Static & Dynamic authorizations
File transfer, API Pull / Push
Consumption metrics
Administration Console operated by the DEP orchestrator
Management of the authentication and vetting process of new registrants
Participants management
News & Info panel publication management
Brand settings
Group management
Notification tools
Business models settings
Monitoring & Metrics
Platform configuration

Figure 9: Summary of Dawex features.

The user experience has been designed to be industrialized and scalable through a data exchange governance model providing control, access, traceability and security (Figure 10).

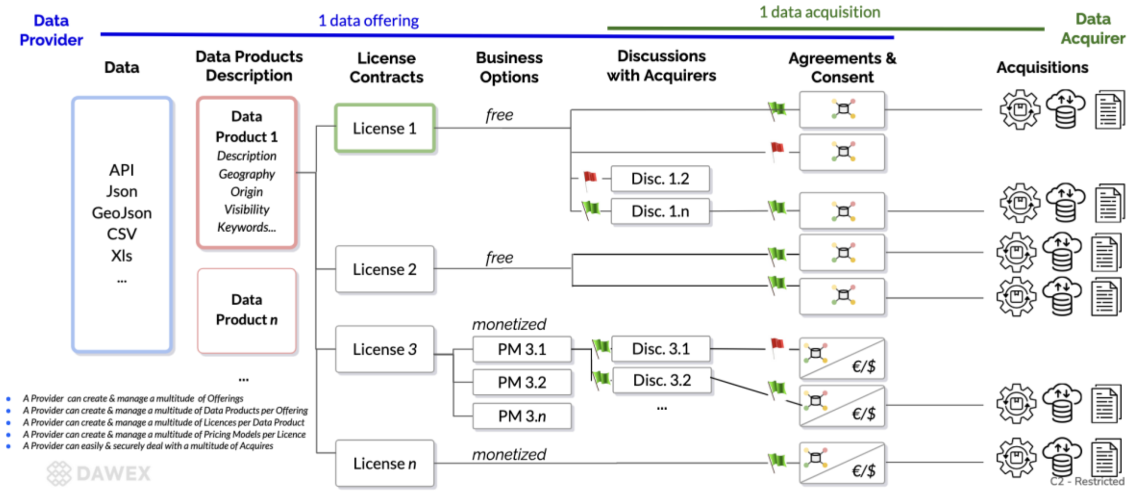

Data Provider
1 data offering
1 data acquisition
Data Acquirer
Data
Data Products Description
License Contracts
Business Options
Discussions with Acquirers
Agreements & Consent
Acquisitions
API
Json
GeoJson
CSV
Xls
...
Data Product 1
Description
Geography
Origin
Visibility
Keywords...
Data Product n
...
License 1
License 2
License 3
License n
free
free
monetized
monetized
PM 3.1
PM 3.2
PM 3.n
Disc. 1.2
Disc. 1.n
Disc. 3.1
Disc. 3.2
...
€/$
€/$
€/$
A Provider can create & manage a multitude of Offerings
A Provider can create & manage a multitude of Data Products per Offering
A Provider can create & manage a multitude of Licences per Data Product
A Provider can create & manage a multitude of Pricing Models per Licence
A Provider can easily & securely deal with a multitude of Acquires
DAWEX
C2 - Restricted

Figure 10: Typical uses of the Dawex marketplace.

4 UPCAST Architecture

This chapter presents the final version of the UPCAST Architecture. The architecture is an enhancement of the one presented in [4], and will be used for the developments and integration tasks of the project.

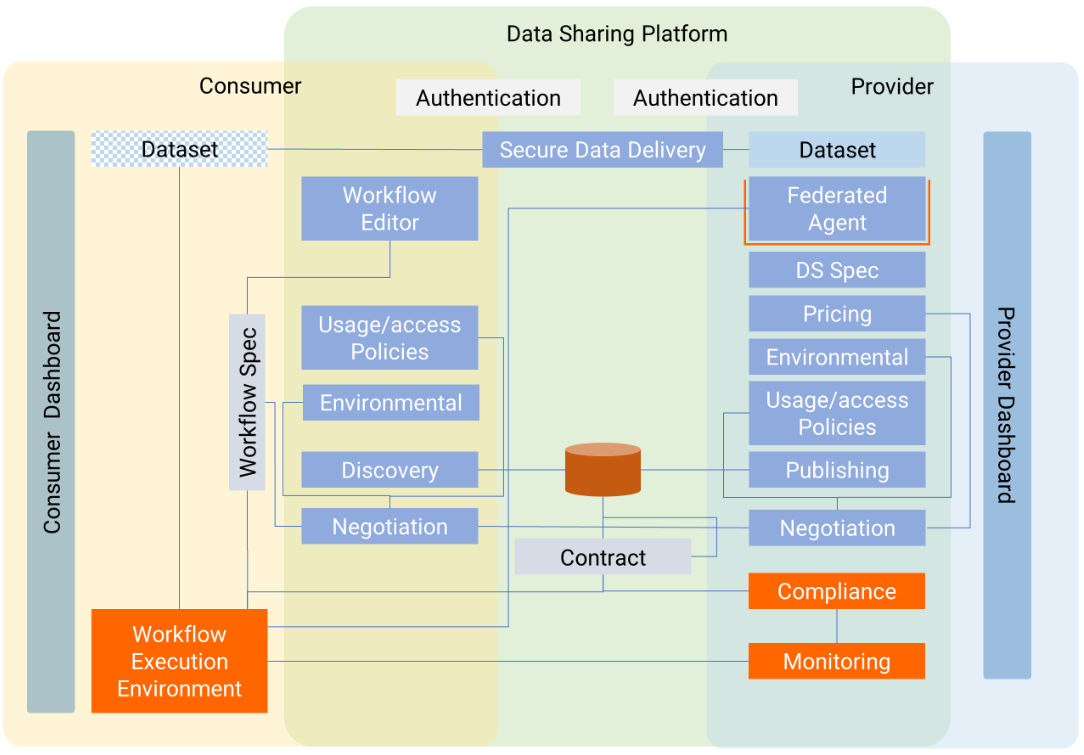


Figure 11: UPCAST Architecture.

Figure 11 shows the UPCAST Architecture. The Architecture shows the domains of the Dataset Provider, the Dataset Consumer and the Data Sharing Platform, which is an abstraction of a Data Marketplace. It is assumed that the Data Sharing Platform implements the authentication of Providers and Consumers, and also provides the hosting environment to which the components of the architecture (plugins) may be deployed or otherwise integrated. The UPCAST Architecture is centralized as the Data Sharing Platform serves as the single place of interaction between the Dataset Provider and the Dataset Consumer. UPCAST has also assessed a distributed version of the architecture, in which the dataset provider and the dataset consumer interact in a peer-to-peer manner with no intermediate Data Sharing Platform. The authentication, persistency, and hosting for plugin deployments functions that are provided by a Data Sharing Platform are crucial for the operation of the UPCAST platform itself, therefore the project has opted for the centralised version of the architecture.

The UPCAST architecture shows the plugins (and corresponding functions) that are available to the provider and those that are available to the consumer as well as dashboards for visualization. A dataset provider uses a subset of the plugins to perform the specification of the dataset resource to publish it in a marketplace. Resource specification includes the annotation of the dataset with plain (type of data, format, creation time, etc.) and semantic metadata, its environmental footprint for its storage by the provider, an estimate of its price, and definition of usage and access policies. These functions are supported by the corresponding plugins as shown in the architecture, which can be used through the Provider Dashboard. Once a dataset resource has been annotated it can be published to a marketplace for interested consumers to search for it.

The consumer uses a subset of the plugins to specify a Data Processing Workflow to model the processing they want to do on a dataset. Moreover, the consumer may also define policies that is obliged to abide with, for example internal policies or legal regulations, and can also estimate the environmental impact of the dataset execution that is modelled by the Data Processing Workflow. Once these specifications are prepared through the corresponding plugins, the consumer searches for datasets that meet their criteria. Once a dataset is discovered, a negotiation takes place between the provider of the dataset and the consumer. The negotiation is supported by the negotiation plugin and its purpose is for the provider and the consumer to agree on the same terms for the dataset execution, as, on one hand the provider has expressed his usage policies, and, on the other hand, the consumer has expressed his own policies they may be subject to for the execution of the dataset. The negotiation, if successful, will result in a contract, which, once agreed by both parties, is secured for later checking compliance of the dataset execution with the terms of the contract. The functions and respective plugins that are available to the consumer may be used through the Consumer Dashboard.

Once a negotiation is completed and a contract has been agreed and signed, execution of the data processing workflow on a dataset can be performed. The first step for the execution to commence is to transfer the dataset from the provider to the consumer space. Once the dataset has been securely transferred, a workflow execution environment is used to carry out the execution. Execution may take place in various execution environments including the consumer's space, the marketplace, a trusted third party, or in the provider itself. The UPCAST architecture has been shaped to show execution of the dataset only in the consumer space, as the result of requirements expressed by UPCAST pilots.

One exception in the consumer processing is the case of Federated Machine Learning, in which parts of the execution may take place in the provider's environment, the reason being that analytics processing of classified data may be allowed but the data itself may not be allowed to leave the provider's environment. In this case, the provider needs to provide a hosting and execution environment for containerized FML components to execute, The UPCAST architecture in Figure 11 contains a Federated Agent component, which abstracts the parts of the execution that need to take place in the provider's space. The Federated Agent component is further highlighted through the surrounding box to indicate that such components should be containerized.

Execution of the dataset is monitored through a number of metrics (detailed in section 5.13), which the execution has the obligation to emit for the producer to verify compliance with the contract.

Execution takes place in the consumer environment in a way that all elements of the execution, including the actual workflow, the (dockerized) components used, the monitoring metrics that are emitted, and so on, can be verified either in real time or in a later stage for compliance to the terms of the contract that has been agreed and that executions are reproducible and auditable. Real time monitoring and corresponding compliance is performed by the Monitoring and Compliance plugins. If any violations are detected during the execution, alerts are shown in the providers' dashboard. Moreover, analytics processing of the monitoring events that are collected during execution are also shown in the provider's dashboard.

5 UPCAST Plugins and Interfaces

This chapter reports a summary of the plugins that are developed by project partners, their functionality, interfaces, and data models. These plugins are shown in the UPCAST architecture of Figure 11 and will be integrated into the final UPCAST platform.

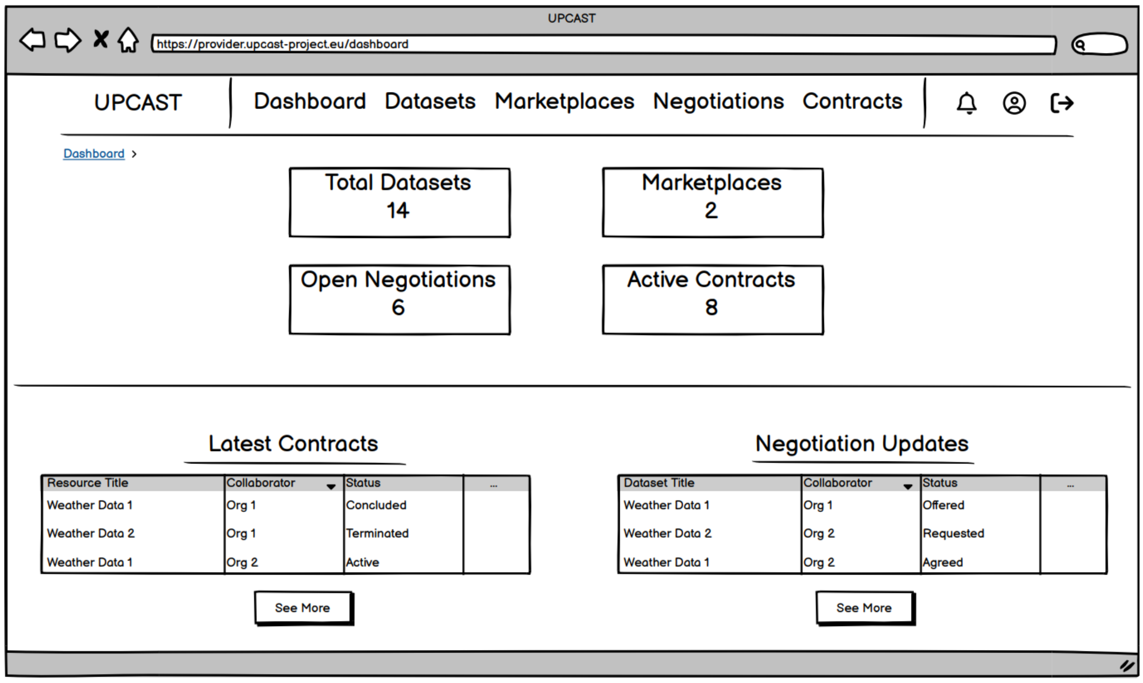
5.1 Resource Specification and Profiling

Functionality: The Resource Specification and Profiling allows the specification and profiling of dataset resources. It involves the tasks of preparing the dataset for publishing it to a marketplace. The preparation tasks involve the annotation the dataset with metadata, optionally environmental impact and pricing information, as well as policies for its usage and access. Resource profiling uses the functions of plugins that are presented in following sections.

Interface: The interface to the resource specification and profiling is a dashboard GUI that is also the entry point is the entry point of the provider to the functions of UPCAST. This section gives an overview of the initial designs of the provider's dashboard, which is used for the resource (dataset) specification and profiling. The dataset providers' dashboard supports the tasks of dataset annotation that are necessary before the dataset is published in a marketplace and thus, becomes available for discovery, negotiation, and eventually processing by potential consumers.

The resource specification and profiling dashboard is presented on the wireframes below, which intend to showcase its functionalities.

The fields that are shown in some wireframes, such as datasets, negotiations, organisations are tentative, and may be updated in the actual implementation to include additional information, which may be necessary for the successful integration of the dashboard with other UPCAST plugins.



Total Datasets
14

Marketplaces
2

Open Negotiations
6

Active Contracts
8

Latest Contracts

Resource Title	Collaborator	Status	...
Weather Data 1	Org 1	Concluded	
Weather Data 2	Org 1	Terminated	
Weather Data 1	Org 2	Active	

Negotiation Updates

Dataset Title	Collaborator	Status	...
Weather Data 1	Org 1	Offered	
Weather Data 2	Org 2	Requested	
Weather Data 1	Org 2	Agreed	

Figure 12: Provider Dashboard.

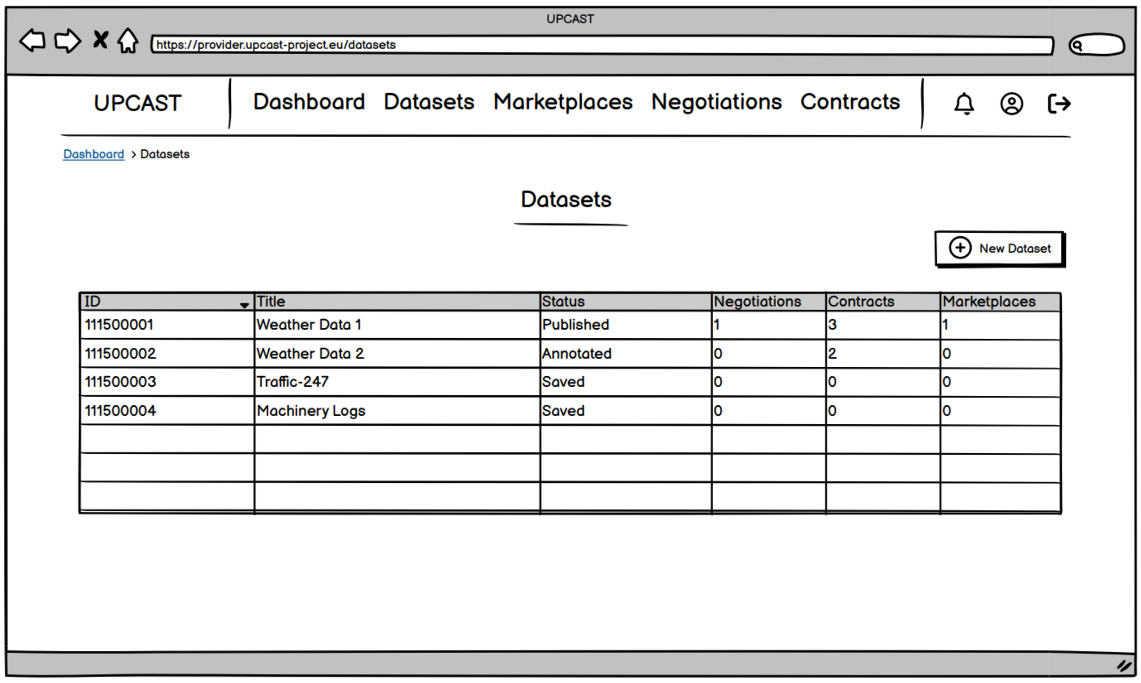
The provider's dashboard is depicted in Figure 12. The dashboard provides overview information of the most important elements that relate to the providers' datasets, marketplaces, and signed contracts while it offers shortcuts to frequently used functions. The dashboard may be reached after successful authentication of the provider. The dashboard includes the following elements:

1. Datasets: A list of all registered datasets by the provider. It also provides functions for creating a new dataset.
2. Marketplaces: An overview of all marketplaces the provider has collaborated for publishing their datasets and their settings. It providers functions to publish a dataset to a marketplace.
3. Negotiations: A list of all negotiations, with functions to filter by dataset, or other criteria.
4. Contracts: A list of all contracts, with functions to filter by dataset, marketplace or other criteria.

Moreover, the page provides functions (on the upper right part) for:

a) Notifications on important events, like a new negotiation request, or the arrival of monitoring events for an execution under an active contract, including alerts of possible contract violations.
b) Profile editing, with settings that relate to account security, or the user's organization, and so on.
c) Logout button.

Customization of the dashboard, like selection of themes, is an option.



ID	Title	Status	Negotiations	Contracts	Marketplaces
111500001	Weather Data 1	Published	1	3	1
111500002	Weather Data 2	Annotated	0	2	0
111500003	Traffic-247	Saved	0	0	0
111500004	Machinery Logs	Saved	0	0	0

Figure 13: Provider Dashboard, List of Datasets.

In the Datasets List screen (Figure 13) provides a sortable list of available datasets along with some important metadata. ID is a globally unique identifier of the dataset. The *Status* field indicates the status of the dataset. When the dataset is annotated with metadata and registered to the dashboard the status is set to "Saved". After annotating the dataset with its usage policies and profiles provided by other plugins, like pricing, and environmental parameters, the status is set to “Annotated”, which indicates that the dataset is ready to be published. Finally, if a dataset has been published in at least one marketplace, the Status is set to “Published”.

Other fields include useful statistics about the dataset, like the number of negotiations, or contracts assigned to it. The new dataset button guides the provider through the creation process for a new dataset.

A dataset must first be registered before it is published. Dataset registration includes the specification of its metadata as part of its profile.

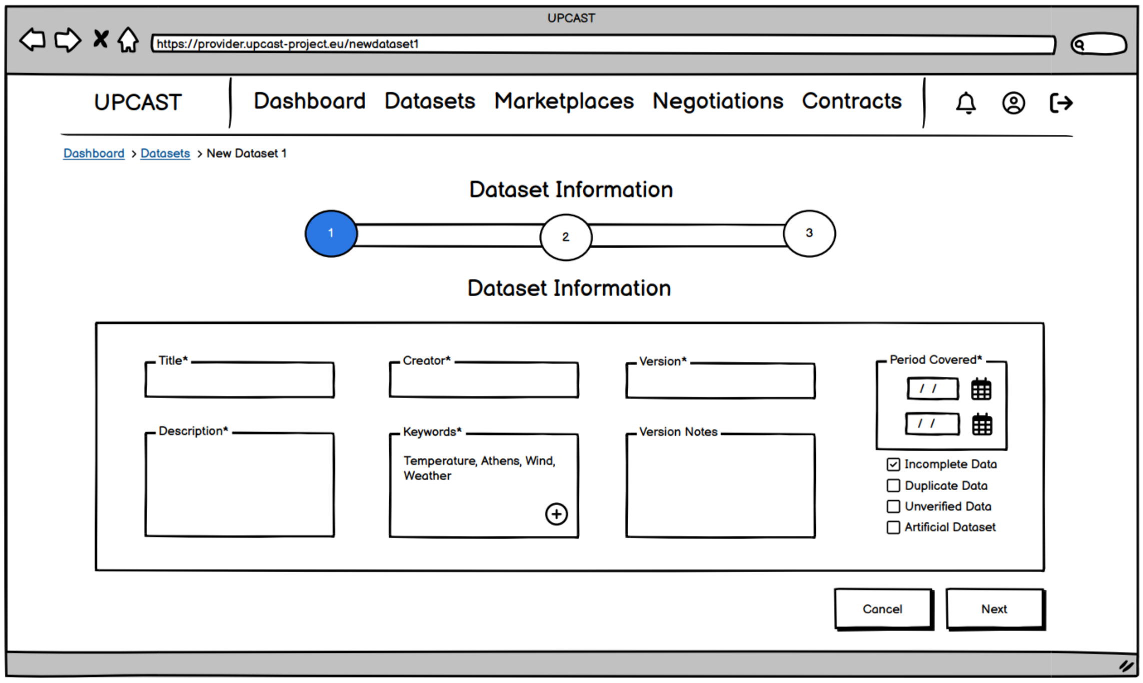
In accordance with the IDSA Data Model for the Dataspace Connector[9] dataset registration comprises three steps that reflect the corresponding layers implied in the data model:

1. Step 1: Dataset Information: Contains general information that describes the contents of the dataset, to cover the Resource layer or the IDSA model.
2. Step 2: Format Information: Contains information on the media type, the file format, or the encoding of the dataset implementation. Corresponds to the Representation layer of the IDSA model.
3. Step 3: File information: Contains details about the actual file(s) containing the dataset, and the contact point where the file is stored. Corresponds to the Artifact layer of the IDSA model.

Figure 14 shows a wireframe design for the first step, in which the overview information of the dataset, including its title, description, list of keywords, the timeframe of data collection and version information is provided. A set of flags is also included that provide the following information:

a) Whether the dataset has incomplete data, like empty fields, or potentially corrupt entries.
b) Whether the dataset has duplicate data, like entries referring to the same object, but collected more than once, either by multiple collection points, or due to errors.
c) Whether the dataset contains unverified data, like potential outliers, or measurement errors.
d) Whether the dataset is artificial, generated automatically through machine learning processes or by compiling pre-existing data.

[9] https://international-data-spaces-association.github.io/DataspaceConnector/Documentation/v6/DataModel



https://provider.upcast-project.eu/newdataset1

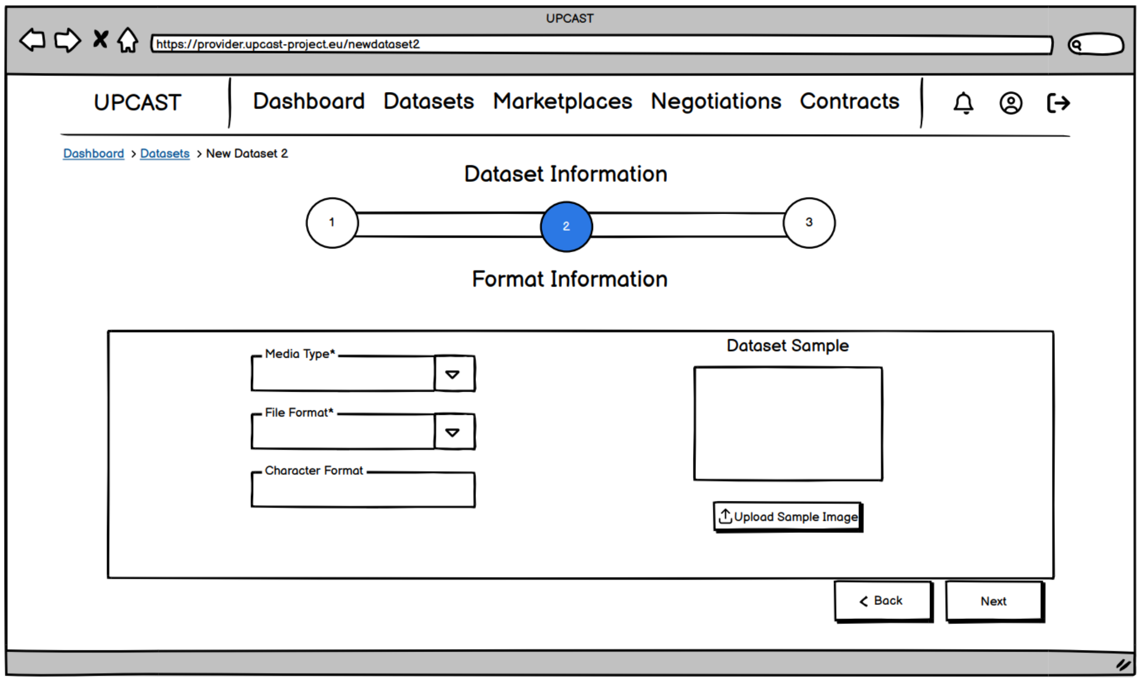
Dashboard > Datasets > New Dataset 1

Keywords*: Temperature, Athens, Wind, Weather

[✓ Incomplete Data] Duplicate Data Unverified Data Artificial Dataset

Figure 14: Provider Dashboard, Dataset Creation, Step 1.

The second step of the dataset registration shown in Figure 15 involves the specification of the media type (text, image, video, audio) and also the file format of the dataset. Depending on the dataset file format additional fields may be specified by the provider. For example, Figure 15 shows the character encoding field, which may be specified for a dataset of text format media type. Moreover, a sample of the dataset may also be uploaded.



https://provider.upcast-project.eu/newdataset2

Dashboard > Datasets > New Dataset 2

Figure 15: Provider Dashboard, Dataset Creation, Step 2.

In the third step of the registration process the actual files that implement the dataset are specified as shown in Figure 16. These include:

a) File metadata like name, byte size or compression format.
b) Hosting information like connection point and file identifiers.
c) Authentication and access control information.

Figure 16 shows some indicative fields. The actual data used in this step depend on the marketplace, or any other related dataset hosting platform.

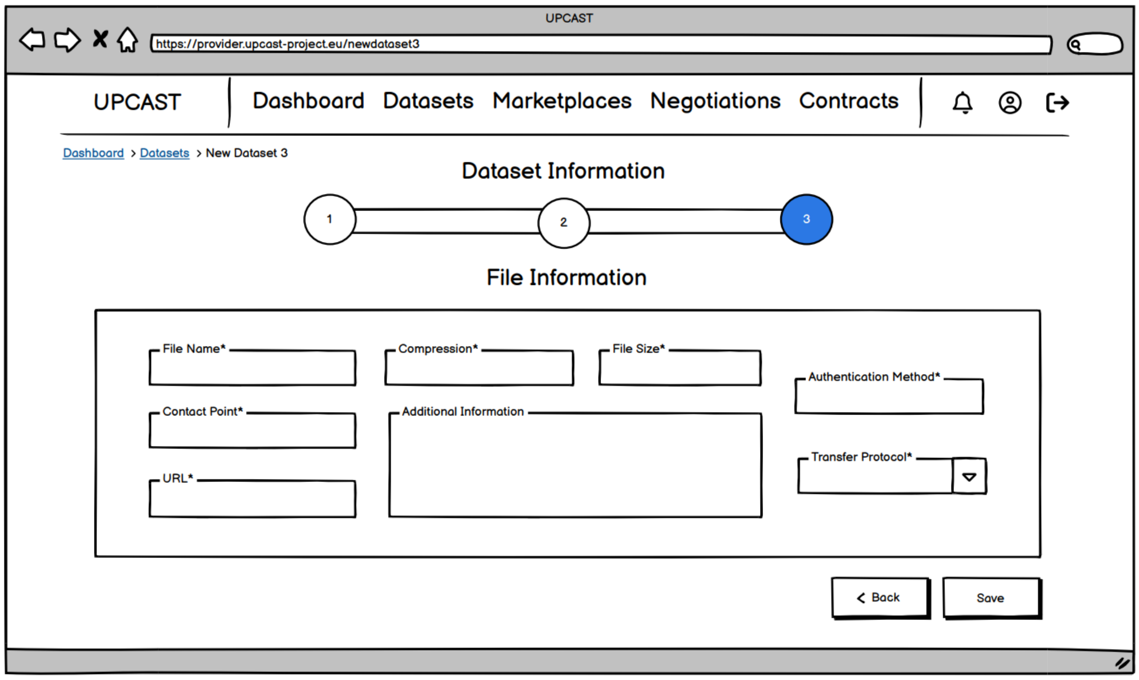
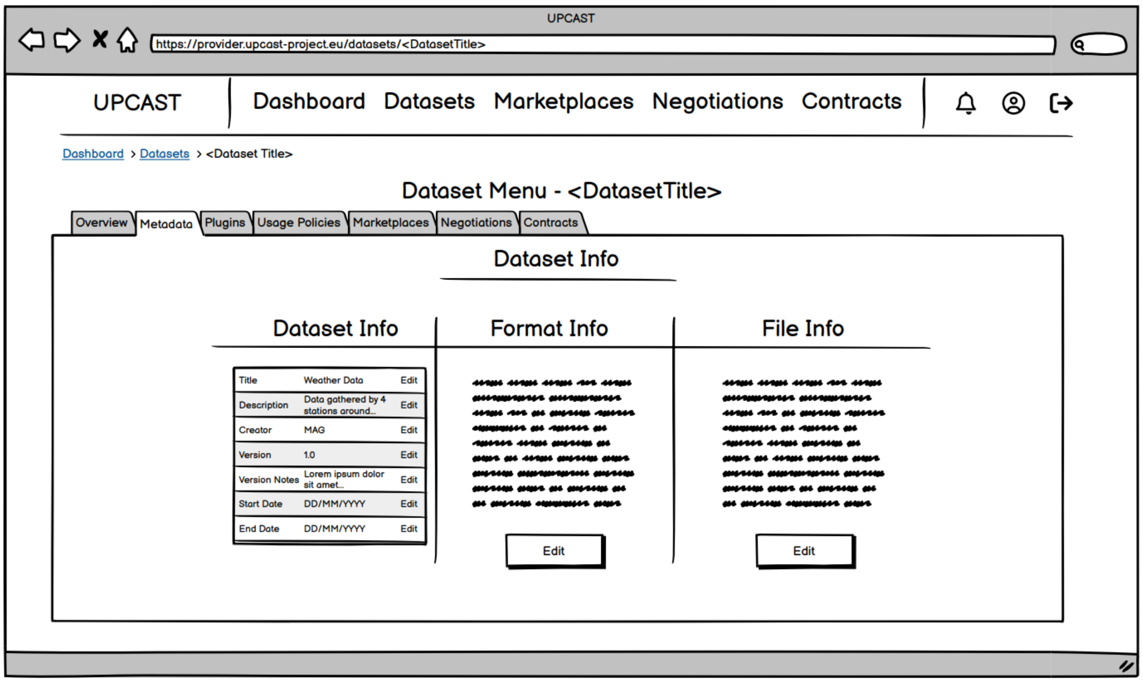


Figure 16: Provider Dashboard, Dataset Creation, Step 3.

After creating a dataset, the provider can access it through the list of datasets that is shown in Figure 17. The following tabs are included:

a) Overview, which will work as a mini dashboard specifically for the dataset showcasing some key points for it.
b) Metadata, containing the information provided by the user during the registration process. The information under this tab is separated in sections and shall be presented either in an array format with an edit option on each mutable field (shown on the left), or in a structured text format, with one single edit button in the bottom to open up a pop-up menu with edit options (shown in the middle and right).
c) Plugins will provide access to the external plugin services used for the annotation of the dataset.
d) Usage Policies, which will provide the user with an interface to pick actions and assign rules to them, or even define new domain specific ones to augment the available vocabulary. There, in accordance with the ODRL model, providers will set the initial rules, which will be used as a starting point in the negotiation process.
e) Marketplaces, where providers will be able to publish datasets to a list of available marketplaces.
f) Negotiations, where providers will access and manage negotiations for the dataset.
g) Contracts, where providers will access and manage contracts for the dataset. This section could also be an entry point for the monitoring interface during execution time.

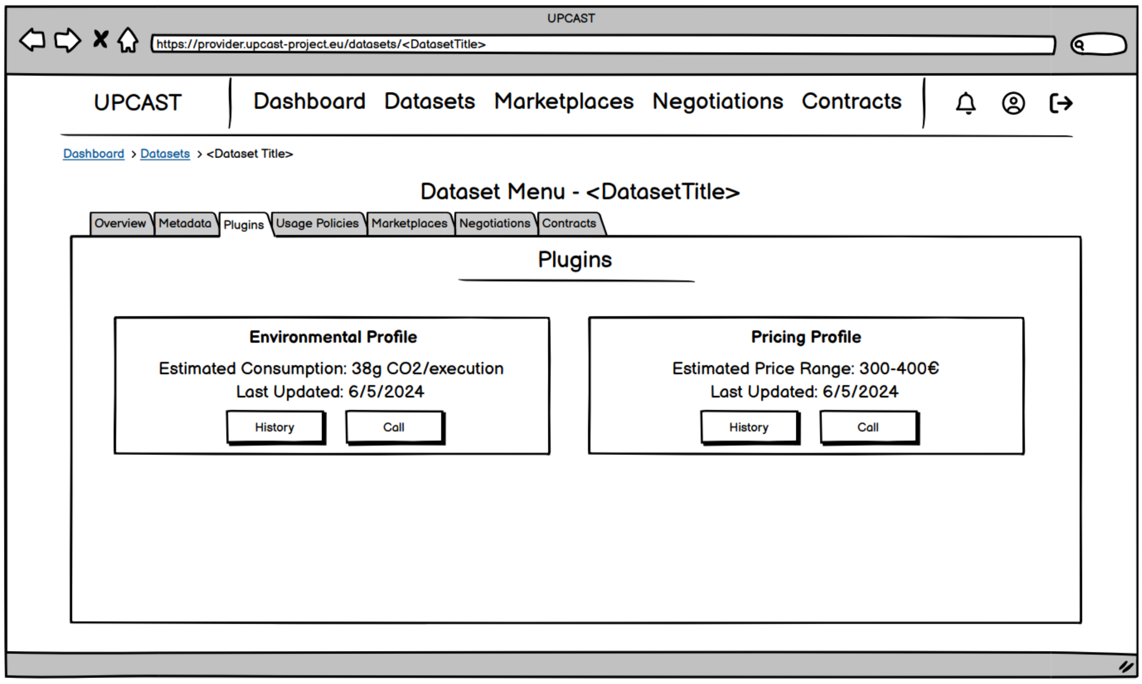


Title	Weather Data	Edit
Description	Data gathered by 4 stations around...	Edit
Creator	MAG	Edit
Version	1.0	Edit
Version Notes	Lorem ipsum dolor sit amet...	Edit
Start Date	DD/MM/YYYY	Edit
End Date	DD/MM/YYYY	Edit

Figure 17: Provider Dashboard, Dataset menu.

The majority of the dashboard tabs that remain are highly dependent on the data and interaction with other services and will be customized during dashboard integration with the data market sharing platforms. Nevertheless, some platform independent tabs are presented in the following.

Figure 18 shows a screen for the integration with external plugins to further annotate the dataset. Each plugin has two buttons. The first opens up a history of plugin invocations and returns for this dataset, while the second activates the invocation to the plugin. Input data to the plugin are provided through a popup window.

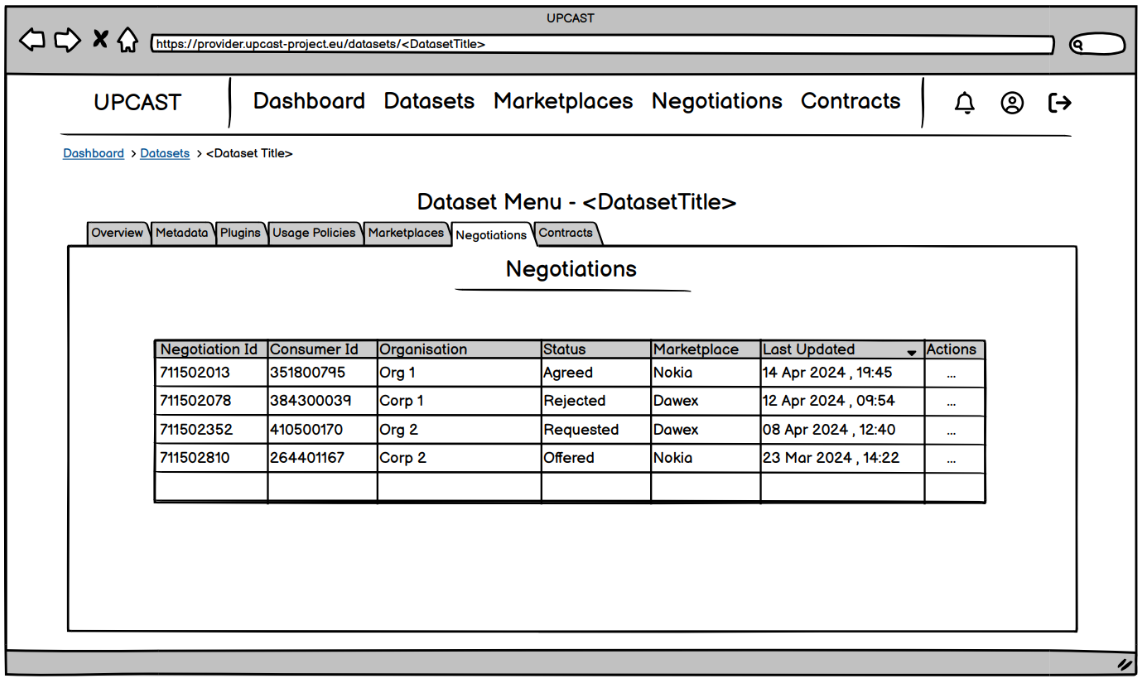


Environmental Profile
Estimated Consumption: 38g CO2/execution
Last Updated: 6/5/2024

Pricing Profile
Estimated Price Range: 300-400€
Last Updated: 6/5/2024

Figure 18: Provider Dashboard, Dataset annotation plugins.

Figure 19 and Figure 20 show the negotiations for a particular dataset and all datasets of a provider.


UPCAST

https://provider.upcast-project.eu/datasets/<DatasetTitle>

UPCAST | Dashboard Datasets Marketplaces Negotiations Contracts

Dashboard > Datasets > <Dataset Title>

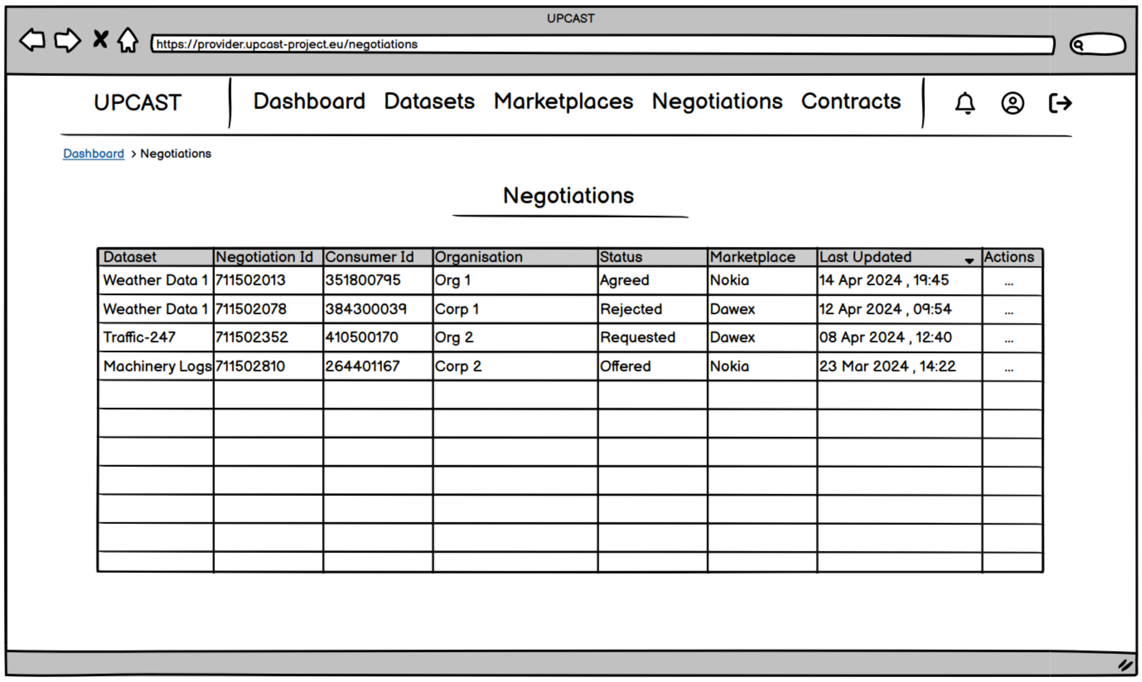
Dataset Menu - <DatasetTitle>

Overview | Metadata | Plugins | Usage Policies | Marketplaces | Negotiations | Contracts

Negotiations

Negotiation Id	Consumer Id	Organisation	Status	Marketplace	Last Updated	Actions
711502013	351800795	Org 1	Agreed	Nokia	14 Apr 2024 , 19:45	...
711502078	384300039	Corp 1	Rejected	Dawex	12 Apr 2024 , 09:54	...
711502352	410500170	Org 2	Requested	Dawex	08 Apr 2024 , 12:40	...
711502810	264401167	Corp 2	Offered	Nokia	23 Mar 2024 , 14:22	...

Figure 19: Provider Dashboard, List of dataset negotiations.


UPCAST

https://provider.upcast-project.eu/negotiations

UPCAST | Dashboard Datasets Marketplaces Negotiations Contracts

Dashboard > Negotiations

Negotiations

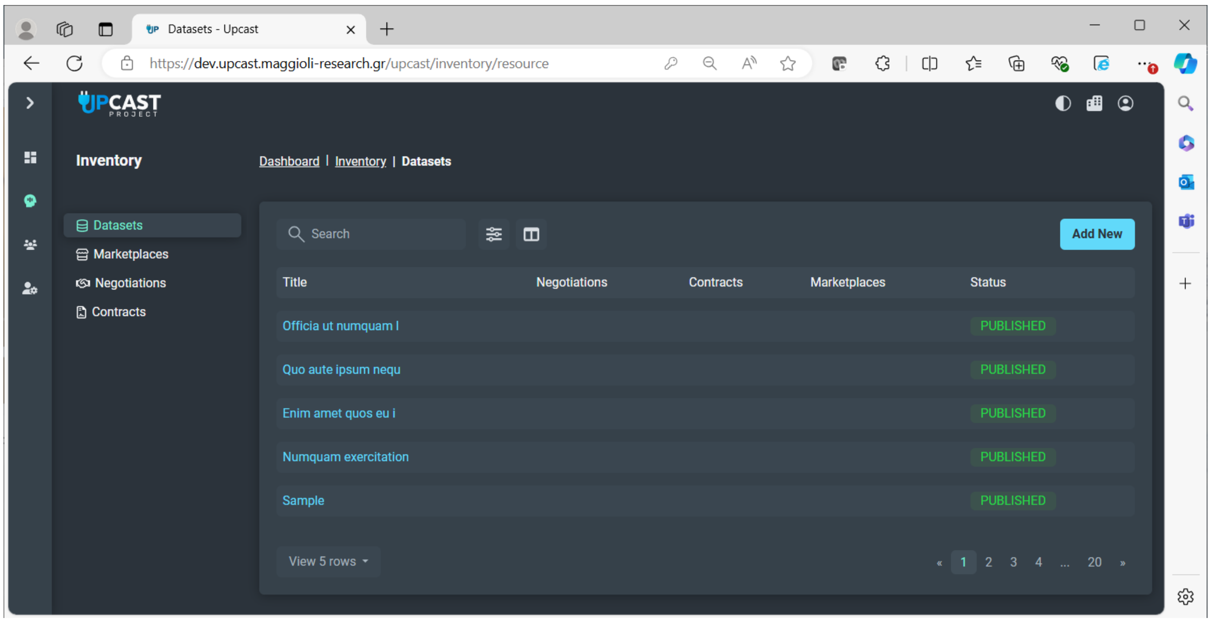
Dataset	Negotiation Id	Consumer Id	Organisation	Status	Marketplace	Last Updated	Actions
Weather Data 1	711502013	351800795	Org 1	Agreed	Nokia	14 Apr 2024 , 19:45	...
Weather Data 1	711502078	384300039	Corp 1	Rejected	Dawex	12 Apr 2024 , 09:54	...
Traffic-247	711502352	410500170	Org 2	Requested	Dawex	08 Apr 2024 , 12:40	...
Machinery Logs	711502810	264401167	Corp 2	Offered	Nokia	23 Mar 2024 , 14:22	...

Figure 20: Provider Dashboard, Negotiations menu.

Both lists contain information on the consumer, which shall be fetched from the negotiation plugin, or the marketplace where the negotiation takes place.

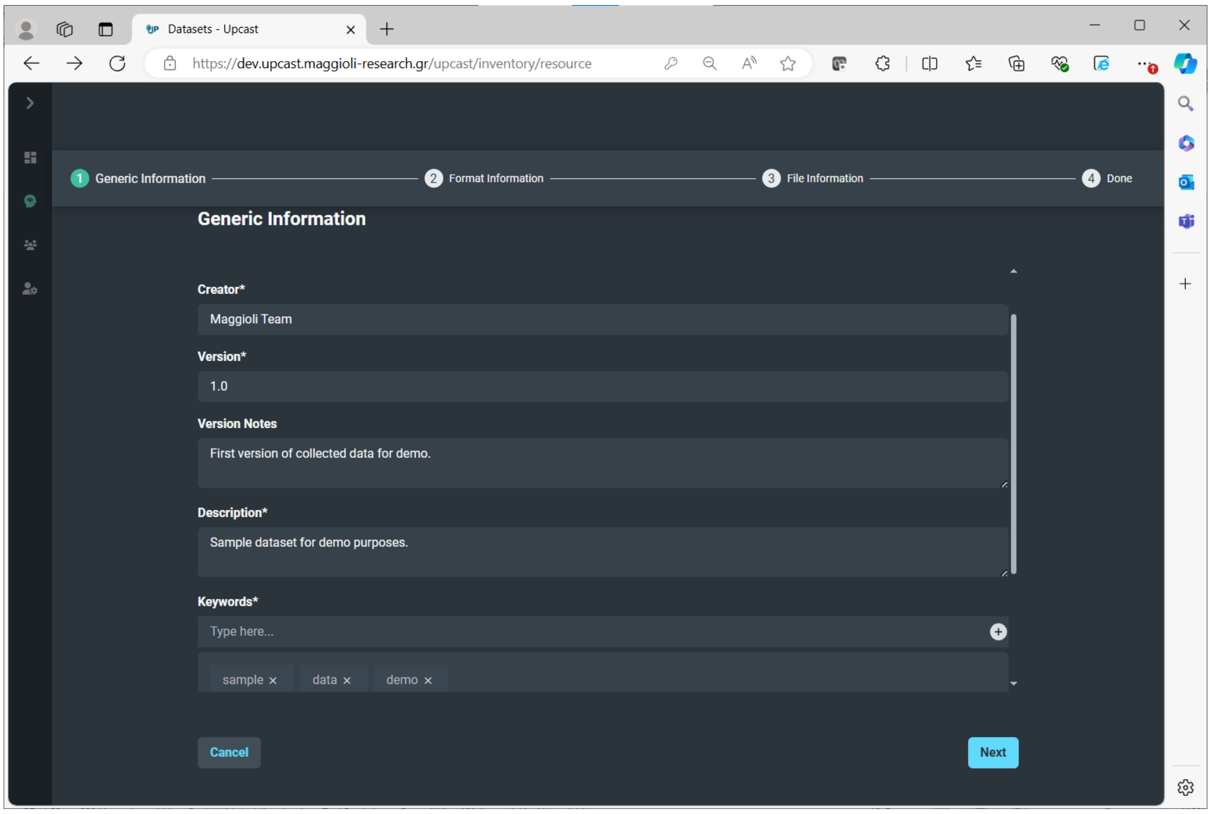
When a negotiation Is finalized, its status is set to “Agreed” or “Rejected” depending on the outcome. During the process, when the last proposal comes from the provider, the status is set to “Offered”, while when it comes from the side of a consumer, the status is set to “Requested”. The terms are based on the negotiation states shown on the IDS negotiation state machine (section 5.11).

Figure 21 is a snapshot of the provider's dashboard that shows the list of the provider's datasets, while Figure 22 is a snapshots of the provider's dashboard that depict the first steps for the creation of a new dataset, and Figure 23 shows the updated list of datasets, which includes the one that has been created.



Title	Negotiations	Contracts	Marketplaces	Status
Officia ut numquam l				PUBLISHED
Quo aute ipsum nequ				PUBLISHED
Enim amet quos eu i				PUBLISHED
Numquam exercitation				PUBLISHED
Sample				PUBLISHED

Figure 21: Provider Dashboard, List of datasets.



Generic Information

Creator*
Maggioli Team

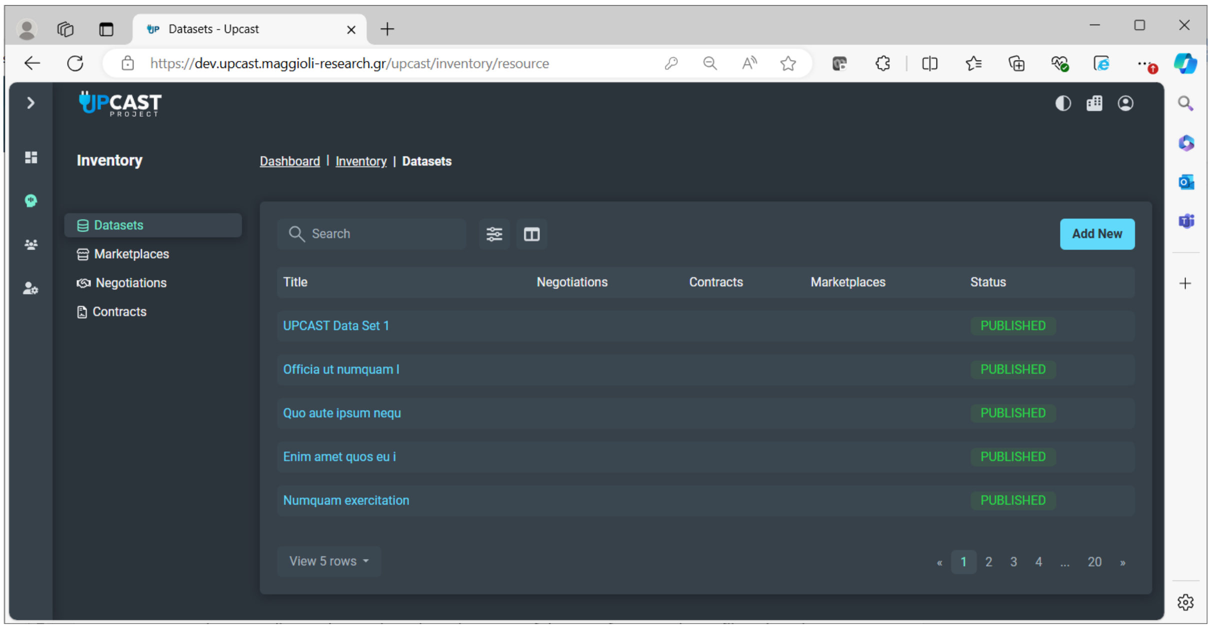
Version*
1.0

Version Notes
First version of collected data for demo.

Description*
Sample dataset for demo purposes.

Keywords*
sample × data × demo ×

Figure 22: Provider Dashboard, New Dataset, Step 1.



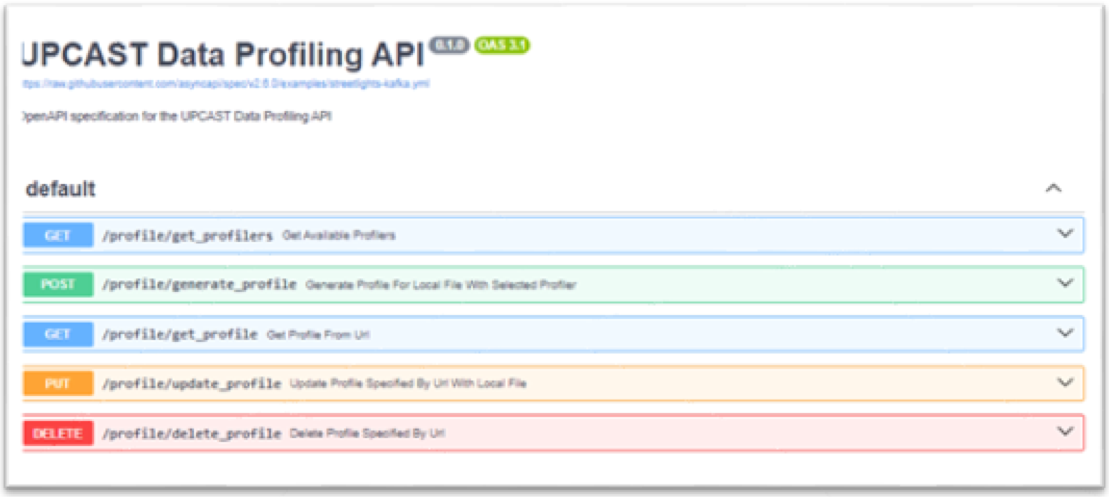
Title	Negotiations	Contracts	Marketplaces	Status
UPCAST Data Set 1				PUBLISHED
Officia ut numquam l				PUBLISHED
Quo aute ipsum nequ				PUBLISHED
Enim amet quos eu i				PUBLISHED
Numquam exercitation				PUBLISHED

Figure 23: Provider dashboard, Updated list of datasets.

5.2 Semantic Profiling

Functionality: The Semantic Profiling plugin is used to enrich the annotation of a dataset based on semantic information that can be deduced from references to the dataset, associations with other similar datasets and so on. The semantic profiler to be used is configurable, which allows for fine tuning of the profiling task.

Interface: Figure 24 shows an excerpt of the OpenAPI specification of the Data Profiling plugin. The full OpenAPI specification for the profiling API is available at https://github.com/EU-UPCAST/OpenAPISpecification/tree/main/profiling.


UPCAST Data Profiling API

OpenAPI specification for the UPCAST Data Profiling API

default

GET /profile/get_profilers Get Available Profilers

POST /profile/generate_profile Generate Profile For Local File With Selected Profiler

GET /profile/get_profile Get Profile From Url

PUT /profile/update_profile Update Profile Specified By Url With Local File

DELETE /profile/delete_profile Delete Profile Specified By Url

Figure 24: Data Profiling API.

Data Model: Figure 25 shows the data model of the semantic profiling. A dataset can have multiple profiles generated using different profilers.

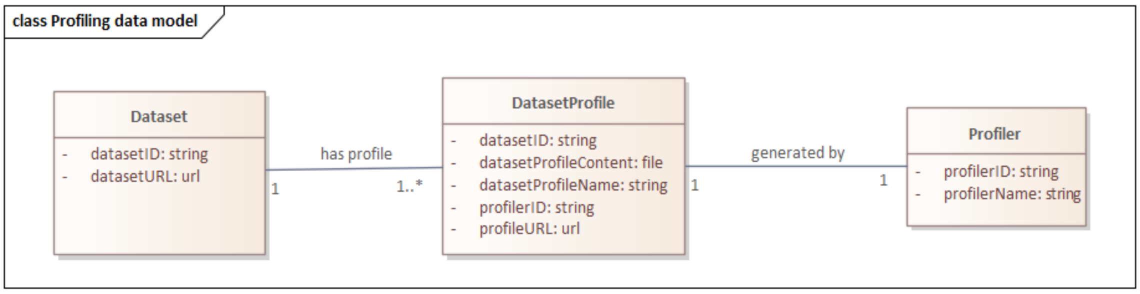


Figure 25: Data Model of Data Profiling.

Typical usages: Figure 26 shows the typical interactions for using the profiling service, which include (a) Connect and configure profilers; (b) get a list of available profilers; (c) generate a profile for a dataset using a selected profiler; and (d) get a profile for a specific dataset.

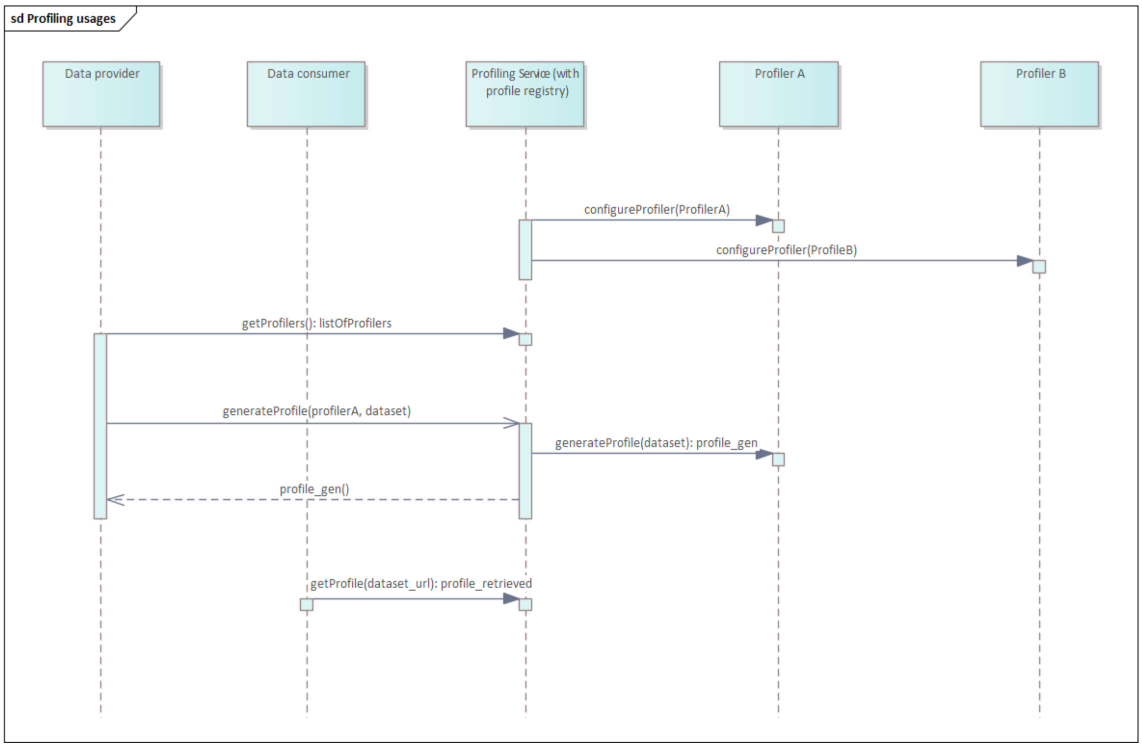

sd Profiling usages
Data provider
Data consumer
Profiling Service (with profile registry)
Profiler A
Profiler B
configureProfiler(ProfilerA)
configureProfiler(ProfileB)
getProfilers(): listOfProfilers
generateProfile(profilerA, dataset)
generateProfile(dataset): profile_gen
profile_gen()
getProfile(dataset_url): profile_retrieved

Figure 26: Sequence Diagram of Data Profiling plugin.

5.3 Resource Publishing

Functionality: The Resource Publishing plugin is a set of services designed to support the publishing of resources to ensure they are discoverable, i.e., the management of a resource catalog. Similar to the Discovery plugin, the Resource Publishing plugin includes a web service API for integration and is configured using a Docker Compose file. The following services are included: API Service, SOLR Engine, PostgreSQL, and a CKAN Backend Service. The plugin services can be executed using Docker commands, which are detailed in the plugin repository.

Interface: The Resource Publishing plugin API provides multiple endpoints to manage the lifecycle of resources, including adding, updating, and deleting resources, as well as making them discoverable through indexing. Here, we outline the technical steps to utilize these API endpoints with a sample UI.

There are two main functionalities supported by the Resource Publishing plugin: (1) publishing datasets and (2) managing resource metadata. Publishing datasets involves indexing the datasets to make them searchable, while managing resource metadata allows for updating and maintaining the discoverability of resources.

The OpenAPI specifications of the Resource Publishing plugin are at https://github.com/EU-UPCAST/OpenAPISpecification/blob/main/discovery_plugin/publish/openapi.yml

5.4 Resource Discovery

Functionality: The Discovery plugin is a set of services, including a web service API for integration. The services are configured in a Docker Compose file, which contains the following services: API Service, SOLR Engine, PostgreSQL, and a CKAN Backend Service. The Discovery plugin services can be run using Docker commands, which are listed in the plugin repository.

Interface: The Discovery plugin API consists of multiple endpoints that manage the complete lifecycle of datasets, including adding, updating, deleting, and searching datasets within the index, as well as discovering similar resources (see ANNEX II). Below, we describe the technical steps to run these API endpoints with a sample UI.

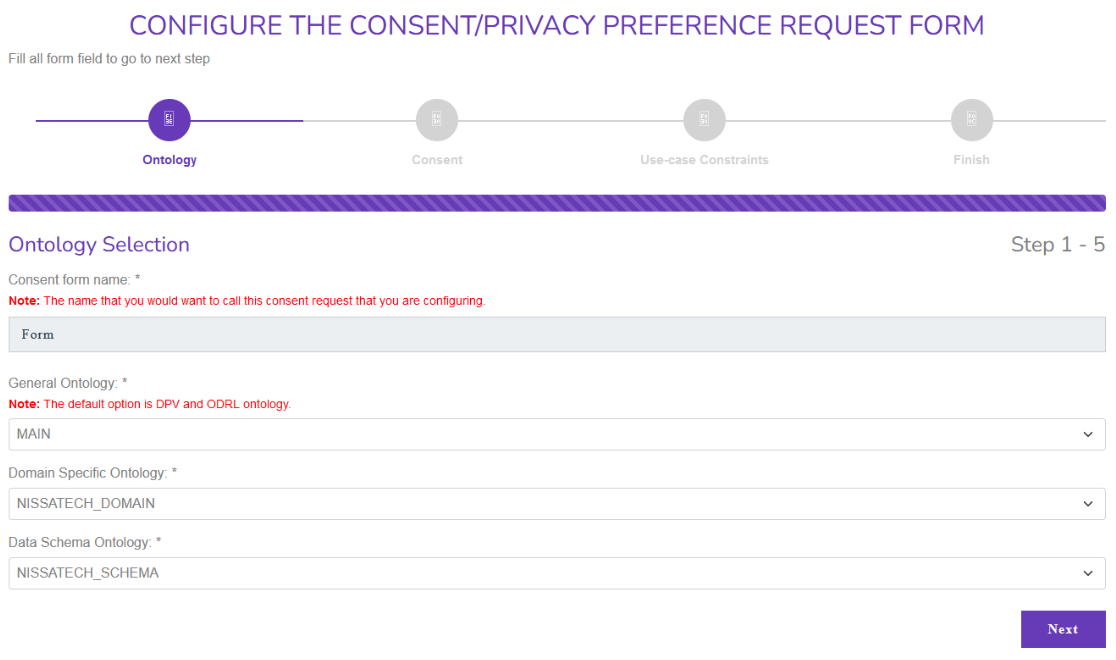
The plugin supports two types of discovery: (1) dataset search and (2) resource search. Dataset search can include multiple parameters for making a Solr query, filtering, sorting, and producing faceted results. To make a dataset searchable, it needs to be added to the index.

The OpenAPI Interface specifications of the Resource Discovery plugin are at https://github.com/EU-UPCAST/OpenAPISpecification/blob/main/discovery_plugin/openapi3_1.yaml

5.5 Privacy and Usage Policy Plugin

Functionality: The privacy and usage policy plugin allows data providers to define access and usage policies over a resource using terms from a default or custom ontology. If users are registered, users can save created policies, upload ontologies to expand the list of terms that can be used in policies. In addition, for the special case of data providers being private citizens, a data consumer can create, configure and send request forms to data providers requesting consent for usage of their data that may contain personal data. On the data provider side, they can check if there are any requests forms sent to them, and then opt in or out of these requests.

Interface: The GUI of the Privacy and Usage Policy plugin are shown in Figure 27, Figure 28, Figure 29.



CONFIGURE THE CONSENT/PRIVACY PREFERENCE REQUEST FORM

Fill all form field to go to next step

Ontology Selection

Step 1 - 5

Consent form name: *

Note: The name that you would want to call this consent request that you are configuring.

Form

General Ontology: *

Note: The default option is DPV and ODRL ontology.

MAIN

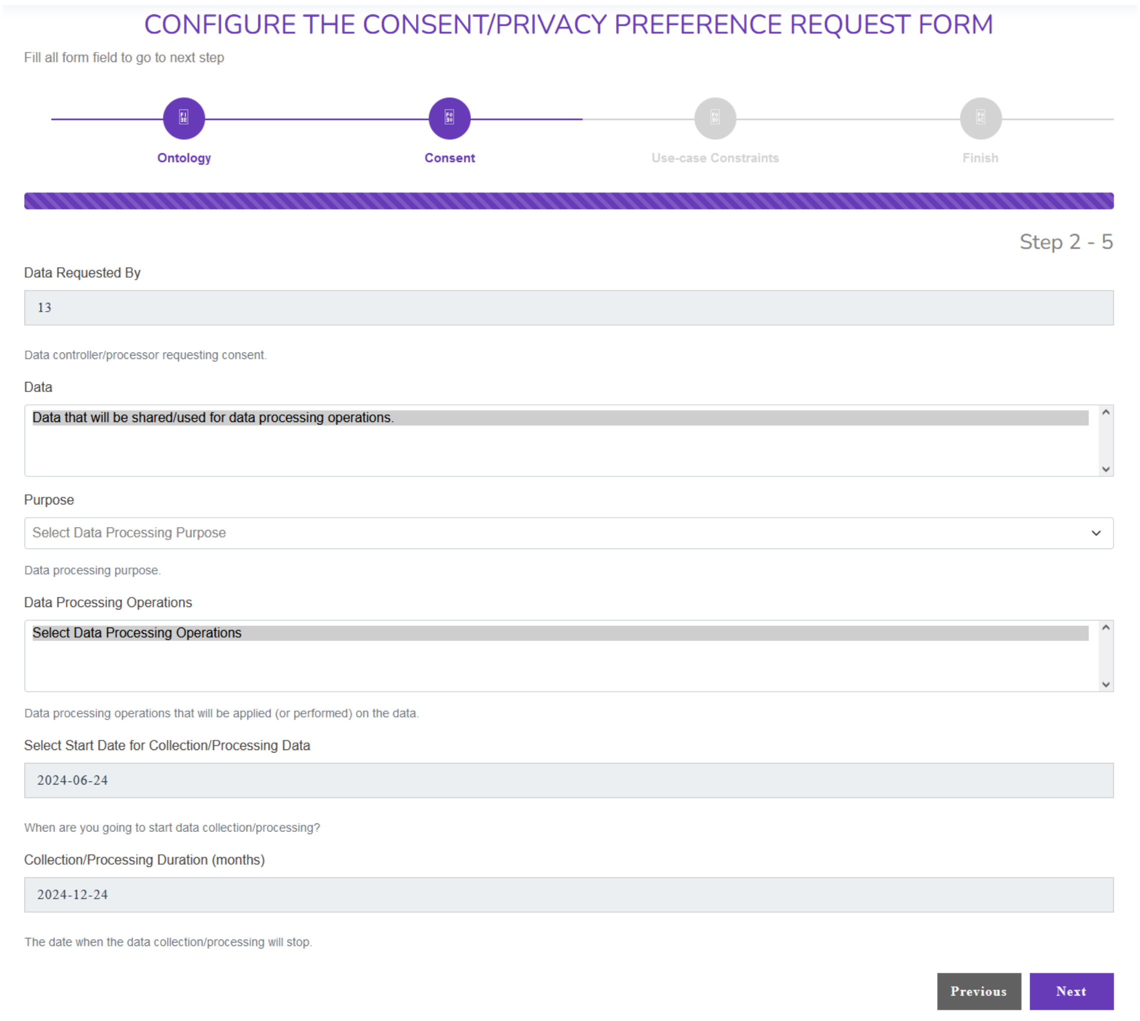
Domain Specific Ontology: *

NISSATECH_DOMAIN

Data Schema Ontology: *

NISSATECH_SCHEMA

Figure 27: Privacy and Usage Policy GUI, snapshot 1.



CONFIGURE THE CONSENT/PRIVACY PREFERENCE REQUEST FORM

Fill all form field to go to next step

Step 2 - 5

Data Requested By

13

Data controller/processor requesting consent.

Data

Data that will be shared/used for data processing operations.

Purpose

Select Data Processing Purpose

Data processing purpose.

Data Processing Operations

Select Data Processing Operations

Data processing operations that will be applied (or performed) on the data.

Select Start Date for Collection/Processing Data

2024-06-24

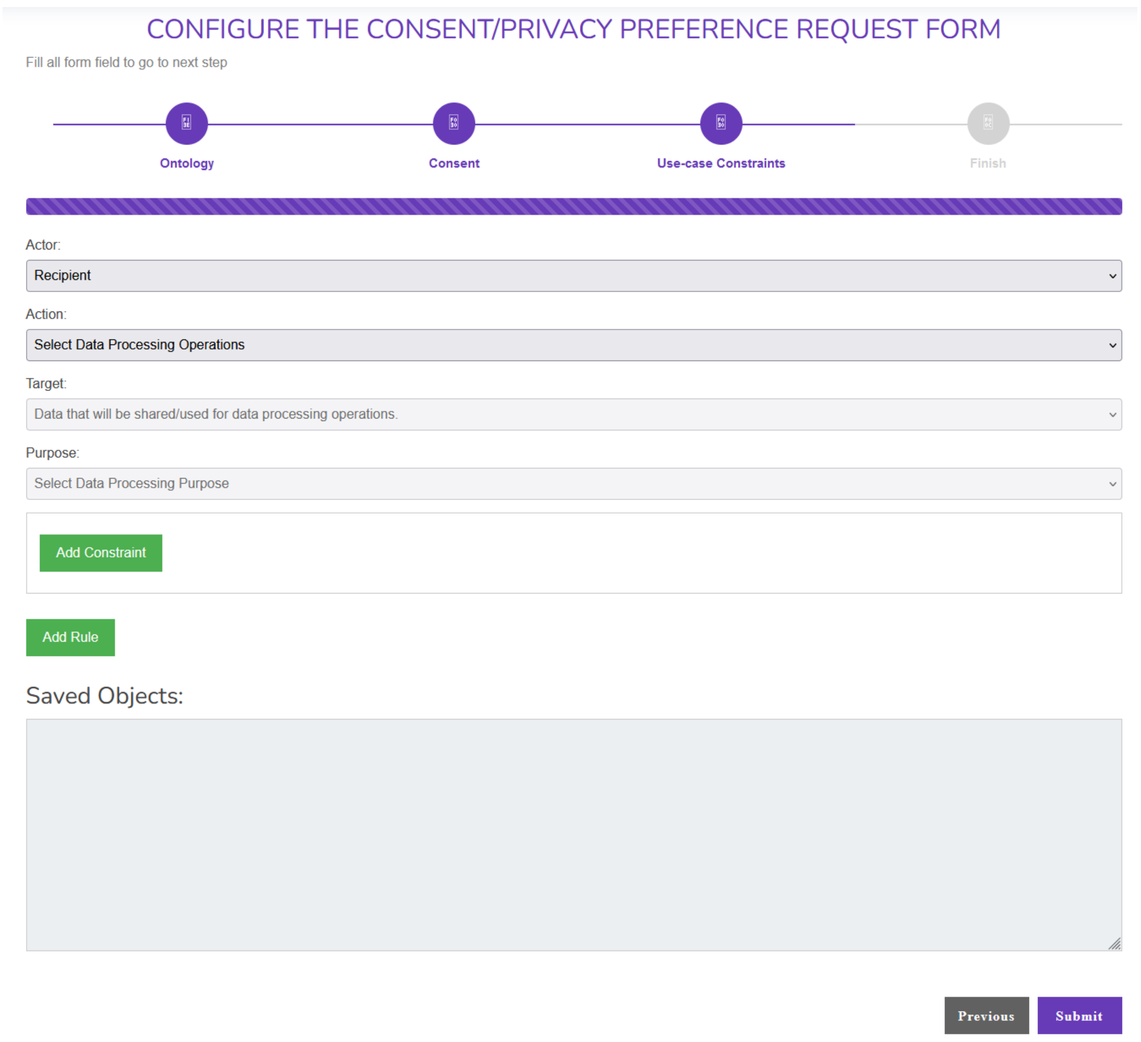
When are you going to start data collection/processing?

Collection/Processing Duration (months)

2024-12-24

The date when the data collection/processing will stop.

Figure 28: Privacy and Usage Policy GUI, snapshot 2.


CONFIGURE THE CONSENT/PRIVACY PREFERENCE REQUEST FORM

Fill all form field to go to next step

Actor: Recipient

Action: Select Data Processing Operations

Target: Data that will be shared/used for data processing operations.

Purpose: Select Data Processing Purpose

Saved Objects:

Figure 29: Privacy and Usage Policy GUI, snapshot 3.

Data Model: The Open Digital Rights Language (ODRL) is used to model policies. ODRL is a policy expression language that provides a flexible and interoperable information model, vocabulary, and encoding mechanism for representing statements about the usage of content and services. The following are several elements that may appear in an ODRL policy:

An **Asset** is a resource or a collection of resources that are the subject of a policy. It can be any form of identifiable (by an IRI) resource, such as data/information, content/media, applications, services, or physical artefacts. On the other hand, an **Asset Collection** is an Asset that represents a set of resources as a single resource. It is used to indicate that all members of the set will be the subject of the policy.

A **Party** is an entity or a collection of entities that undertake functional roles in a policy, such as a person, collection of people, organisation, or agent. An agent is a person or thing that takes an active role or produces a specified effect. Like Assets, they are usually identified by an IRI.

A Party Collection is a Party that is a single entity representing a set of member entities. This indicates that all the members of the set will undertake the same functional role in the Rule. A Party Collection may have any number of refinement property values of type Constraint.

An ODRL policy may be one of three different types: an offer if the policy is defined by a data provider, a request if it is issued by a data consumer, and an agreement if it is the result of a successful negotiation. Formally, an ODRL policy may be defined as a set of rules $\mathfrak{R} = \{\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_n\}$ that combine to specify permissions, restrictions, or obligations on a variety of legal entities.

Formally, a **Rule** is a 5-tuple $\mathcal{R} = (\mathcal{A}, DP, DC, \mathcal{P}, \mathcal{T})$, where $\mathcal{A}$ is the real-world action (of type Action), often a data process, to which this rule applies to; DP and DC are the assigner and assignee of the rule, respectively (both of type Party); $\mathcal{P}$ denotes the purpose of the action; and $\mathcal{T}$ is the Asset, or in our model, the (sub-)dataset that is the subject of the rule. Elements that are common to every rule in a policy can be expressed at the level of the policy. A Rule may be one of three, disjoint types that determine the semantic meaning of the rule: a **Permission**, a **Prohibition**, or an **Obligation**.

In addition, ODRL allows for the definition of simple logical expressions called **Constraints**. **Constraints** are boolean/logical expressions that can be used to refine the semantics of an Action and Party/Asset Collection or declare the conditions applicable to a Rule. Constraints can be represented as a Constraint or Logical Constraint. A Logical Constraint will refer to existing Constraints as its operands. When multiple Constraints apply to the same Rule, Action, Party/Asset Collection, then they are interpreted as conjunction that must all be satisfied.

A constraint contains the following elements: a **left operand** (of type odrl:LeftOperand), a relational **operator,** and a **right operand** that can be a literal value, an IRI or, if the operator is set-based, a set of the former. If the comparison returns a match the Constraint is **satisfied**, otherwise it is **not satisfied**. Some constraints

In our model, a constraint is a 3-tuple $c = (\lambda, op, \rho)$ where the left operand λ is a class property, the operator is a binary operator op (as defined in the ODRL specification), and the right operand ρ is a literal value with a datatype or (if op is odrl:isA) a class name. We assume that an ontology has been provided that defines class hierarchies and class properties. The operator *op* can be any of the following binary operators: odrl:equals (=), odrl:gt (>), odrl:gteq (≥), odrl:lt (<), odrl:lteq (≤), odrl:neq (≠), odrl:isA (or rdf:type), odrl:hasPart (⊃), odrl:isPartOf (⊂), odrl:isAllOf (≡), odrl:isAnyOf (∈), and odrl:isNoneOf (∉). The semantic interpretation of a constraint is that the values of the class property λ are compared to the value ρ by the binary operator, which may be true or false. We proceed to describe the formulation of ODRL constraints as logical formulae. Let $id(e)$ be a function that outputs a variable unique to an ODRL element e, and $parent(e)$ be a function that returns $id(e')$ if e is an element at the level of e' (e.g., for a rule $\mathcal{R}$, $parent(\mathcal{A})$ is $id(\mathcal{R})$), we may express c as the following logic formula:

$$f(c) = \begin{cases} \lambda(parent(c), id(c)) \wedge \rho(id(c)) \; if \; op = isA \\ \lambda(parent(c), id(c)) \wedge (id(c) \; op \; \rho) \; otherwise \end{cases}$$

A refinement $R = \{c_1, c_2, \dots, c_n\}$ is a set of constraints, whose purpose is to filter the instances of a class or members of a dataset to those for which every constraint in the refinement holds true. As a logical formula, it can be expressed as the conjunction of all constraints that comprise it:

$$f(R) = \bigwedge_{c_i \in \mathcal{R}} f(c_i)$$

A **Logical Constraint** is used for expressions which compare two or more operands which are existing Constraints by one logical operator. If the comparison returns a logical match, then the Logical Constraint is **satisfied**, otherwise it is **not satisfied**. For example, three Constraints could be logically **and**-ed indicating that all three must be true for the Logical Constraint to be satisfied. A Logical Constraint must have one **operand** indicating the logical relationship of the compared existing constraints; the operand must be *or*, *xone*, *and andSequence*.

Constraints may be used to define usage policies; some Instances of Usage Policy Patterns are as follows:

- Allow the Usage of the Data (provides data usage without any restrictions).
- Interval-restricted Data Usage (provides data usage within a specified time interval).
- Duration-restricted Data Usage (allows data usage for a specified period starting from access).
- Location Restricted Policy.
- Perpetual Data Sale (Payment once).
- Data Rental (Payment frequently).
- Role-restricted Data Usage.
- Purpose-restricted Data Usage Policy.
- Restricted Number of Usages (allow data usage for n times).
- Security Level Restricted Policy (allow data access with a specified security level).
- Use Data and Delete it After (allows data usage within a specified time interval with the restriction to delete it at a specified time stamp).
- Attach Policy when Distribute to a Third-party.
- Distribute only if encrypted.

In our model, a logical constraint is a pair $b = (R, op)$ where R is a set of constraints and *op* is a Boolean operator that determines the truth value of *b* depending on the values in R. In particular, a refinement R is semantically equivalent to a logical constraint $b = (R, op)$ where op is the logical conjunction (odrl:And).

An **Action** indicates an operation that can be exercised on a target Asset (data source). It may have any number of refinements that refine the semantics of the operation. The ODRL Information Model defines two top-level Actions:

- use - actions that involve general usage by parties.
- transfer - actions that involve in the transfer of ownership to third parties.

In our model, the action of a rule is a pair $\mathcal{A} = (A, R)$ where *A* is either a class name (sub-class of dpv:Action) or an IRI referencing an instance of a dpv:Action or odrl:Action, as defined in the relevant ontology (default is ODRL and DPV), and R is its refinement. If *A* is a class name *C* and $c \in R$ is a constraint $c = (classProperty, op, value)$, then $\mathcal{A}$ encompasses actions of type C whose property *classProperty* when compared to value by the operator op is true. When translated into ODRL (in JSON-LD syntax), it should look as follows:

Where A, L_i, op_i, R_i are translated into their respective string representations. Finally, we may express the action as the following logical formula:

$$f(\mathcal{A}) = hasAction(parent(\mathcal{A}), id(\mathcal{A})) \wedge A(id(\mathcal{A})) \wedge f(\mathcal{R})$$

The assigner of a rule *DP* is an IRI that references an instance of an odrl:Party. On the other hand, the assignee of a rule is a pair $DC = (N, R)$ where *N* is either a class name or an IRI referencing an instance of an odrl:Party, and R is its refinement. If *N* is an odrl:Party and R is a refinement over some of its members, then the assignee is an odrl:PartyCollection. Then, expressed as a logic formula, we get the following:

$$f(DP) = hasAssigner(parent(DP), DP)$$

$$f(DC) = hasAssignee(parent(DC), id(DC)) \wedge N(id(DC)) \wedge f(R)$$

In standard ODRL, the purpose that is subject of a rule can only be expressed as a constraint, using the left operand odrl:purpose. Following the analysis of UPCAST requirements, we introduce the following innovation:, A purpose can be assigned directly as an element of a rule, and is formalised as a pair $\mathcal{P} = (P, R)$ where *P* is a class name (sub-class of dpv:Purpose), and R is its refinement. It is expressed as a logic formula as follows:

$$f(\mathcal{P}) = hasPurpose(parent(\mathcal{P}), id(\mathcal{P})) \land P(id(\mathcal{P})) \land f(R)$$

The target of a rule is a pair $\mathcal{T} = (D, R)$ where D is either a class name or an IRI referencing a dataset or instance of odrl:Asset, and R is its refinement. If D is an odrl:Asset and R is a refinement over some of its members, then the target is an odrl:AssetCollection. Of special note is a constraint $c \in R$ of the form $c = (upcast{:}query, odrl{:}eq, Q)$ where Q is a query in a standard query language syntax (e.g., SQL or SPARQL). In this case, the target is $Q(D)$ or the result of the execution of Q over D. As a logical formula:

$$f(\mathcal{T}) = hasTarget\big(parent(\mathcal{T}), id(\mathcal{T})\big) \land D\big(id(\mathcal{T})\big) \land f(R)$$

If the target has a constraint that specifies a query with head $Q(\vec{x})$ and body $Q_1(\overrightarrow{x_1}) \land \ldots \land Q_n(\overrightarrow{x_n})$, then:

$$f(\mathcal{T}) = hasTarget(parent(\mathcal{T}), \vec{x}) \land\ Q_1(\overrightarrow{x_1}) \land \ldots \land Q_n(\overrightarrow{x_n}) \land f(R)$$

The logical formula for a rule $\mathcal{R}$

$$f(\mathcal{R}) = f(\mathcal{A}) \land f(DP) \land f(DC) \land f(\mathcal{P}) \land f(\mathcal{T})$$

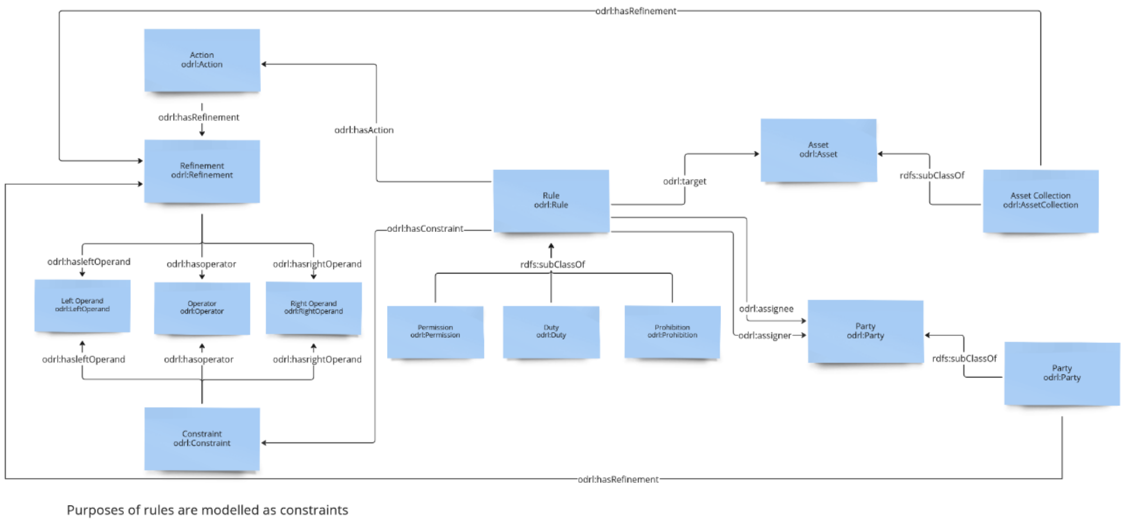


Figure 30: Data model of Rules showing modelling purposes as constraints.

Describe data model of interactions between APIs. ODRL policies created through this plugin can be passed to the Negotiation plugin, where it will be part of the UPCAST offer or request in the negotiation plugin.

Serialisation to JSON-LD

We serialise the action $\mathcal{A} = (A, R)$ as follows:

```
"action": A
        "refinement": [
                $for\ each\ c_i \in R$:
                        {"leftOperand": $\lambda_i$,
                        "op": $op_i$,
                        "rightOperand": $\rho_i$}
                ]
]
```

We serialise the assigner DP and assignee $DC = (N, R)$ as follows:

```
"assigner": DP,
"assignee": N
          "refinement": [
                    $for\ each\ c_i \in R$:
                              {"leftOperand": $\lambda_i$,
                              "op": $op_i$,
                              "rightOperand": $\rho_i$}
                    ]
]
```

We serialise the purpose $\mathcal{P} = (P, R)$ as follows:

```
"constraint": [
          {"leftOperand": "Purpose",
          "op": "eq",
          "rightOperand": P},
          {"and": [
                    $for\ each\ c_i \in R$:
                              {"leftOperand": $\lambda_i$,
                              "op": $op_i$,
                              "rightOperand": $\rho\ _i$}
                    ]
          }
]
```

We serialise the target $\mathcal{T} = (D, R)$ as follows:

```
"target": D
          "refinement": [
                    $for\ each\ c_i \in R$:
                              {"leftOperand": $\lambda_i$,
                              "op": $op_i$,
                              "rightOperand": $\rho_i$}
          ]
]
```

```
"rule":
          "action": A,
          "assignee": DC,
          "assigner": DP,
          "target": D
          "constraint": [
                    P,
                    $for\ each\ c_i \in R$:
                              {"leftOperand": $\lambda_i$,
                              "op": $op_i$,
                              "rightOperand": $\rho_i$}
                    ]
```

```
]
```

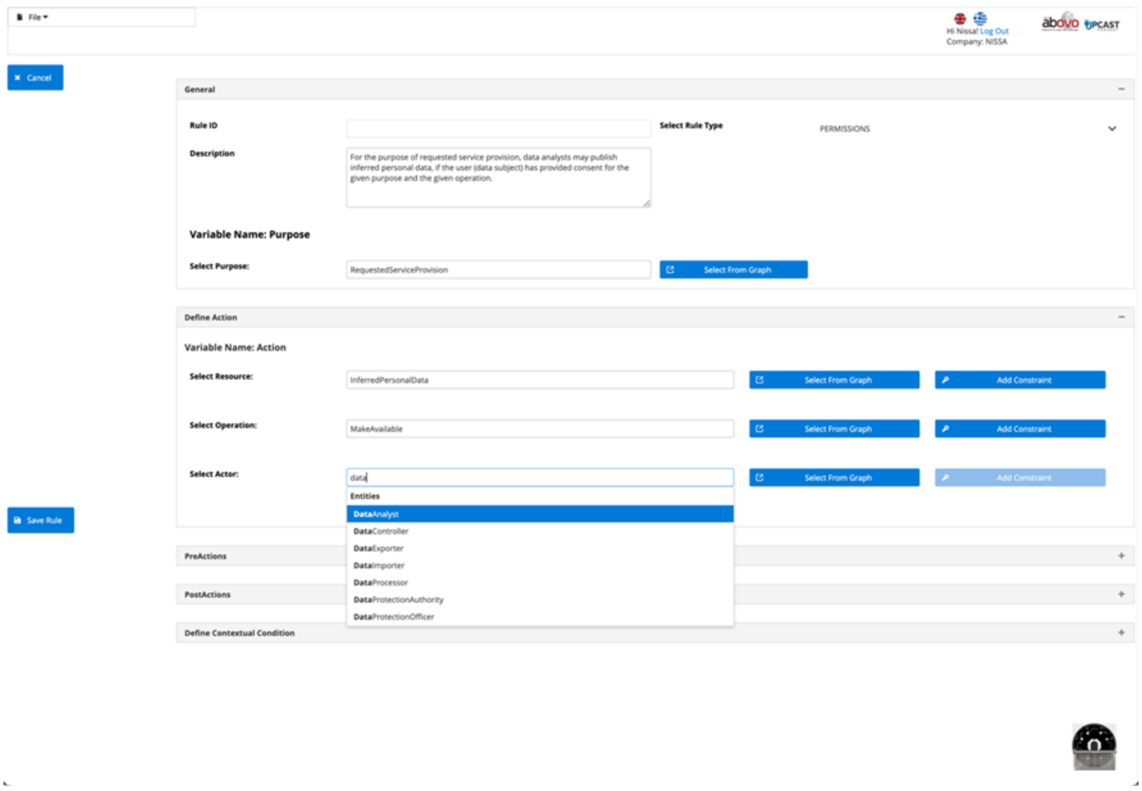
In the above serialization, “rule” can be either “permission”, “prohibition” or “duty” depending on the type of rule. Each of the elements of a rule is replaced with its corresponding serialisation, as defined above.

```
policy”:
        for each $\mathcal{R} \in \mathfrak{R}$:
                $\mathcal{R}$
]
```

Typical usages: A data consumer needs fitness data from users subscribed to their service for training personalisation. (0) The data consumer uploads ontologies that define terms such as fitness data, and purposes such as training personalisation (that cannot be found in the default ontologies ODRL and DPV). (1) The data consumer configures a new request form by specifying the general ontology, domain specific ontology and data schema ontology. (2) The data consumer chooses the data they wish to request (extracted from the data schema ontology), the purpose (selecting training personalisation from the ontology) and the data processing operations (extracted from the domain specific ontology) and the datetime interval during which they need the data. (3) The data consumer adds other more specific ODRL-compliant constraints through the UI. (4) The data consumer submits the request form, which is then sent to all data providers subscribed to the service. From the data provider side, a subscriber will (1) be informed that they have a request pending. (2) Access the policy plugin where they can click on the request form. (3) The data provider can opt in or out of providing any of the data in the request form, which will (4) produce an ODRL policy describing the permissions and prohibitions, which will be sent back to the data consumer.Consumer side:

Functionality: Consumer-side PUC provides the functionality for specifying an organisation’s internal policies that regulate its operations against a variety of provisions; in the general case, these may originate from applicable legislation (e.g., the GDPR) or other operational and data governance considerations. The user may create, update and delete access and usage control rules. Management of generic organisation-specific and legislation-related rules is provided by a dedicated UI and includes conflict resolution between rules and rules merging, i.e., mechanisms for the elimination of deprecated policies (i.e., overridden by other policies), as well as identification of conflicts between resource provider’s access and usage constraints and resource consumer’s access and usage intentions and subsequent suggestions for conflict resolution.

Interface: The OpenAPI specification for the Consumer-side PUC is available at: https://github.com/EU-UPCAST/OpenAPISpecification/tree/main/privacy_plugin/Consumer. Users may define and manage the policies of their organisations via the goodFlows Policy Model Editor, as shown in Figure 31 and Figure 32.


File

Hi Nissa! Log Out
Company: NISSA

Cancel

General

Rule ID:

Select Rule Type: PERMISSIONS

Description: For the purpose of requested service provision, data analysts may publish inferred personal data, if the user (data subject) has provided consent for the given purpose and the given operation.

Variable Name: Purpose

Select Purpose: RequestedServiceProvision — Select From Graph

Define Action

Variable Name: Action

Select Resource: InferredPersonalData — Select From Graph — Add Constraint

Select Operation: MakeAvailable — Select From Graph — Add Constraint

Select Actor: data — Select From Graph — Add Constraint

Entities
- DataAnalyst
- DataController
- DataExporter
- DataImporter
- DataProcessor
- DataProtectionAuthority
- DataProtectionOfficer

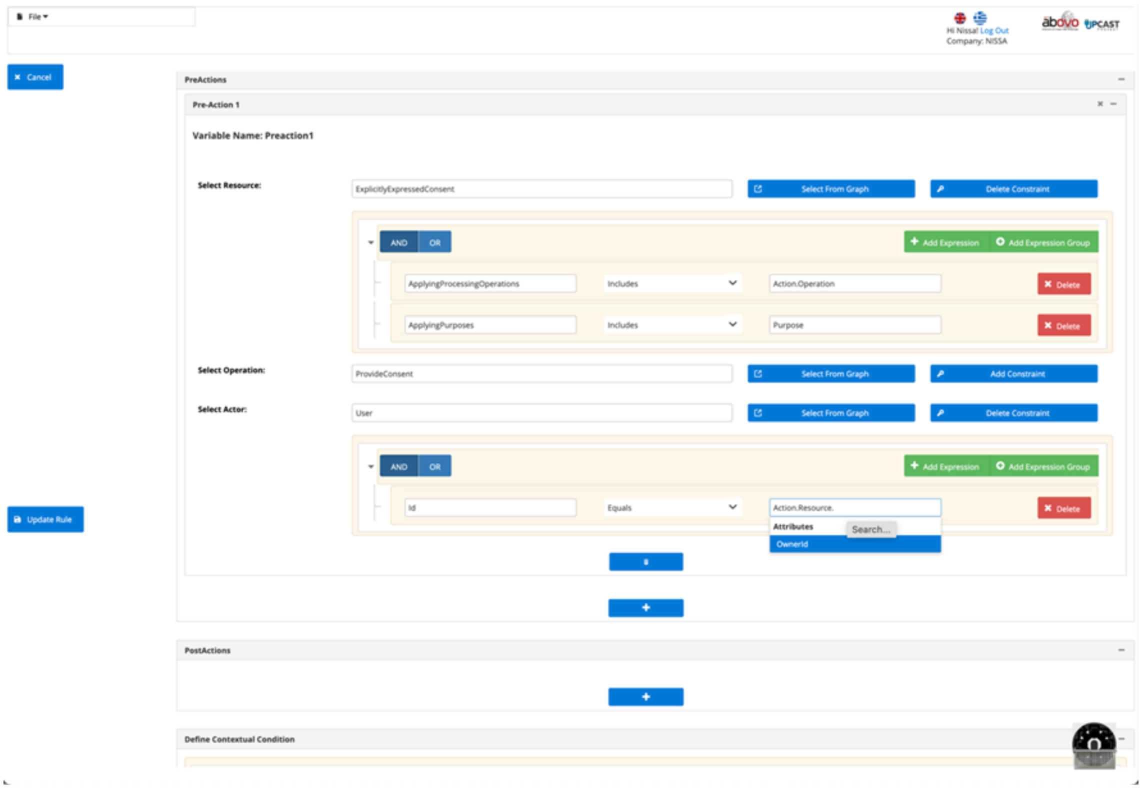
Save Rule

PreActions

PostActions

Define Contextual Condition

Figure 31: Rule Creation form, general rule information and definition of applicable actions.


File

Hi Nissa! Log Out
Company: NISSA

Cancel

PreActions

Pre-Action 1

Variable Name: Preaction1

Select Resource: ExplicitlyExpressedConsent — Select From Graph — Delete Constraint

AND OR — Add Expression — Add Expression Group

ApplyingProcessingOperations	Includes	Action.Operation	Delete
ApplyingPurposes	Includes	Purpose	Delete

Select Operation: ProvideConsent — Select From Graph — Add Constraint

Select Actor: User — Select From Graph — Delete Constraint

AND OR — Add Expression — Add Expression Group

Id	Equals	Action.Resource.	Delete

Attributes — Search...
- OwnerId

Update Rule

PostActions

Define Contextual Condition

Figure 32: Rule Creation form - defining preActions and constraints upon action entities.

Data Model: At the consumer-side, access and usage control rules are defined by means of the goodFlows Access and Usage Control model, a design choice made for leveraging and building upon existing functionality of goodFlows related to the management of rules (inheritance of rules across hierarchies, complex reasoning algorithms for rule combination and conflict resolution among rules) and the verification of DPWs against the underlying policy model. The respective rule language is in line with the ODRL vocabulary, allowing for the rather straightforward translation between the two languages; RP's constraints expressed in ODRL generated upon Resource Specification are translated into the underlying semantic policy language and vice versa in order to assist the negotiation procedure.

In goodFlows, an access and usage control rule is modelled through the following structure (Figure 4):

$$\left.\begin{matrix} Permission \\ Prohibition \\ Obligation \end{matrix}\right\} (pu, act, preAct, cont, postAct)$$

where *act* is the action that the rule applies to; *pu* is the purpose for which *act* is permitted/prohibited/obliged to be executed; *cont* is a structure of contextual parameters; *preAct* is a structure of actions that should have preceded; *postAct* refers to the action(s) that must be executed following the rule enforcement.

Actions constitute conceptual triples of *{actor, operation, resource, organisation}* and are formed leveraging the entities of the Information Model. In more detail, action's entities are defined as a tuple *⟨concept, constraint⟩*, where *concept* is a semantic type, and *constraint* is an expression (or a logical relation thereof) assigning values to attributes and/or sub-concepts of the given entity. This mechanism allows the definition of access and usage control rules inline with the Attribute-based Access Control (ABAC) paradigm. For this, appropriate concepts are provided, including Expressions, the instances of which provide for modelling atomic constraints with a subject and a value, and Variables, that allow the specification of concepts in relation to other concepts (e.g., set the value of a resource's constraint to be relative to the semantic type of the actor), thereby maximising expressiveness.

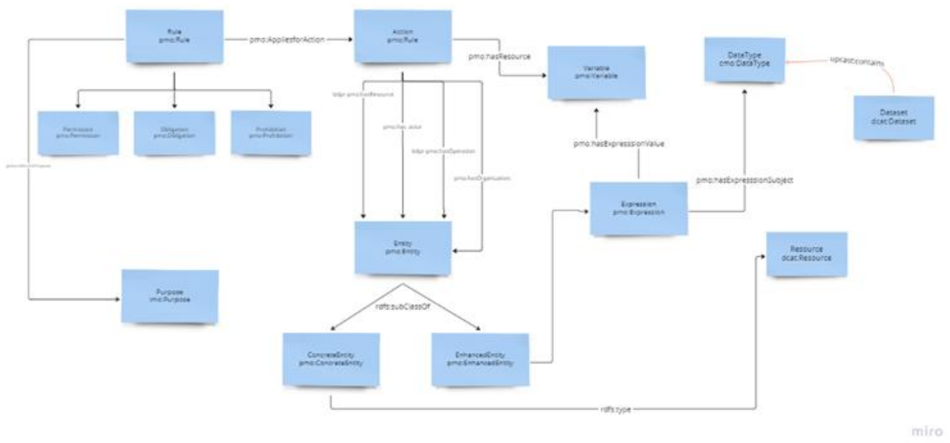
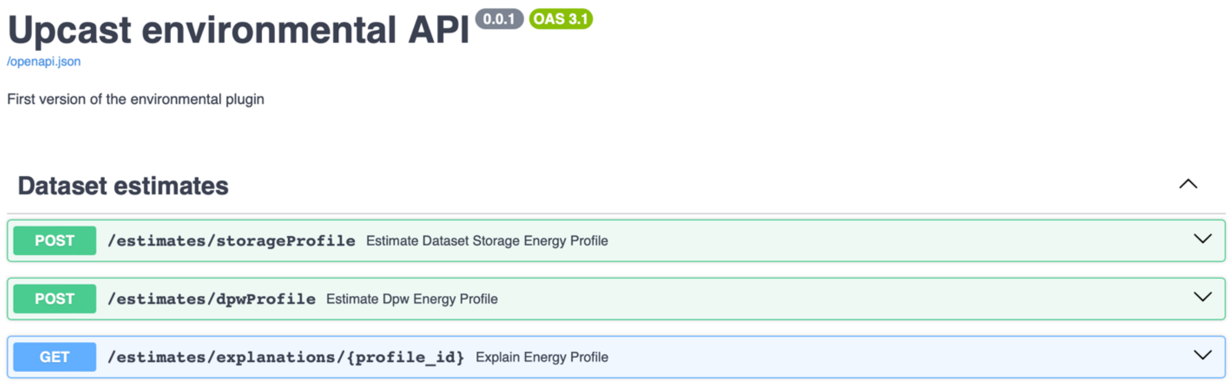


Figure 33: goodFlows access and usage control rule data model.

Typical usages: The consumer defines organisation-specific access and usage control rules and rules prescribed by applicable regulations (e.g., GDPR). Any DPW designed by the consumer will be first verified against these rules in order to be compliant with the consumer's internal policies. The DPW reflects the intentions of the consumer regarding datasets offered by providers; for the latter, specific access and usage constraints apply, expressed in ODRL by the providers, which are translated to the consumer's policy language in order for the DPW to be also checked against them and conflicts identification and possible resolution to take place.

5.6 Environmental Impact

Functionality: The environmental impact optimiser plugin includes efficient AI models, power monitoring and visualisation tools to estimate the environmental impact of a data processing workflow, by profiling datasets based on their potential impact and calculating energy metrics such as estimated carbon footprint. The plugin estimates the energy consumption and intensity of a dataset based on its metadata and hardware information where the resource is created, stored or processed. This will be used to create an energy profile for each resource. Furthermore, using Explainable AI (XAI) techniques, the end user will obtain transparent feedback on the algorithm's decision to assign a particular energy profile to a dataset.
Interface: Figure 34 shows the REST API Swagger interface. There are three endpoints: one to estimate the dataset energy consumption at idle, one to estimate the data processing workflow energy consumption and the third endpoint to explain the energy consumption estimation. The current Open API specification is available in the project's GitHub repository https://github.com/EU-UPCAST/OpenAPISpecification/tree/main/environmental_plugin


Upcast environmental API 0.0.1 OAS 3.1
/openapi.json

First version of the environmental plugin

Dataset estimates

POST /estimates/storageProfile Estimate Dataset Storage Energy Profile

POST /estimates/dpwProfile Estimate Dpw Energy Profile

GET /estimates/explanations/{profile_id} Explain Energy Profile

Figure 34: The environmental plugin swagger interface.

Data Model: Figure 35 shows the data model of the environmental plugin. The plugin inputs and outputs respect the upcast data model.

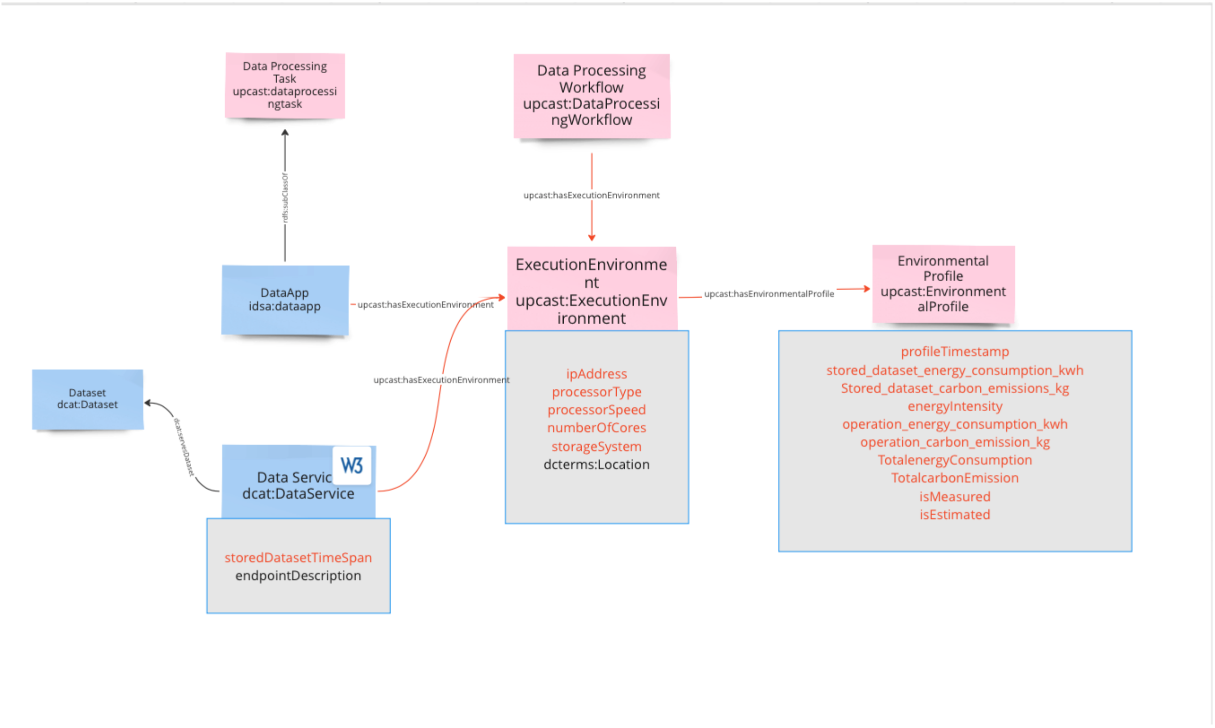

Data Processing Task upcast:dataprocessingtask
Data Processing Workflow upcast:DataProcessingWorkflow
upcast:hasExecutionEnvironment
rdfs:subClassOf
DataApp idsa:dataapp
upcast:hasExecutionEnvironment
ExecutionEnvironment upcast:ExecutionEnvironment
upcast:hasEnvironmentalProfile
Environmental Profile upcast:EnvironmentalProfile
profileTimestamp
stored_dataset_energy_consumption_kwh
Stored_dataset_carbon_emissions_kg
energyIntensity
operation_energy_consumption_kwh
operation_carbon_emission_kg
TotalenergyConsumption
TotalcarbonEmission
isMeasured
isEstimated
ipAddress
processorType
processorSpeed
numberOfCores
storageSystem
dcterms:Location
upcast:hasExecutionEnvironment
Dataset dcat:Dataset
dcat:servesDataset
Data Servic dcat:DataService
storedDatasetTimeSpan
endpointDescription

Figure 35: Data model of the Environmental plugin.

Typical usages: The environmental plug-in can be used by a provider to estimate the energy consumption and intensity of a resource storage at idle based on its metadata and on information of the hardware where the resource will be stored. It can also be used by a consumer to estimate the DPW energy consumption based on its metadata and on information of the hardware where it will be executed.

5.7 Pricing

Functionality: The data pricing plugin aims to provide sellers and buyers with a hint of the market price of a dataset taking as inputs its metadata features and based on market prices observed in already existing commercial marketplaces, as well as similar datasets existing in the market and a normalised classification of the dataset. The plugin allows the users to:

- Find data products sold in the market similar to an input description
- Find a reasonable price range for an input data product
- Find an explanation for the suggested price range, including the most relevant features affecting the price prediction and the most relevant words in the description.

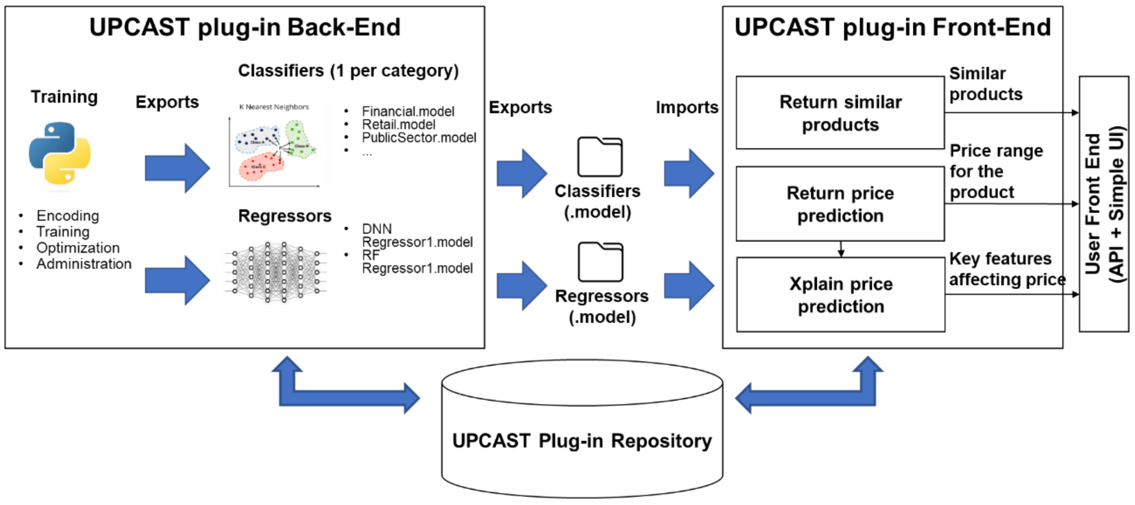
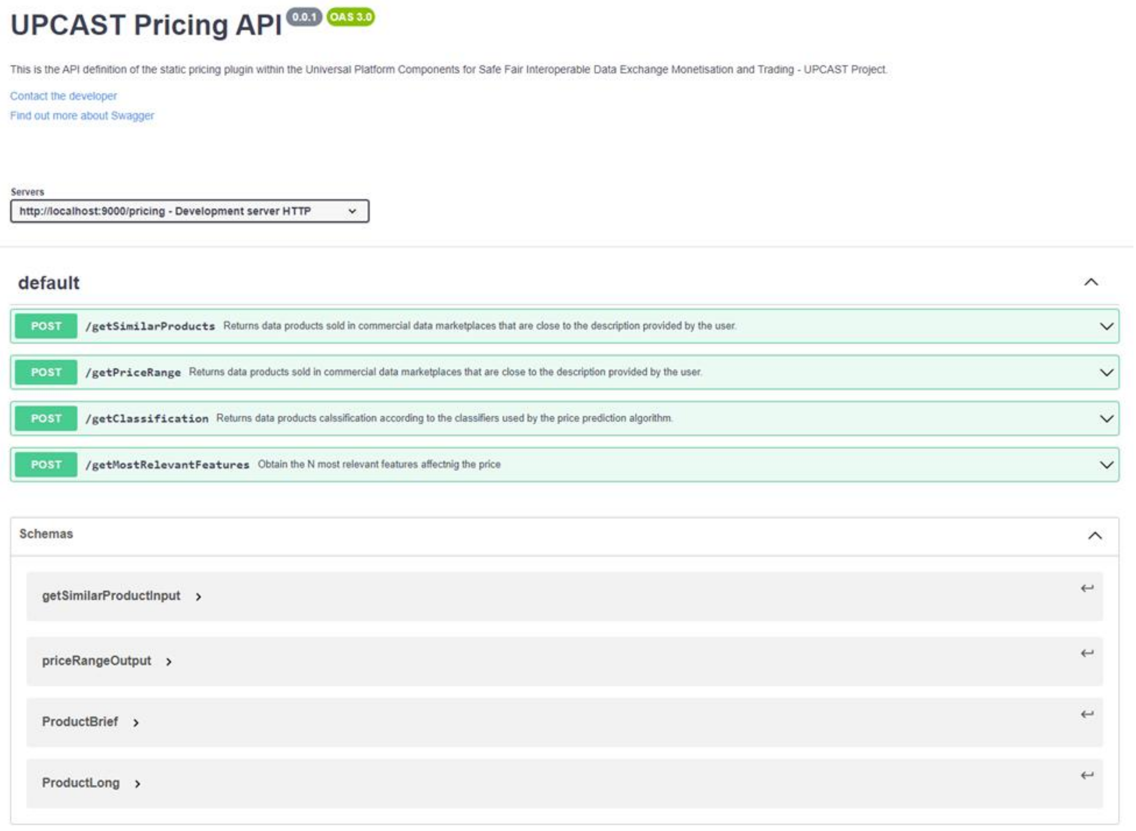


Figure 36: UPCAST Static pricing plugin architecture.

Figure 37 shows an excerpt of the OpenAPI specification of the pricing plugin. The full specification is available at https://github.com/EU-UPCAST/pricing-plugin.



UPCAST Pricing API 0.0.1 OAS 3.0

This is the API definition of the static pricing plugin within the Universal Platform Components for Safe Fair Interoperable Data Exchange Monetisation and Trading - UPCAST Project.

Contact the developer
Find out more about Swagger

Servers
http://localhost:9000/pricing - Development server HTTP

default

POST `/getSimilarProducts` Returns data products sold in commercial data marketplaces that are close to the description provided by the user.

POST `/getPriceRange` Returns data products sold in commercial data marketplaces that are close to the description provided by the user.

POST `/getClassification` Returns data products calssification according to the classifiers used by the price prediction algorithm.

POST `/getMostRelevantFeatures` Obtain the N most relevant features affectnig the price

Schemas

getSimilarProductInput

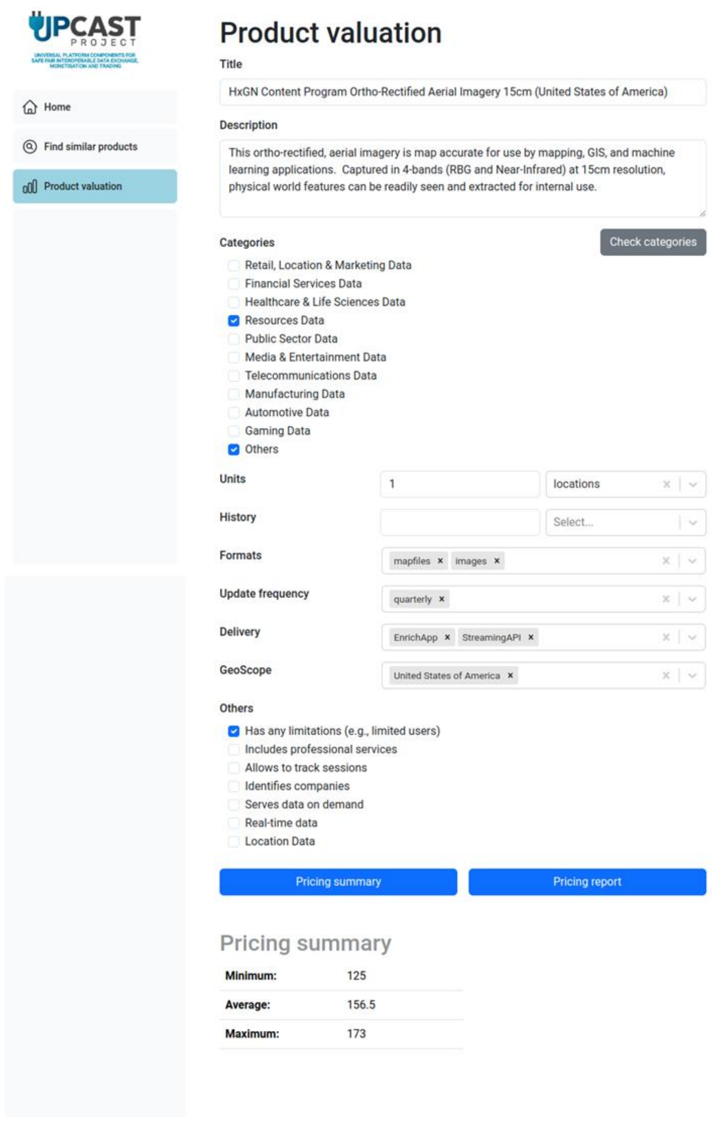
priceRangeOutput

ProductBrief

ProductLong

Figure 37: OpenAPI3.0 of the Static Pricing plugin.

Additionally, a front end was developed to allow users to interact with the application in a more friendly way, and to demonstrate the functionality of the API. The front end provides basic functionality to call the Pricing API endpoints and visualize the results and explanations as shown in Figure 38 and Figure 39.



Product valuation

Title

HxGN Content Program Ortho-Rectified Aerial Imagery 15cm (United States of America)

Description

This ortho-rectified, aerial imagery is map accurate for use by mapping, GIS, and machine learning applications. Captured in 4-bands (RBG and Near-Infrared) at 15cm resolution, physical world features can be readily seen and extracted for internal use.

Categories — Check categories

- Retail, Location & Marketing Data
- Financial Services Data
- Healthcare & Life Sciences Data
- [✓ Resources Data]
- Public Sector Data
- Media & Entertainment Data
- Telecommunications Data
- Manufacturing Data
- Automotive Data
- Gaming Data
- [✓ Others]

Units	1 — locations
History	Select...
Formats	mapfiles, images
Update frequency	quarterly
Delivery	EnrichApp, StreamingAPI
GeoScope	United States of America

Others

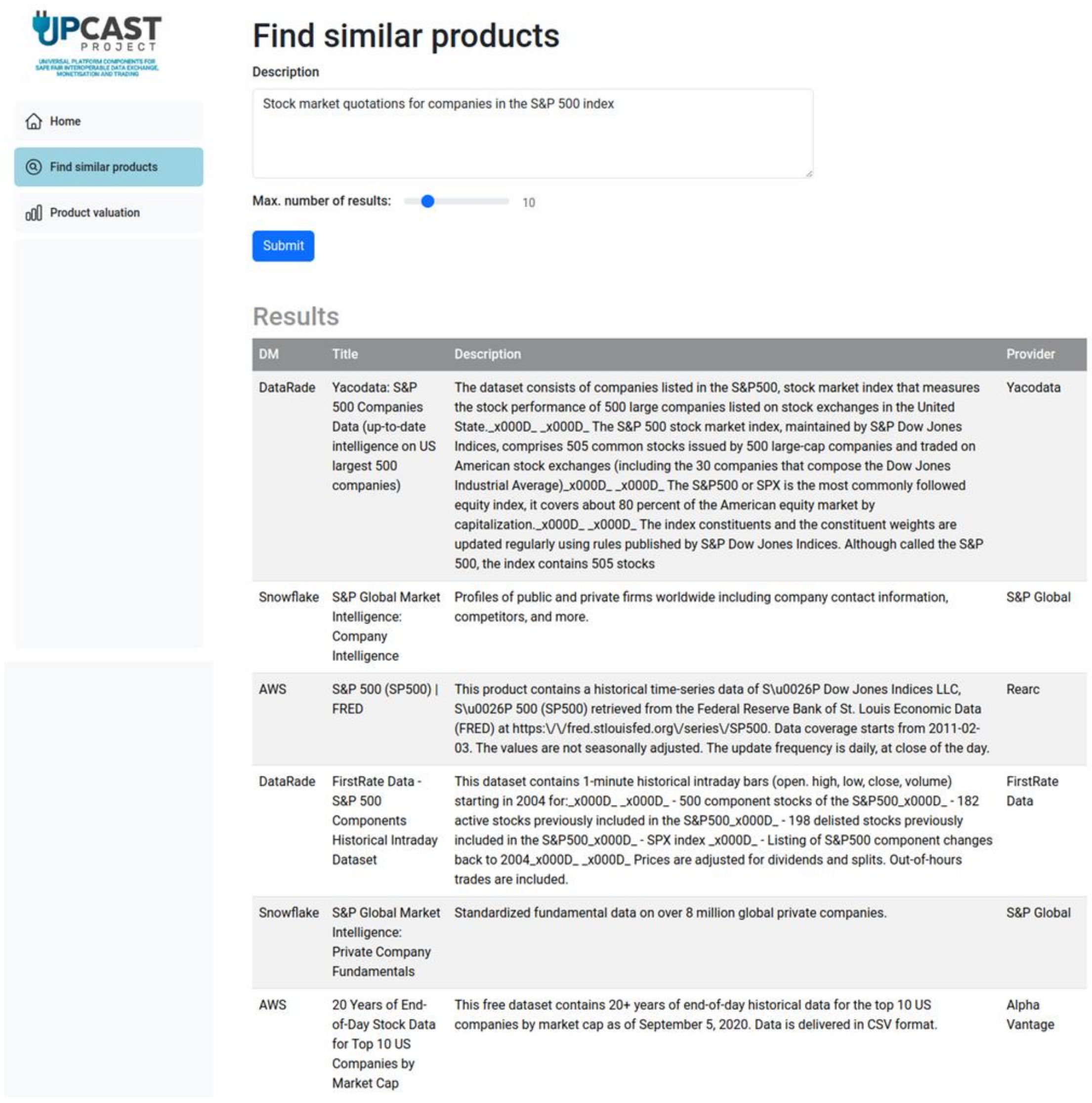
- [✓ Has any limitations (e.g., limited users)]
- Includes professional services
- Allows to track sessions
- Identifies companies
- Serves data on demand
- Real-time data
- Location Data

Pricing summary — Pricing report

Pricing summary

Minimum:	125
Average:	156.5
Maximum:	173

Figure 38: UPCAST Static Pricing interface.


UPCAST PROJECT
UNIVERSAL PLATFORM COMPONENTS FOR SAFE FAIR INTEROPERABLE DATA EXCHANGE, MONETISATION AND TRADING
Home
Find similar products
Product valuation
Submit

Find similar products

Description

Stock market quotations for companies in the S&P 500 index

Max. number of results: 10

Results

DM	Title	Description	Provider
DataRade	Yacodata: S&P 500 Companies Data (up-to-date intelligence on US largest 500 companies)	The dataset consists of companies listed in the S&P500, stock market index that measures the stock performance of 500 large companies listed on stock exchanges in the United State._x000D_ _x000D_ The S&P 500 stock market index, maintained by S&P Dow Jones Indices, comprises 505 common stocks issued by 500 large-cap companies and traded on American stock exchanges (including the 30 companies that compose the Dow Jones Industrial Average)_x000D_ _x000D_ The S&P500 or SPX is the most commonly followed equity index, it covers about 80 percent of the American equity market by capitalization._x000D_ _x000D_ The index constituents and the constituent weights are updated regularly using rules published by S&P Dow Jones Indices. Although called the S&P 500, the index contains 505 stocks	Yacodata
Snowflake	S&P Global Market Intelligence: Company Intelligence	Profiles of public and private firms worldwide including company contact information, competitors, and more.	S&P Global
AWS	S&P 500 (SP500) | FRED	This product contains a historical time-series data of S\u0026P Dow Jones Indices LLC, S\u0026P 500 (SP500) retrieved from the Federal Reserve Bank of St. Louis Economic Data (FRED) at https:\/\/fred.stlouisfed.org\/series\/SP500. Data coverage starts from 2011-02-03. The values are not seasonally adjusted. The update frequency is daily, at close of the day.	Rearc
DataRade	FirstRate Data - S&P 500 Components Historical Intraday Dataset	This dataset contains 1-minute historical intraday bars (open. high, low, close, volume) starting in 2004 for:_x000D_ _x000D_ - 500 component stocks of the S&P500_x000D_ - 182 active stocks previously included in the S&P500_x000D_ - 198 delisted stocks previously included in the S&P500_x000D_ - SPX index _x000D_ - Listing of S&P500 component changes back to 2004_x000D_ _x000D_ Prices are adjusted for dividends and splits. Out-of-hours trades are included.	FirstRate Data
Snowflake	S&P Global Market Intelligence: Private Company Fundamentals	Standardized fundamental data on over 8 million global private companies.	S&P Global
AWS	20 Years of End-of-Day Stock Data for Top 10 US Companies by Market Cap	This free dataset contains 20+ years of end-of-day historical data for the top 10 US companies by market cap as of September 5, 2020. Data is delivered in CSV format.	Alpha Vantage

Figure 39: UPCAST Static Pricing functions to obtain similar datasets in the market.

Data Model: The pricing plug-in will rely upon the data about data products and data vendors from different commercial marketplaces to support the work in [2] and [3].
The fields and features used to specify data products and obtain a price have been included in the vocabulary of UPCAST.

Typical usages: The pricing plugin can be used in several steps of the go-to-market process of a data product. First, it can be used to point to other relevant vendors and products offering similar services when designing the data products. Second, it can be used by data providers as a first reference to set the price of data products advertised in data marketplaces. Moreover, it can be used by data consumers as a reference to get an idea of the budget required for a sourcing operation, or as a market reference to decide whether the price asked by a seller falls within a reasonable interval according to prices observed in the market. Finally, it could be helpful in negotiation processes to help both parties agree on the price of data products involved in a data transaction.

5.8 Data Processing Workflow Modelling

Functionality: This plugin enables the specification of UPCAST Data Processing Workflows (DPWs), serving as the primary means through which data consumers state their intentions for the data they seek to acquire. These intentions are derived from jointly considering a variety of aspects, including: the processing operations intended to be performed; the entities in direct or indirect control of their execution; the attributes of the acquired data and the conditions under which any processing and exchange is meant to take place; the stated purposes that the DPW in question is intended to serve. As such, DPWs constitute the basis for subsequent negotiation on the data consumer side. Part of the DPW modelling environment is shown in Figure 40.

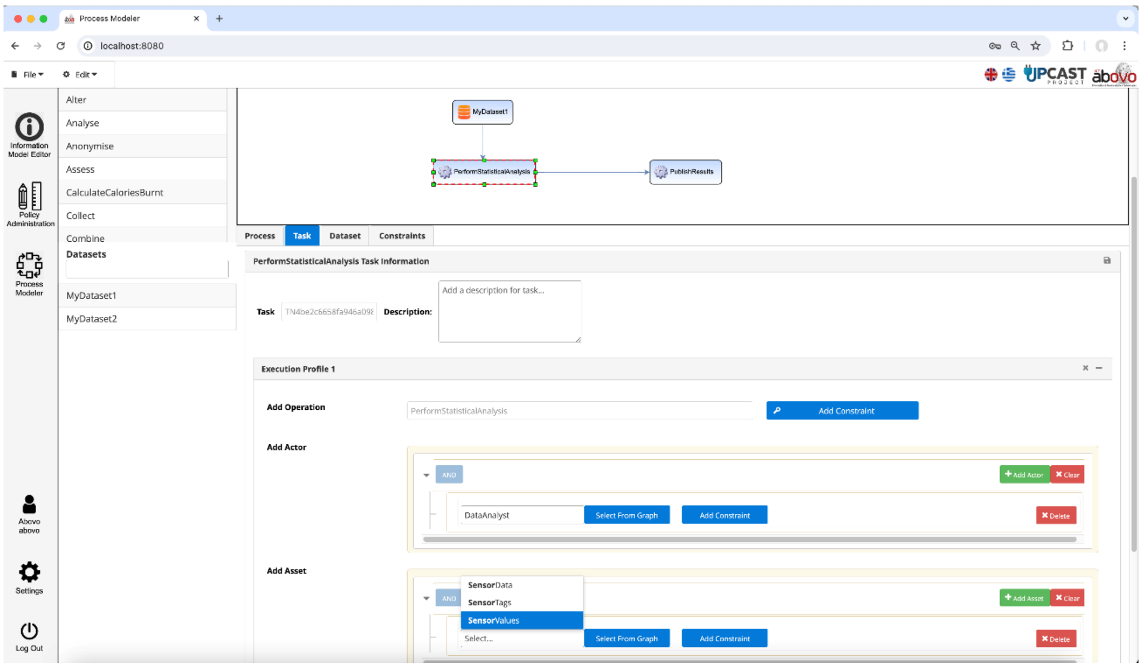
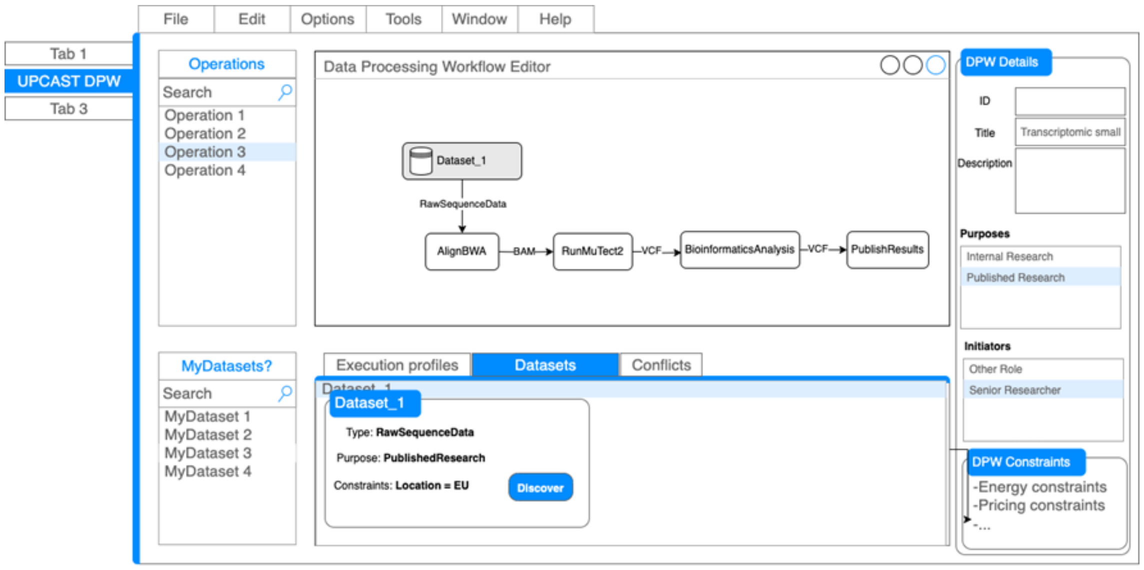


Figure 40: DPW modelling.

Furthermore, this plugin allows the resource consumer to interact with other UPCAST plugins, comprising the entry point to several UPCAST offerings and functionalities, thereby playing the role of the consumer dashboard. In this context, the plugin enables the consumer, upon the specification of a data processing workflow, to search and discover dataset to be used within the workflow, to define environmental and pricing constraints, to invoke the corresponding plugins in order to assess environmental impact and calculate fair pricing, while comprising the consumer's gateway towards negotiating the acquisition of datasets.

Figure 41 illustrates dataset discovery from within the workflow modelling environment, Figure 42 assumes that a dataset has been discovered and depicts its properties, whereas Figure 43 showcases the conflicts identified that will result in the initiation of the negation process. In the bottom right of these wireframes, the buttons for defining and viewing environmental and pricing constraints are shown.

Interface: Figure 44 is an excerpt of the OpenAPI specification of the DPW modelling. The full specification is available at https://github.com/EU-UPCAST/OpenAPISpecification/tree/main/privacy_plugin/Consumer



Type: **RawSequenceData**

Purpose: **PublishedResearch**

Constraints: **Location = EU**

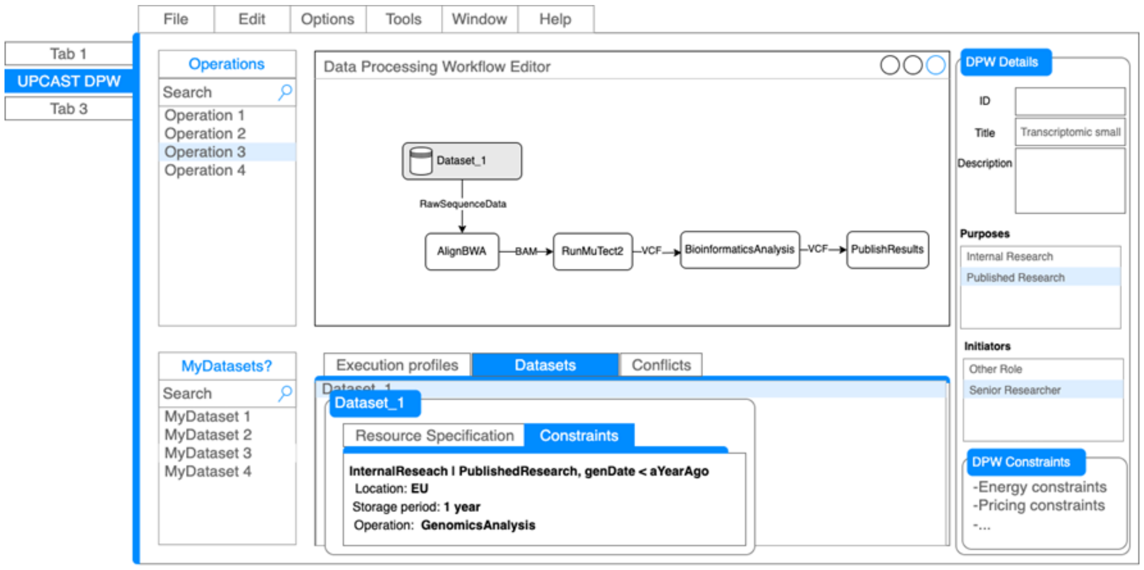
Title: Transcriptomic small

Purposes: Internal Research, Published Research

Initiators: Other Role, Senior Researcher

-Energy constraints
-Pricing constraints
-...

Figure 41: Dataset discovery



InternalReseach I PublishedResearch, genDate < aYearAgo
Location: **EU**
Storage period: **1 year**
Operation: **GenomicsAnalysis**

Title: Transcriptomic small

Purposes: Internal Research, Published Research

Initiators: Other Role, Senior Researcher

-Energy constraints
-Pricing constraints
-...

Figure 42: Properties of the discovered dataset.

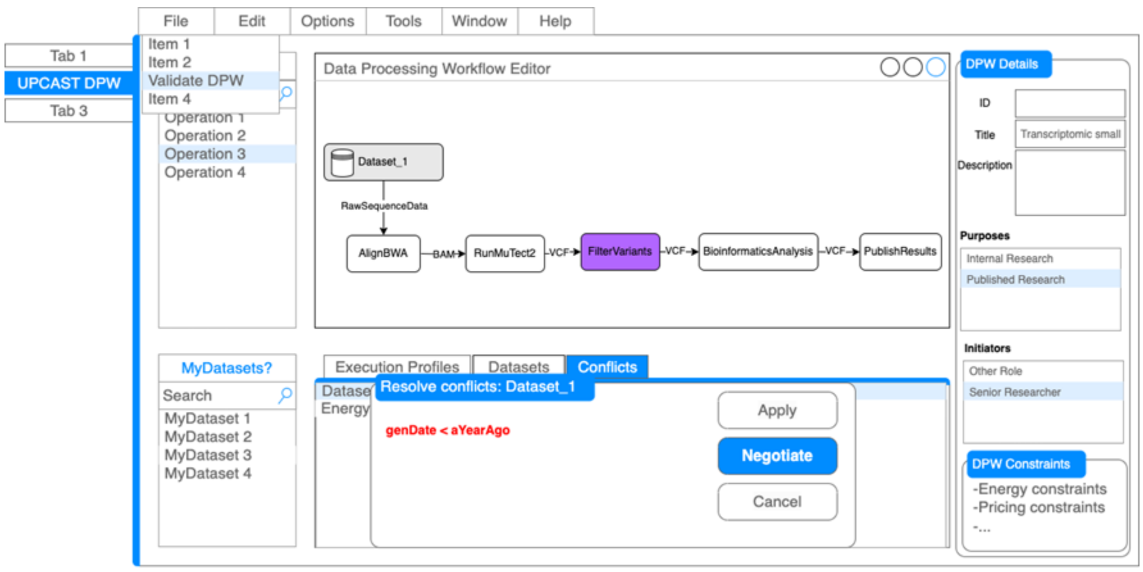
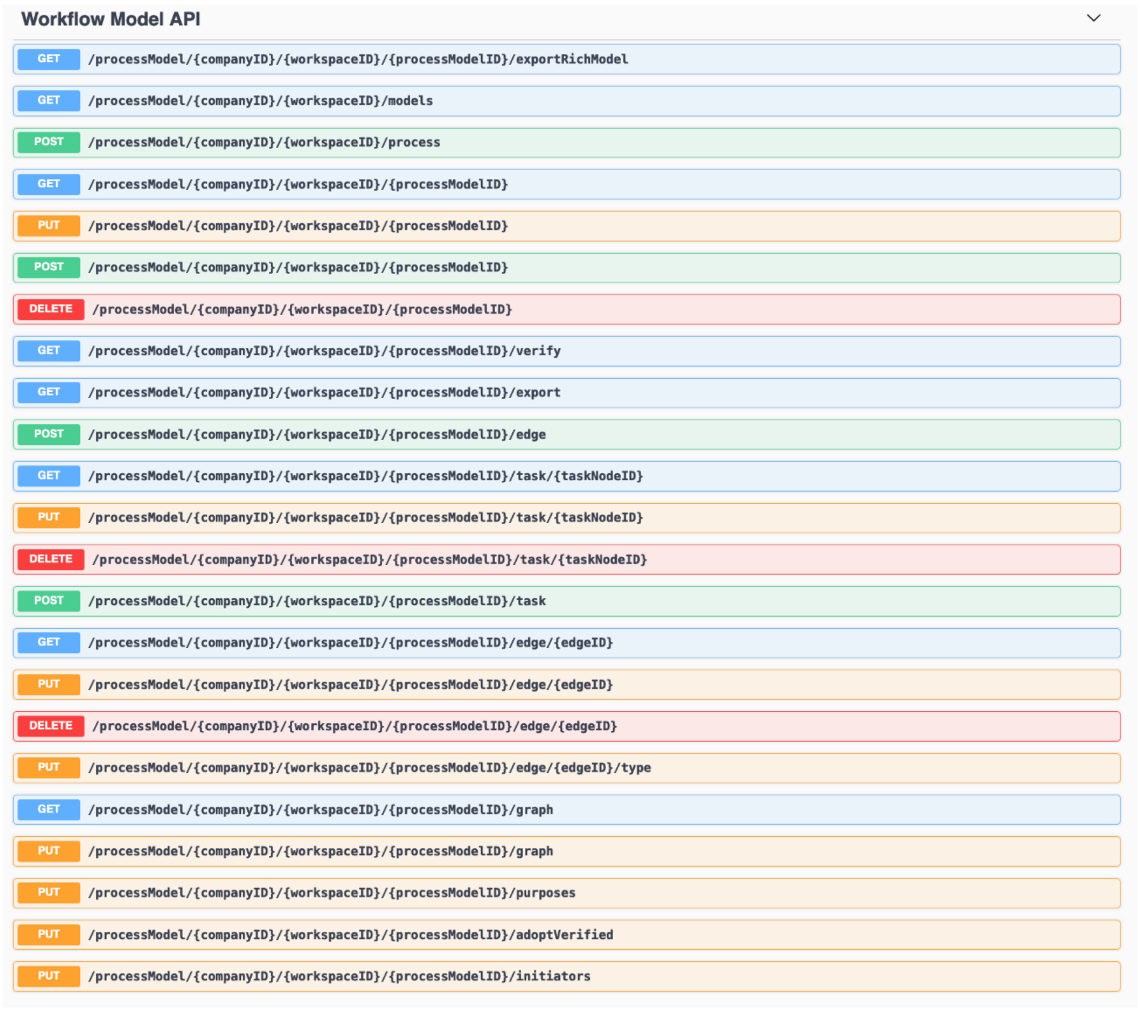


Figure 43: Conflict identification prior to negotiation.



Workflow Model API

```
GET     /processModel/{companyID}/{workspaceID}/{processModelID}/exportRichModel
GET     /processModel/{companyID}/{workspaceID}/models
POST    /processModel/{companyID}/{workspaceID}/process
GET     /processModel/{companyID}/{workspaceID}/{processModelID}
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}
POST    /processModel/{companyID}/{workspaceID}/{processModelID}
DELETE  /processModel/{companyID}/{workspaceID}/{processModelID}
GET     /processModel/{companyID}/{workspaceID}/{processModelID}/verify
GET     /processModel/{companyID}/{workspaceID}/{processModelID}/export
POST    /processModel/{companyID}/{workspaceID}/{processModelID}/edge
GET     /processModel/{companyID}/{workspaceID}/{processModelID}/task/{taskNodeID}
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/task/{taskNodeID}
DELETE  /processModel/{companyID}/{workspaceID}/{processModelID}/task/{taskNodeID}
POST    /processModel/{companyID}/{workspaceID}/{processModelID}/task
GET     /processModel/{companyID}/{workspaceID}/{processModelID}/edge/{edgeID}
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/edge/{edgeID}
DELETE  /processModel/{companyID}/{workspaceID}/{processModelID}/edge/{edgeID}
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/edge/{edgeID}/type
GET     /processModel/{companyID}/{workspaceID}/{processModelID}/graph
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/graph
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/purposes
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/adoptVerified
PUT     /processModel/{companyID}/{workspaceID}/{processModelID}/initiators
```

Figure 44: DPW API.

Data Model: UPCAST DPWs are defined as ontologies, based on an appropriate metamodel called Workflow Model Ontology (WMO). According to this, a DPW is a graph where tasks are the nodes, and the edges define the sequence of application, as well as overall data and control flow (Figure 45). A DPW is meant to serve one or more purposes, intended to be initiated by specific entities or entities with particular characteristics, and is linked to a given execution environment through proper interaction with the Environmental plugin.

Edges carry Information Entities that define the nature of data to be transferred from one task to the next. A wmo:InformationEntity may refer to a specific dataset, or describe transferred data in abstract terms, through a data type and potential additional constraints. An edge may also be characterised by flow conditions and constraints, further specifying and/or restricting the occurrence of implied transition.

A TaskNode has one to several execution profiles. An ExecutionProfile describes an actor that applies an operation (implemented by a DataProcessingTask) on an Asset (often a Dataset), subject to some task conditions. More specifically, wmo:ActorEntity instances denote the entities assigned with the execution of tasks. They may be defined at either concrete (e.g., a particular person or organisation) or abstract level (e.g., some role and possibly additional constraints). Asset Entities represent the objects on which tasks are performed, meaning that they are accessed, processed or modified for the purpose of delivering the corresponding functionalities. They can similarly be defined in either concrete or abstract terms. In UPCAST they are typically expected to refer to (types of) datasets.

The wmo:OperationEntity describes the type of functionality executed at each workflow step, and is mapped to an odrl:Action. It is primarily defined at an abstract level, through the corresponding operation type, and linked to its concrete materialisation through the upcast:implementedBy property.

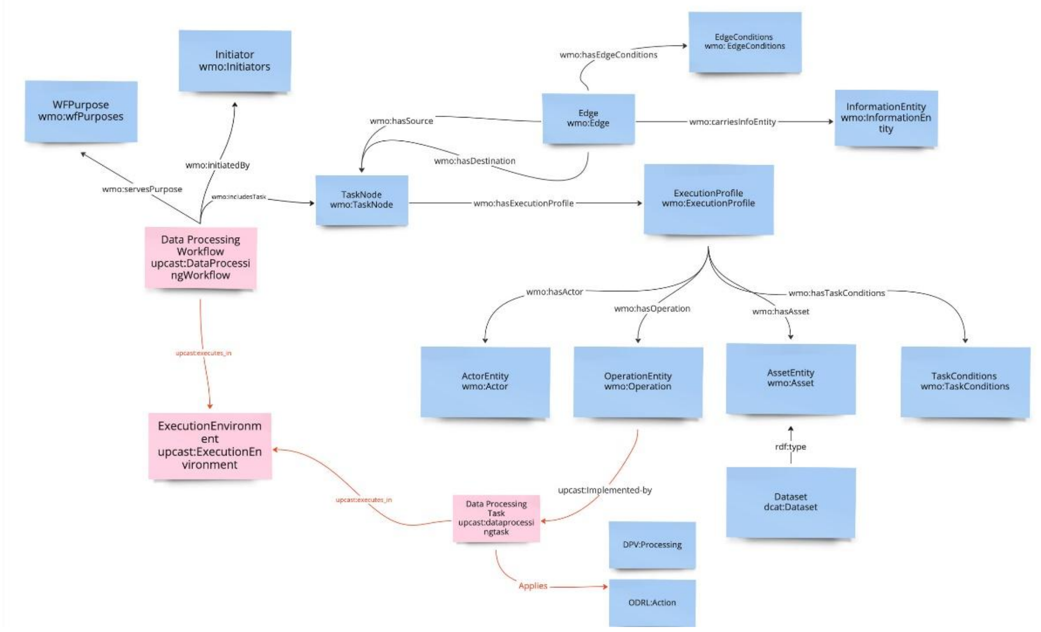


Figure 45: DPW Data model.

Typical usages: At this point, the typical interactions that have been specified to involve the DPW plugin include the following (Figure 46): 1) Define a new DPW with known —i.e., already discovered— datasets (step 1); 2) Validate the DPW against both internal organisational policies as well as dataset provider's constraints, making use of the functionalities of the PUC plugin (step 2); 3) View the validated form of the DPW, together with visualisations of potential conflicts with individual data providers (step 3); 4) Per provider, accept their preferences as declared in the resource specification by means of the ODRL policies, and update the DPW accordingly, in order to proceed with the agreement through interaction with the Negotiation plugin (step 4); 5) Alternatively, initiate negotiation by invoking the Negotiation plugin (step 5), whereby any potential conflicts returned by the PUC are communicated to the user (steps 6[10], 7); 6) View the results of the negotiation per provider (step 8); 7) Validate the DPW taking into account any existing negotiation results (step 9).

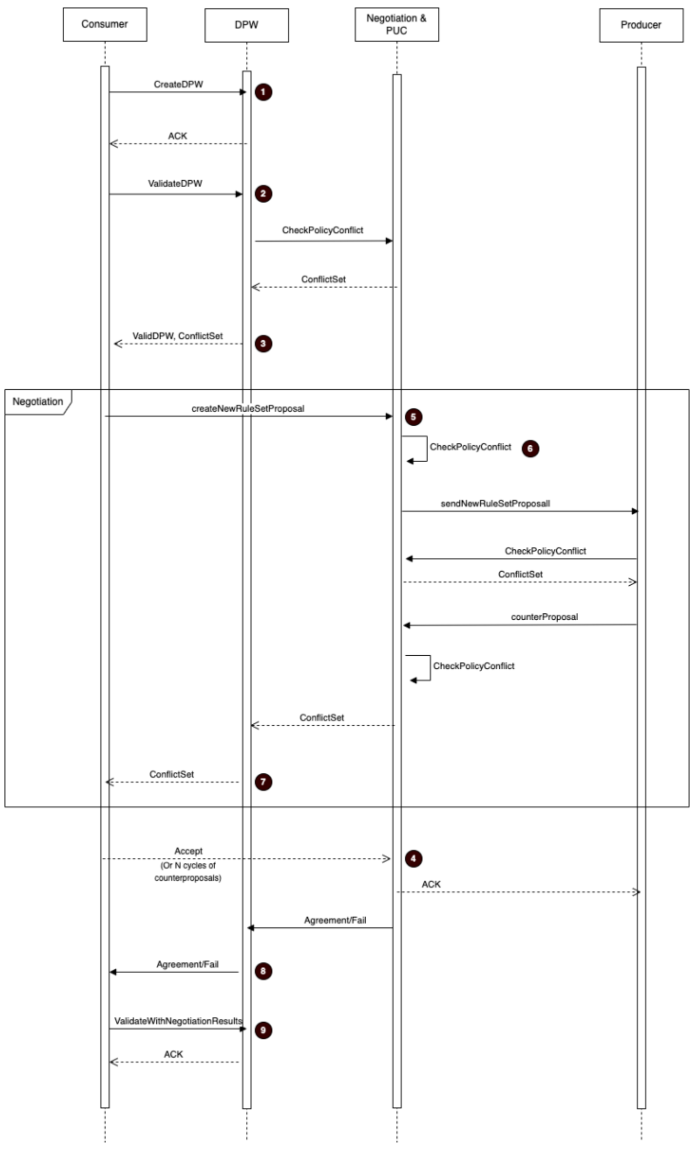


Figure 46: Interactions with the DPW plugin.

Integrations with additional plugins are expected with the second version of the DPW plugin, as is, for instance, the case with the Discovery plugin for the dynamic identification of matching datasets.

5.9 Federated Machine Learning

Federated Machine Learning (FML) is a machine learning technique that enables training models on distributed datasets without the need to centralize the data. In FML, the model is trained across multiple decentralized edge devices or servers holding local data samples,

[10] This step is introduced to ensure that any modifications to the DPW following step 5 are accounted for.

without exchanging them. This approach addresses critical issues around data privacy, security, access rights, and heterogeneous data.

There are multiple benefits of FML:

- **Data Privacy**: Raw data never leaves the local devices, protecting user privacy.
- **Regulatory Compliance**: Helps in complying with data protection regulations like GDPR.
- **Reduced Data Transfer**: Only model updates are shared, reducing bandwidth usage.
- **Access to Diverse Data**: Enables learning from a wide range of data sources without centralization.
- **Continuous Learning**: Models can be updated frequently with fresh, real-world data.
- **Scalability**: Can handle large numbers of participants and huge amounts of decentralized data.
- **Reduced Central Storage Need**: No need for a large central data repository.
- **Collaboration**: Allows multiple organizations to collaborate without sharing sensitive data.

A usage scenario in the context of UPCAST is as follows, a Data Consumer needs to train a Machine Learning model. She discovers in a Data Marketplace multiple Data Providers that own data that could be useful to train the model. However, the usage constraints of the Data Providers prohibit the transfer of data outside of their premises. Providers are still interested in doing business with Data Consumer and are willing to use their own computational resources to collaborate with the Consumer without having to transfer data.

We regard FML as a special kind of Data Operation, we assume the FML plugin is available in the resource catalog of the Data Marketplace used (or IDSA DataApp Store). Its metadata can be then used as a step in a Data Processing Workflow, connecting the FML plugin with the rest of the UPCAST infrastructure. The continuation of the usage scenario above is as follows:

1. The Data Consumer creates a Data Processing Workflow including a FML step. For ease of exposition, we assume FML is the only step.
2. The Data Consumer negotiates with each provider the usage conditions of each dataset, assisted by the DPW plugin to check there are no conflicts between the agreements of multiple providers.
3. Once all negotiations have been successfully completed, the FML plugin provides the Data Consumer and Data Providers with the setup need to train a Machine Learning model in a Federated way.

Figure 47 describes the general setup of the FML plugin. The central server coordinates the learning process and aggregates model updates. Clients (edge devices or local servers) train the model on their local data. Only model updates are shared, not the raw data.

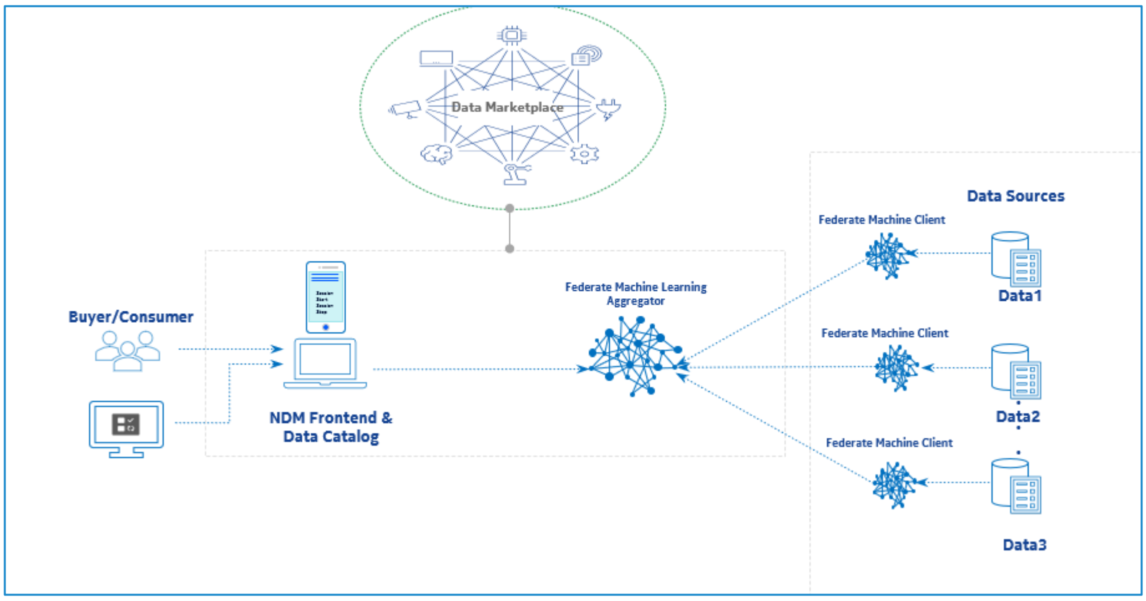


Figure 47: Federated Machine Learning plugin high level design.

An important condition that needs to be agreed during negotiation is that Data Providers must make available computing infrastructure to participate in the Federated training of a ML model. They must be able to install the FML plugin agent on their server and their server should be able to run a machine learning process on the local data.

5.10 Data Integration and Exchange

Functionality: In a data integration/exchange scenario, we have a number of distributed data sources, associated schema mappings, target/ontology dependencies, and a query to answer. There are two prevalent approaches to answering the query. Data Exchange (DE) uses "forward-chaining" algorithms to compute the entailment closure of the schema mappings and the ontology axioms. A prevalent formalisation of these algorithms is the chase family of algorithms which consolidates the data to a centralised warehouse ready for query answering. Virtual Integration, also known as Ontology Based Data Access (OBDA), or Ontology Mediated Query Answering, query rewriting, or "backward-chaining", processes the ontology axioms and mappings "backwards" (from consequent to premise) in order to rewrite the query such that it can be answered directly over the sources.

Our UPCAST Data Integration Operation is essentially an entire framework for combining different algorithms, implementations and systems in an end-to-end approach to Data Warehousing (DW) or Virtual Integration (VI). Our framework identifies and packages a number of DW and VI systems writing translators, parsers and integration code to combine and easily transfer between these implementations. Providing a unified framework allows us to investigate systems' limits and their interplay. We also make it easier to answer a query by combining algorithms from both approaches, for example, using the chase on the source-to-target mappings (not the data) to produce an equivalent setting with chased mappings and without target dependencies, then using a query rewriting algorithm to answer the query over the new source-to-target mappings. In this initial design, we present our framework infrastructure, set of tools, and functionalities via a user web interface.

In our work we build upon the ForBackBench, a framework that creates a unified approach to compare several DW/VI systems. ForBackBench comes with a set of common forward- and backward-chaining systems, common testing scenarios from literature, and preintegrated converter tools to simplify the comparing and evaluation process, allow automated testing, and allow for easier extension in the future. The query-answering systems that initially became available on ForBackBench were classified in three groups: query rewriting (Rapid, Iqaros, Graal, Ontop, OntopR, GQR), chase (RDFox, Rulewerk, Llunatic), and hybrid systems (ChaseGQR). Through a command-line interface and a web interface, the user of the framework can run end-to-end experiments on combinations of query-answering systems and scenarios, build new scenarios by providing ontology or TGDs files, and generate data with different sizes.

Additionally, the command-line interface allows the user to run any converter as a stand-alone tool. The full details of ForBackBench functionalities are available in [4].

Although the backward-chaining systems included as part of ForBackBench came mostly from the Ontology-Based Data Access (OBDA) area of SW, the forward-chaining systems came from the state-of-the-art data exchange technologies in the DB domain. For our UPCAST DI Operator we are integrating in ForBackBench state-of-the-art ontology materialisation engines from the SW domain. This is bridging the gap between the DB and SW areas in DI/DE.

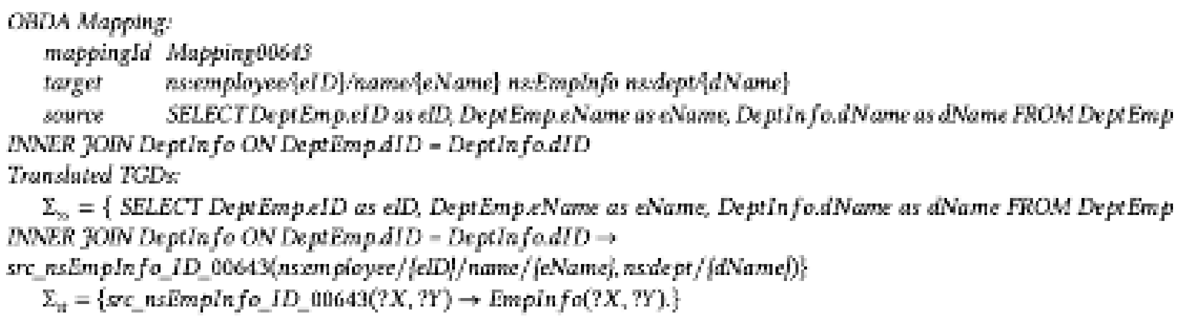
Our method includes end-to-end translation algorithms and implementation between the most common SW mapping languages (OBDA mappings, R2RML, and RML) and TGDs. We will integrat state-of-the-art ontology materialisation systems into ForBackBench such as: Morph-KGC, RMLMapper, and SDM-RDFizer. We have already published an initial round of experiments pointing out the different ways algorithms and engines are used and shedding light on the performance of these systems.

Data Model: In data integration/exchange settings there are two families of schemas: a source schema and a target/mediating schema. The mappings between the two kinds of schemas, as well as ontology axioms/constraints on top of the target schema, can be expressed via schema-mappings. In the database community these mappings are represented by database dependencies known as Tuple-Generating Dependencies (TGDs), where source-to-target TGDs express mappings between the source and the target, and target TGDs express dependencies on the target schema [6], [7]. In the semantic web community description logics, in particular, the DL-LiteR family of description logics [8] and the associated OWL2 QL profile, are used to represent the target ontology [9], while mapping languages such as R2RML [10], RML [11], [12], and OBDA-mappings [13] are used to express the mappings between the data sources and the target ontology [8].

These are standard data models and languages that we don't reproduce here. The extension we have worked is translations between mapping languages. We focus on the mapping translation between SW mappings (that use triples) and DB mappings (that use rules).

Mappings to TGDs Translation In some cases, mappings languages use "Skolem" or "built-in" functions that cannot be encoded into TGDs in a straightforward manner. For instance, the following example shows an OBDA mapping object that creates new values for the arguments of $ns{:}EmpInfo$ by combining multiple columns of the source tables. Therefore, we introduce a new sub-language of TGDs: Skolem source-to-source TGDs. A Skolem ss-TGD(Σss) is a TGD of the form $\forall$, $y(\varphi(x,y) \rightarrow \psi$ $(\mu(x)$, $\mu(y)))$ where μ can be an arbitrary user-defined function (initially we limit ourselves to string concatenation).

Example



OBDA Mapping:

mappingId *Mapping00643*

target *ns:employee/{eID}/name/{eName} ns:EmpInfo ns:dept/{dName}*

source *SELECT DeptEmp.eID as eID, DeptEmp.eName as eName, DeptInfo.dName as dName FROM DeptEmp INNER JOIN DeptInfo ON DeptEmp.dID = DeptInfo.dID*

Translated TGDs:

Σ_{ss} = { *SELECT DeptEmp.eID as eID, DeptEmp.eName as eName, DeptInfo.dName as dName FROM DeptEmp INNER JOIN DeptInfo ON DeptEmp.dID = DeptInfo.dID* $\rightarrow$ *src_nsEmpInfo_ID_00643(ns:employee/{eID}/name/{eName}, ns:dept/{dName})*}

Σ_{st} = {*src_nsEmpInfo_ID_00643(?X, ?Y)* $\rightarrow$ *EmpInfo(?X, ?Y).*}

Definition 1. For all TGDs σ of at most arity two, σ is a Skolem source-to-source TGD if (1) φ and ψ are over the source schema, (2) φ, the body, is an SQL query, (3) ψ, the head, is a ChaseBench query, where μ, the arguments, are concatenation functions of multiple columns of source tables.

For ontology materialisation scenarios we implement two phases: the offline phase, where we convert OBDA, RML, or R2RML mappings to TGDs (see the example above) and the online phase, where we generate and load data for the new source tables. The choice of supported mapping languages was guided by the current scope of our framework, which is mostly limited to relational DBs.

For each source-to-target (st) TGD mapping (we assume/-transform TGDs to have a single head atom), we define a triple map (S, P, O, ϕ) where the TGD head atom translates (via the natural way) into S, P, O and ϕ is a generated SQL query reflecting the TGD body. Every mapping language (OBDA mappings, R2RML, or RML) contains a “target” that can be constructed from S, P, O and a logical source in SQL or CSV. Thus we appropriately translate our triple map to a chosen language. We can also support translating back to a mapping language from a scenario that includes both st-TGDs as well as our own Skolem ss-TGDs (thus reverting the example), in which case the SQL query ϕ is obtained from the Skolem ss-TGDs (since it appears explicitly). If we have Skolem ss-TGDs, we use the namespaces contained in them; otherwise, we invent an example namespace for the final mapping.

Typical Usage: The typical usage for our UPCAST DI Operator will involve: (1) selecting the data sources, (2) selecting the target schema and format, (3) specifying the schema mappings, (3) specifying a query (optional for the data warehousing case), (4) selecting our own implementation or any custom algorithm to run either data warehousing (forward-chaining), or virtual integration (query rewriting/backward-chaining), (5) running and getting the data output of the integration. These will be available as API calls, integrated with the DPW editor, or our (currently under development) standalone web interface.

5.11 Negotiation and Contracting

Functionality: This plugin is designed to streamline the complex processes of negotiation and contract management. It facilitates efficient communication and collaboration between data producers and data consumers by providing a centralized platform. Users can easily initiate, conduct, and finalize negotiations, discussing usage policies, pricing and environmental impacts as the resource description features, Data Processing Workflows (DPWs), and other details.

Moreover, the plugin includes robust contract management features, allowing parties to create, review, sign, and execute contracts with ease. By automating routine tasks and offering customizable DPWs, it enhances data sharing while ensuring compliance with regulatory requirements.

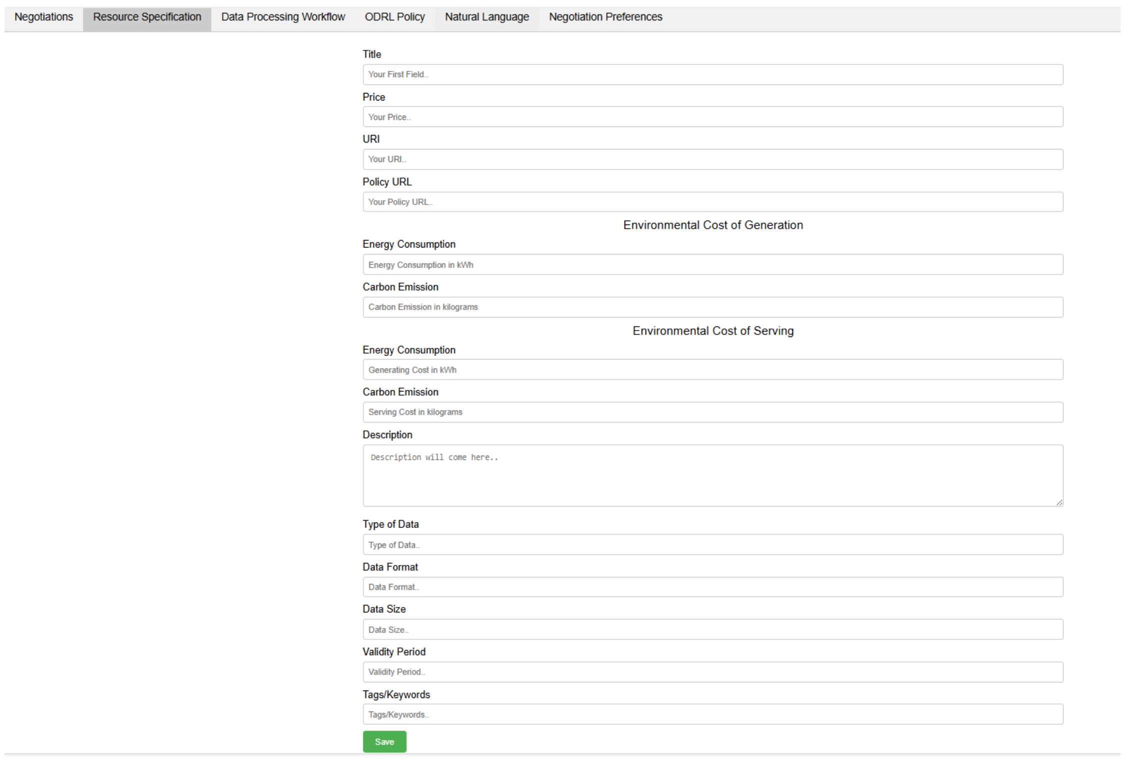
Major changes since D1.2: In D1.2 resource consumer has defined the DPW including resources owned by third parties. Applying GoodFlows, after defining a DPW and a request, the consumer can generate a request.

Additionally, we can assume a stand-alone scenario that the resource consumer searches the Data Catalogue for a resource. Finding the proper resource, the consumer fetches the resource specification and its ODRL offer from the Data Catalogue and generates a request.

Interface: The negotiation and contracting plugin within the UPCAST platform, has three key interfaces to function effectively:

1. User Interface (UI): Provides a user-friendly interface for resource producers and consumers to initiate and manage negotiations. UI Includes dashboards, forms for inputting negotiation terms, and tools for tracking progress and updates.
2. Application Programming Interface (API): Facilitates communication between resource producers and consumers, as well as contract management. Negotiation API enables automated data exchange, retrieval of resource specifications, and integration with databases.
3. Data Processing Interface: Manages the execution of Data Processing Workflows (DPWs) as per negotiated term. This interface handles data input, processing tasks, and output generation while ensuring compliance with agreed policies and procedures.

User Interface (UI) has a resource specification form and two main dashboards for resource producer and resource consumer. Via the sample resource specification form shown in Figure 48 producers can create detailed resource specifications, including metadata, privacy policies, and pricing information. This functionality is provided for the case the Negotiation plugin is used as standalone and not integrated to the UPCAST platform, in which case the provider dashboard presented in section 5.1 is used.



Title
Your First Field..

Price
Your Price..

URI
Your URI..

Policy URL
Your Policy URL..

Environmental Cost of Generation

Energy Consumption
Energy Consumption in kWh

Carbon Emission
Carbon Emission in kilograms

Environmental Cost of Serving

Energy Consumption
Generating Cost in kWh

Carbon Emission
Serving Cost in kilograms

Description
Description will come here..

Type of Data
Type of Data..

Data Format
Data Format..

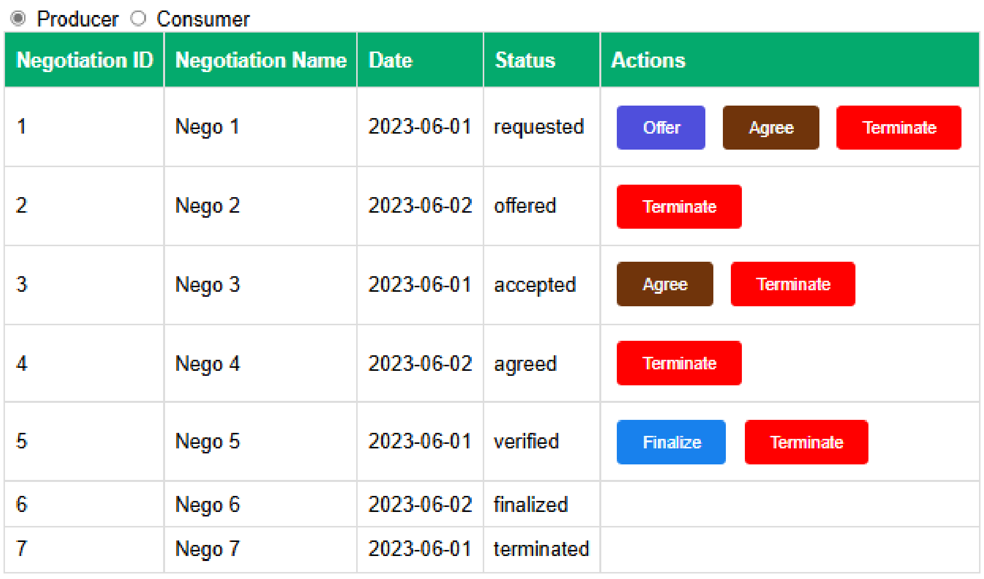
Data Size
Data Size..

Validity Period
Validity Period..

Tags/Keywords
Tags/Keywords..

Figure 48: Resource Specification.

A resource producer can manage negotiations through the producer dashboard that is shown in Figure 49. Producers can view a list of ongoing negotiations, track their status, and respond to consumer offers. The interface provides real-time updates and actions that can be taken for each negotiation.



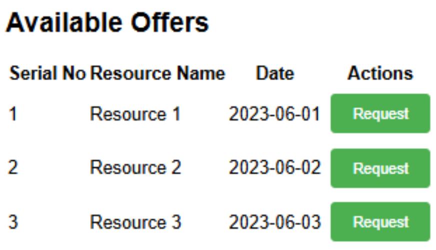
[● Producer] ○ Consumer

Negotiation ID	Negotiation Name	Date	Status	Actions
1	Nego 1	2023-06-01	requested	Offer Agree Terminate
2	Nego 2	2023-06-02	offered	Terminate
3	Nego 3	2023-06-01	accepted	Agree Terminate
4	Nego 4	2023-06-02	agreed	Terminate
5	Nego 5	2023-06-01	verified	Finalize Terminate
6	Nego 6	2023-06-02	finalized	
7	Nego 7	2023-06-01	terminated	

Figure 49: Negotiations GUI.

The producer interface displays a list of negotiations, detailing information such as negotiation ID, name, date, status, and available actions. Producers can agree, finalize, or terminate negotiations based on the current status.

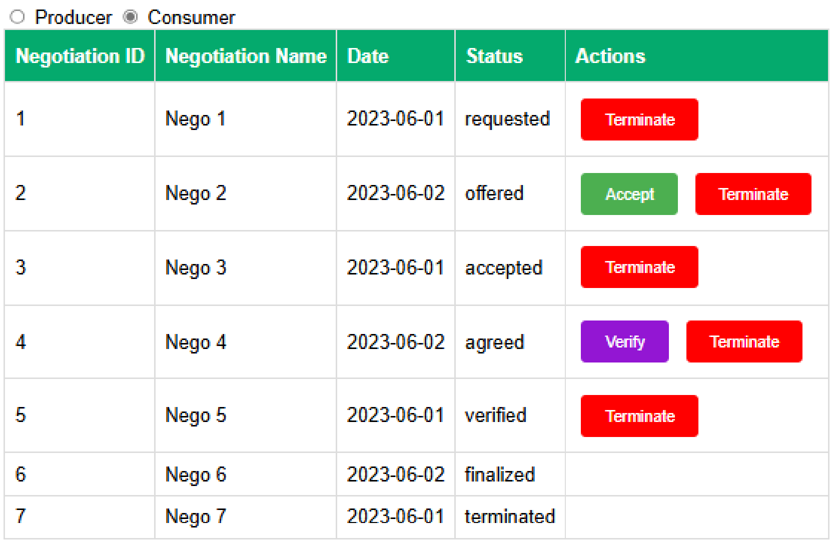
The resource consumer dashboard shown in Figure 50, and Figure 51 provides an interface for consumers to start a negotiation by sending a request based on the existing offers and is intended to be used when the negotiation plugin is used as a standalone functionality. The consumer dashboard that is integrated in the UPCAST platform includes functions for DPW modelling, discovery of datasets, and is presented in section 5.8. Consumers can also negotiate on various aspects, including price, usage terms, and data delivery conditions by updating the requests. Moreover, consumers have the option to accept a producer's offer, verify an agreement sent by a producer, or terminate a negotiation.



Available Offers

Serial No	Resource Name	Date	Actions
1	Resource 1	2023-06-01	Request
2	Resource 2	2023-06-02	Request
3	Resource 3	2023-06-03	Request

Figure 50: Consumer Dashboard for datasets available for negotiations.



○ Producer [✓ Consumer]

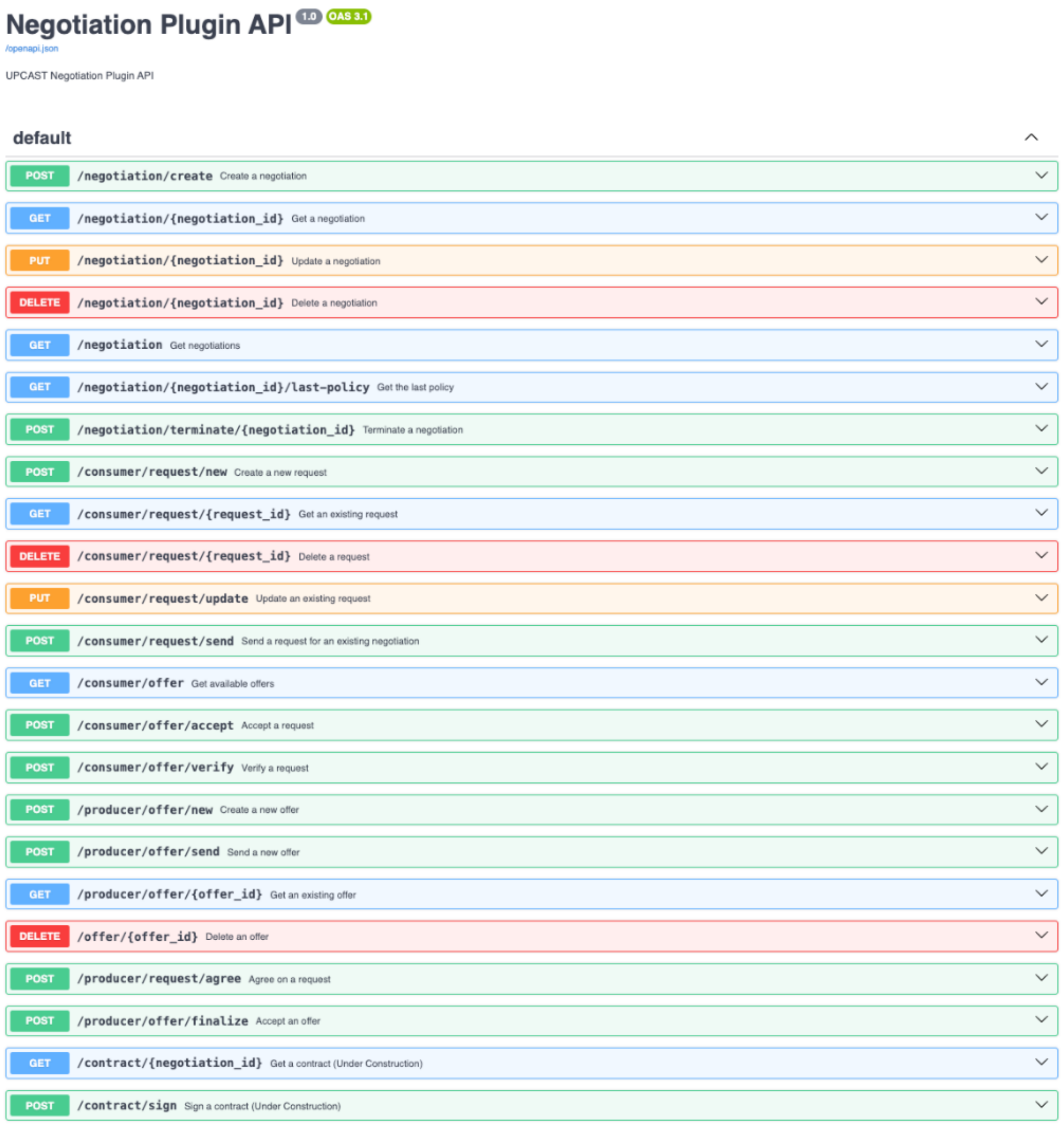
Negotiation ID	Negotiation Name	Date	Status	Actions
1	Nego 1	2023-06-01	requested	Terminate
2	Nego 2	2023-06-02	offered	Accept Terminate
3	Nego 3	2023-06-01	accepted	Terminate
4	Nego 4	2023-06-02	agreed	Verify Terminate
5	Nego 5	2023-06-01	verified	Terminate
6	Nego 6	2023-06-02	finalized	
7	Nego 7	2023-06-01	terminated	

Figure 51: Consumer Dashboard for datasets available for negotiations.

Application Programming Interface (API) provides several endpoints to facilitate the negotiation process, including but not limited to:

- Resource Management: Endpoints for creating, updating, and publishing resources.
- Negotiation Management: Endpoints for initiating, responding to, and finalizing negotiations.
- Contract Management: Endpoints for generating and managing contracts post-negotiation.

The API ensures secure and efficient data exchange, supporting JSON and other relevant data formats (Figure 52, Figure 53). Detailed API documentation and usage examples are available at UPCAST API Documentation.

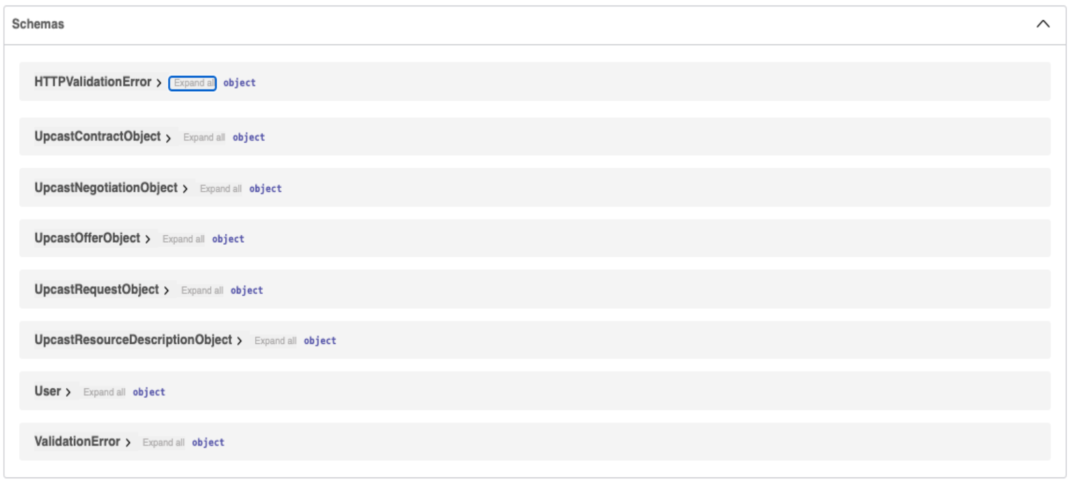


Negotiation Plugin API 1.0 OAS 3.1
/openapi.json
UPCAST Negotiation Plugin API

default

POST /negotiation/create Create a negotiation
GET /negotiation/{negotiation_id} Get a negotiation
PUT /negotiation/{negotiation_id} Update a negotiation
DELETE /negotiation/{negotiation_id} Delete a negotiation
GET /negotiation Get negotiations
GET /negotiation/{negotiation_id}/last-policy Get the last policy
POST /negotiation/terminate/{negotiation_id} Terminate a negotiation
POST /consumer/request/new Create a new request
GET /consumer/request/{request_id} Get an existing request
DELETE /consumer/request/{request_id} Delete a request
PUT /consumer/request/update Update an existing request
POST /consumer/request/send Send a request for an existing negotiation
GET /consumer/offer Get available offers
POST /consumer/offer/accept Accept a request
POST /consumer/offer/verify Verify a request
POST /producer/offer/new Create a new offer
POST /producer/offer/send Send a new offer
GET /producer/offer/{offer_id} Get an existing offer
DELETE /offer/{offer_id} Delete an offer
POST /producer/request/agree Agree on a request
POST /producer/offer/finalize Accept an offer
GET /contract/{negotiation_id} Get a contract (Under Construction)
POST /contract/sign Sign a contract (Under Construction)

Figure 52: Negotiations plugin API specification.


Schemas

HTTPValidationError > Expand all object

UpcastContractObject > Expand all object

UpcastNegotiationObject > Expand all object

UpcastOfferObject > Expand all object

UpcastRequestObject > Expand all object

UpcastResourceDescriptionObject > Expand all object

User > Expand all object

ValidationError > Expand all object

Figure 53: Negotiations plugin API specification.

Data Processing Interface provides an interface for resource consumers to initiate a request by discovering a resource and generating a DPW, as well as update DPWs during negotiations. This interface has been provided by Data Processing Workflow modelling and is depicted in Figure 40.

Data Model: The UPCAST negotiation plugin allows users to create, offer, request, and negotiate data sharing contracts. UPCAST contracts extend the usage control specification defined by International-Data-Spaces-Association (IDSA), which in turn uses the Open Digital Rights Language (ODRL), enabling more descriptive and technology-independent contracts. UPCAST contracts also utilize other ontologies such as Data Privacy Vocabulary (DPV), which defines an ontology that allows for the definition of the use, processing and purpose of processing of data under relevant legislation, notably the GDPR.

Figure 54 shows the data model of the negotiation and contracting plugin. The model is compliant to the extent possible with the model that supports the IDSA Dataspaces Negotiation Protocol comprised of a back and forth of Offers and Requests, then when accepted by both parties become an Agreement. We do add a UPCAST:Contract entity that includes the agreement, plus other details necessary for a contract: Signature, Start and end date, Natural Language part and link to a UPCAST:Data Processing Workflow.

We apply the same pattern to Request. Offers may include DPW patterns that are more general than DPWs (this is a Soton research interest, it might be easier to assume they are DPWs) We follow as much as possible the model that supports the IDSA Dataspaces Negotiation Protocol comprised of a back and forth of Offers and Requests, than when accepted by both parties become an Agreement. We do add a UPCAST:Contract entity that includes the agreement, plus other details necessary for a contract: Signature, Start and end date, Natural Language part and link to a UPCAST:Data Processing Workflow. We apply the same pattern to Request. Offers may include DPW patterns that are more general than DPWs.

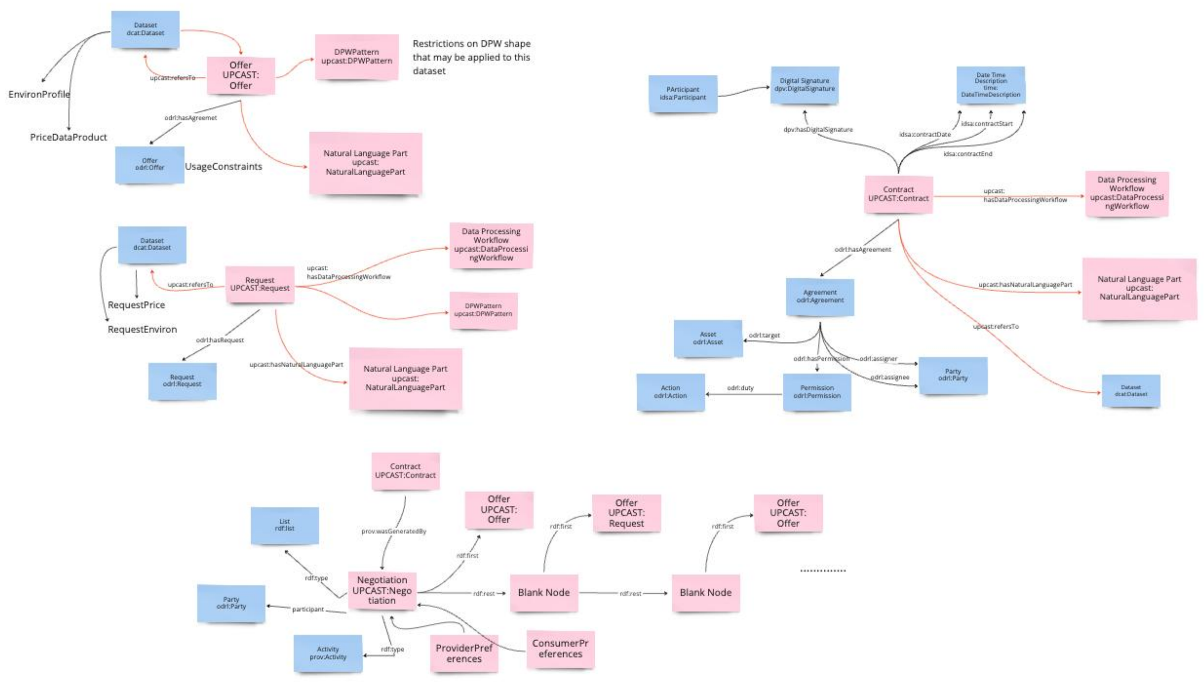


Figure 54: Negotiation plugin Data Model.

Typical use: A typical use of the Negotiation and Contracting plugin is shown in the following, with example interactions in JSON format:

1. Resource Consumer Finds a Resource: A resource consumer finds a resource interesting and sends a request to initiate a negotiation with the resource producer.

Request

```
{
"id": "string",
"name": "string",
"type": "string",
"consumer": {
"id": "string",
"name": "string"
},
"producer": {
"id": "string",
"name": "string"
},
"data_processing_workflow_object": {},
"natural_language_document": "string",
"resource_description_object": {
"id": "string",
"name": "string",
"price": 0,
"environmental_effect": "string",
"resource_description": {},
"created_at": "2024-06-12T14:51:48.998Z",
"updated_at": "2024-06-12T14:51:48.998Z"
},
"odrl_policy": {},
"negotiation_id": "string",
"created_at": "2024-06-12T14:51:48.998Z",
"updated_at": "2024-06-12T14:51:48.998Z"
}
```

Response

```
{
"id": "string",
"name": "string",
"consumer": {
"id": "string",
"name": "string"
},
"producer": {
"id": "string",
"name": "string"
},
"negotiation_status": "string",
"resource_description": {},
"dpw": {},
"nlp": "string",
"conflict_status": "string",
"negotiations": [
{}
],
"created_at": "2024-06-12T14:51:49.000Z",
"updated_at": "2024-06-12T14:51:49.000Z"
}
```

2. Exchanging Requests and Counteroffers: The resource consumer and producer exchange requests and counteroffers, respectively, until they reach an agreement or decide to terminate the negotiation.

Request

```
{
"id": "string",
"name": "string",
"type": "string",
"consumer": {
"id": "string",
"name": "string"
},
"producer": {
"id": "string",
"name": "string"
},
"data_processing_workflow_object": {},
"natural_language_document": "string",
"resource_description_object": {
"id": "string",
"name": "string",
"price": 0,
"environmental_effect": "string",
"resource_description": {},
"created_at": "2024-06-12T14:54:58.208Z",
"updated_at": "2024-06-12T14:54:58.208Z"
},
"odrl_policy": {},
"negotiation_id": "string",
"created_at": "2024-06-12T14:54:58.208Z",
"updated_at": "2024-06-12T14:54:58.208Z"
}
```

Response

```
{
"status": "request-sent",
"id": "string",
}
```

Producer Responds with a Counteroffer

Request

```
{
"id": "string",
"name": "string",
"type": "string",
"consumer": {
"id": "string",
"name": "string"
},
"producer": {
"id": "string",
"name": "string"
},
"data_processing_workflow_object": {},
"natural_language_document": "string",
"resource_description_object": {
"id": "string",
"name": "string",
"price": 0,
"environmental_effect": "string",
"resource_description": {},
"created_at": "2024-06-12T15:11:10.250Z",
"updated_at": "2024-06-12T15:11:10.251Z"
},
"odrl_policy": {},
"negotiation_id": "string",
"created_at": "2024-06-12T15:11:10.251Z",
"updated_at": "2024-06-12T15:11:10.251Z"
}
```

Response

```
{
"status": "counter-offer-sent",
"id": "string",
}
```

3. Reach an Agreement: The producer sends an agreement message in two situations: (1) receiving a request and agreeing with it; (2) sending a counteroffer and receiving an acceptance from the consumer.

4. Finalizing the Agreement: Once both parties reach an agreement, the negotiation is finalized.

Request

```
{
"action": "finalize_agreement",
"data": { "negotiation_id": "string",
}
```

Response

```
{
"status": "agreement_finalized",
"agreement": {
"data_processing_workflow_object": {},
"natural_language_document": "string",
"resource_description_object": {
"id": "string",
"name": "string",
"price": 0,
"environmental_effect": "string",
"resource_description": {},
"created_at": "2024-06-12T15:11:10.250Z",
"updated_at": "2024-06-12T15:11:10.251Z"
},
"odrl_policy": {},
"negotiation_id": "string",
"created_at": "2024-06-12T15:11:10.251Z",
"updated_at": "2024-06-12T15:11:10.251Z"
}
}
```

5. Creating and Signing the Contract: Based on the agreement, a contract is created and signed by both parties.

Request

```
{
"id": "string",
"name": "string",
"corresponding_parties": {},
"data_processing_workflow_object": {},
"natural_language_document": "string",
"resource_description_object": {
"id": "string",
"name": "string",
"price": 0,
"environmental_effect": "string",
"resource_description": {},
"created_at": "2024-06-12T15:27:56.685Z",
"updated_at": "2024-06-12T15:27:56.685Z"
},
"metadata": {},
"status": "string",
"negotiation_id": "string",
"created_at": "2024-06-12T15:27:56.685Z",
"updated_at": "2024-06-12T15:27:56.685Z"
}
```

Response

```
{
"status": "Contract_Signed",
}
```

These JSON examples illustrate the typical interactions and data exchanges that occur during the negotiation and contracting process within the plugin. This structured approach ensures that both resource consumers and producers can effectively manage their negotiations and contractual agreements.

Contract generation supported by LLM

The contract generation process within the UPCAST plugin is significantly enhanced by the integration of Large Language Models (LLMs). These advanced AI models facilitate the automatic generation of comprehensive and precise contracts based on the negotiation outcomes. By analyzing the details of the negotiation, including usage policies, pricing structures, and specific data processing requirements, the LLM can draft contracts that accurately reflect the agreed terms. This automation not only speeds up the contract creation process but also reduces the risk of human error and ensures that all legal and regulatory aspects are meticulously addressed. The LLM's ability to understand and generate natural language makes it an invaluable tool in creating clear and enforceable contracts, thereby streamlining the entire negotiation and contracting workflow within the UPCAST platform.

Additional requirements

Table 1 lists additional requirements for the Negotiation and Contracting plugin in addition to those presented in [1].

Table 1: Additional requirements for Negotiation.

Requirement ID	Description	Related Pilot Requirements / User Stories	Verification	Priority	Technical Feasibility
REQ_NE_F_15	Users should be able to state ranges of values (e.g. price or carbon consumption) they are willing to negotiate.		Synthetic/real use-cases	Should have	Feasible
REQ_NE_F_16	Users should be able to state ranges of classes in a hierarchy they are willing to accept (e.g., willing to accept a request for usage for medical research instead of general research).		Synthetic/real use-cases	Should have	Feasible

5.12 Secure Data Exchange

Data exchange includes all tasks for the secure transfer of a dataset from the provider to the consumer for executing the Data Processing Workflow. These functions are supported by the Secure Data Delivery plugin that is shown in Figure 11.

The *Safe, Traceable, and Secure Exchange of Data* functions of the plugin allow secure data delivery within secure execution environments. The capabilities of the plugin should allow to address two main use cases:

- One data provider transferring data to only one data consumer
- One data provider transferring data to multiple data consumers.

To meet the requirements for secure data exchange, the capabilities of the plugin will be the following:

- Enforce safe and secure transfer and delivery of data and resources.
- Monitor performance, execution and compliance of data transfer using the plugin.
- Practical and scalable solution, handling large volumes of data.
- Minimize energy consumption of data transfer.
- Deployable in multiple data platforms and marketplaces, compliant with Gaia-X[11] specifications.

The approach chosen by Dawex to provide these capabilities relies on principles for an open architecture with trust as its focal point, able to interact both with other UPCAST plugins, and with other existing data connectors:

- Allow data provider to perform data exchange with a trusted *Safe data product Transfer Plugin*,
- Interconnect distributed data connectors to perform data exchange under the supervision of a data space orchestrator.
- Enable organizations to design and manage data products from multiple data source in their own environment,
- Facilitate deployment of the plugin with cloud agnostic and low consumption solution,
- Provide advanced tracing and telemetry metrics to analyse performance, execution and compliance of Data Transfer.

5.12.1 Data exchange scenarios

This section gives an overview of different data exchange scenarios, namely, one-to-one and one-to-many.

One to one data exchange

In this scenario, depicted in Figure 55, the *Secure Data Delivery* Plugin is working as a request/response proxy to exchange data between a data provider and a data consumer. Once terms are negotiated and a contract is established, the data consumer will be allowed to request data product transfer securely to a trusted data destination or consume the data product from within application execution directly.

[11] https://gaia-x.eu/

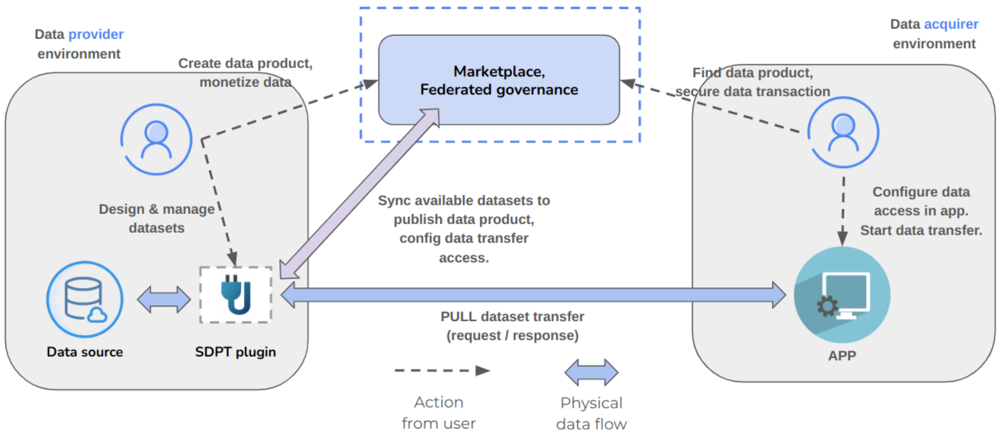

Data provider environment
Create data product, monetize data
Marketplace, Federated governance
Data acquirer environment
Find data product, secure data transaction
Design & manage datasets
Sync available datasets to publish data product, config data transfer access.
Configure data access in app. Start data transfer.
Data source
SDPT plugin
PULL dataset transfer (request / response)
APP
Action from user
Physical data flow

Figure 55: One to one data exchange scenario.

One to many data exchange

In this scenario, depicted in Figure 56, the Secure Data Delivery plugin functions as a broadcast proxy to exchange data from a data provider to multiple data consumers. Based on a data product delivered by the data provider, the plugin will automatically transfer this data product to all active subscribers based on their access and usage rights that are specified in the negotiated contract. The data consumer will negotiate a data contract before subscribing to the data product delivery process.

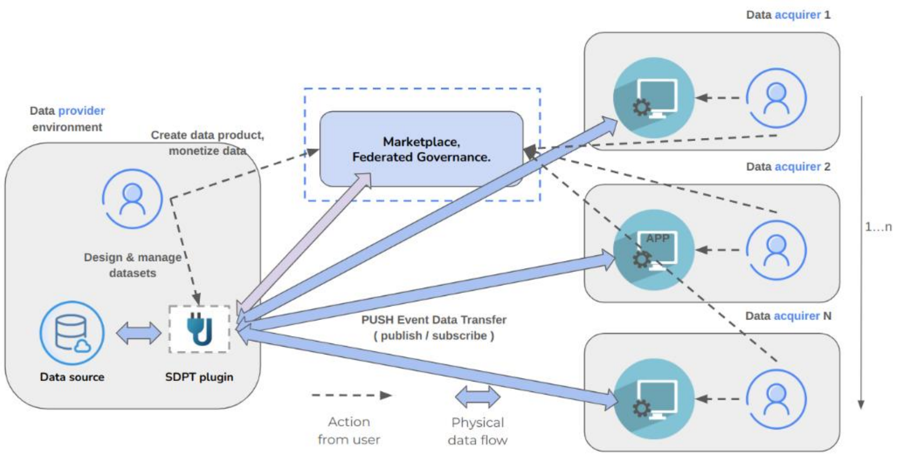

Data acquirer 1
Data provider environment
Create data product, monetize data
Marketplace, Federated Governance.
Data acquirer 2
Design & manage datasets
APP
1...n
PUSH Event Data Transfer (publish / subscribe)
Data acquirer N
Data source
SDPT plugin
Action from user
Physical data flow

Figure 56: One to many data exchange scenario.

5.12.2 Capabilities

The Secure Data Delivery plugin capabilities are shown in Table 2.

Table 2: Dawex Secure Data Delivery capabilities.

Monetization	**Data Product Exchange Management**	**Mediation routing**
• Data product listing • Realtime data transfer consumption	• Data product versioning and deprecation strategy • Healthcheck monitoring	• Data transfer routing • Policy and restriction enforcement

Integration	Observability	Security
• Configure and manage Data Source • Support standard transfer protocol • Provide access to technical documentation	• Aggregate logs and metrics • Reliability, availability, performance • Alert triggering • Traffic analysis to detect suspicious activity	• Restrict data product access • Enable authentication standards • Automatically refresh authentication token • Key access management

5.12.3 Architecture

The Secure Data Delivery plugin architecture approach is driven by the capabilities presented above and is focused to meet:

Performance and Consumption

- Microservices architecture to improve scalability and reliability,
- Minimal footprint of microservices consumption (memory, CPU),
- Enable access to advanced tracing and telemetry metrics without affecting data transfer performance.

Interoperability

- Allow interconnection with UPCAST plugins by API or Kafka client,
- Allow synchronization with Data Marketplaces by API,
- Provide public documentation for technical integration.

Figure 57 shows the backend services architecture of the Secure Data Delivery plugin. Backend services are based on multiple micro services: Manager API, Metrics API, Gateway API. Each service is developed following domain driven architecture to isolate responsibilities during Data Transfer.

Manager API: Manages Data Source and data product configuration. The Manager will match standard security protocols to access Data Source and provide an authentication process for data product accessibility. It should also provide an interface for data product listing.

Metrics API: This service is dedicated to observability functions. Metrics are collected with the Metrics Collector and stored in a Time Series Database. The service exposes API and Kafka client to allow metrics requesting such as: execution time, transfer status, latency distinguished by contract, consumer, data product.

Gateway API: The Gateway isolates an API dedicated to Data Transfer routing, and requests routing enforcement, which is essential for respecting Data Policies and protecting data product access. Isolation focuses also on performance issues as Data Transfer latency should not be impacted by tasks from another domain. This service is highly scalable to handle large volumes of data.

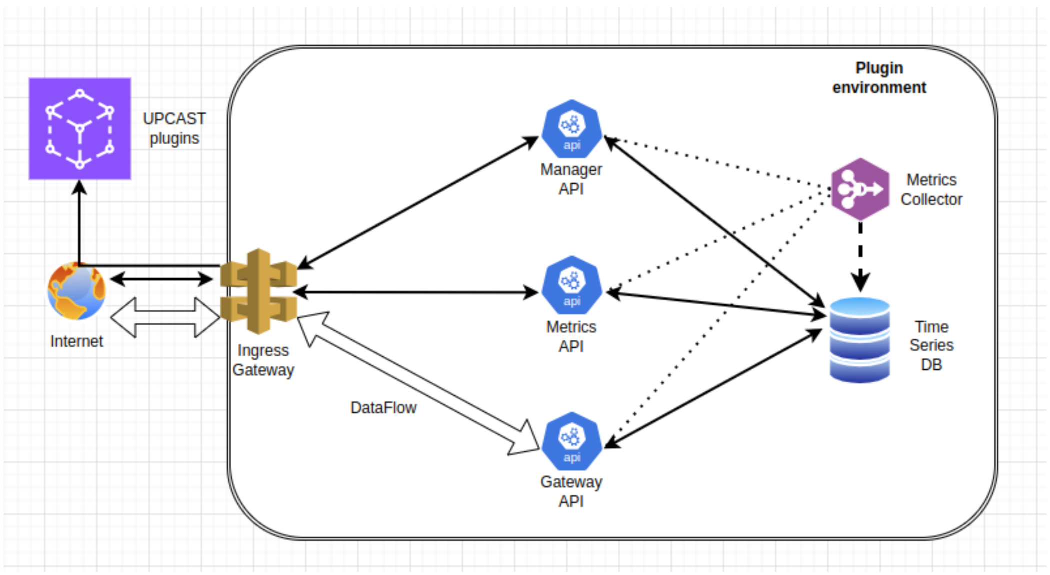


Figure 57: Safe data delivery Plugin Architecture.

Frontend: The plugin could expose a web interface to display management functions and metrics visualization. Based on the plugin capabilities, each user can create its own frontend, serving its needs. Developing a frontend interface for this plugin should be considered within the scope of the monitoring plugin as an option.

The benefits of the plugin Architecture are

- Horizontal and vertical scalability with microservices,
- Storage oriented for telemetry analytics,
- Easy integration with tiers party applications (such as other UPCAST plugins),
- Exposed API for efficient and secure interconnection,
- Secure by design: single public entry point with the ingress,
- Decorelate frontend rendering for improved performance,
- Simple and reliable for fast learning, easy maintenance and low resource consumption,
- Virtualizable (docker) for cloud agnostic deployment.

5.12.4 Plugin interoperability

As detailed in the previous section, the Secure Data Delivery plugin will expose APIs to access resources, allowing easy integration with other plugins, remotely or locally. Additional plugins can also be integrated inside the *Secure Data Delivery Plugin* architecture.

To integrate with the monitoring plugin, the Secure Data Transfer plugin will send Kafka messages and respect a strong naming convention as detailed in section 5.13.

5.13 Monitoring

This section presents the monitoring functions of the UPCAST platform. UPCAST monitoring has two parts: (a) Execution (or runtime) monitoring that is used by the dataset provider to monitor the execution of a dataset by the consumer following the agreed workflow and contract between the two, and (b) monitoring of the plugins, which is used to maintain a log of all actions taken during dataset preparation, annotation and negotiation. The following subsections present the monitoring functions of UPCAST.

5.13.1 Execution Monitoring

The UPCAST platform includes functionalities for monitoring the execution of datasets for maintaining a record of the execution for the dataset provider and also checking the compliance of the execution with the agreed workflow. The monitoring process is triggered by the start of the dataset execution and lasts until its completion. During the process, monitoring events are collected from the pilots' execution environment, they are logged, analyzed and presented to the dataset provider.

The streaming of monitoring events is implemented with standard streaming platforms. In UPCAST Apache Kafka[12] will be used for this purpose. Apache Kafka is a widely used, distributed, highly scalable, elastic, fault-tolerant, and secure event streaming platform that implements the publish-subscribe model for exchanging messages between publishers of messages (dataset consumers in the case of UPCAST) and subscribers to messages (dataset provider in the case of UPCAST).

Messages are communicated through abstractions that are called topics, which are collections of messages. Each topic has a name that is unique across the entire Kafka cluster. Messages are sent to and read from specific topics. Publishers of messages (UPCAST dataset consumers) write monitoring events to a topic, whereas subscribers of messages (UPCAST dataset providers) read those events from a topic. A given topic may have several publishers and several subscribers. Whereas publishers can publish messages to at topic concurrently, the way messages are consumer by parallel consumers depends on the existence of consumer groups, which are collections of consumers. All consumers in a consumer group share the messages of a topic, whereas consumers in different consumer groups consume the same data. This model is very versatile and allows for a multitude of patterns for the consumption and processing of the streamed messages,

Topics are organized into partitions, which can be processed in parallel by multiple consumers in a consumer group. Kafka guarantees sequencing of messages only within a partition and not across partitions. This guarantee has strong implications on the structure of topics in partitions, and dents on the application requirements. In the case of UPCAST exactly because sequencing of messages is a strong requirement as it is important for the subsequent compliance process, each topic must be organized as a single partition.

Topics have names, which are unique for the Kafka cluster. Since in UPCAST several dataset executions may take place, a strong naming convention for the names of topics must be implemented. Therefore, topics will be named as

UPCAST-<contract-id>-<execution id>

where

1. *contract-id* is the unique contract identifier under which this execution takes place. The contract with contract-id identifies the producer, the consumer, the dataset and the workflow that will be executed on it.
2. *execution id* is the unique identifier of the execution for the particular dataset, assuming that each dataset consumer assigns a unique id to each such execution for a given contract.

The monitoring events are JSON objects and have the following structure

```
{
  source: [source component, type: string]
  timestamp: [timestamp of event at source, type: datetime]
  metric: [name of the metric monitored, type: string]
```

[12] https://kafka.apache.org/

```
  value: [value of the metric, type: string]
  result: [result of the metric, type: object]
  log: [log string, type: string]
}
```

The semantics of the JSON fields are as follows:

- source: the name of the source component that emits the JSON object. Each component that implements the execution flow has a unique name. The name of the source is used mainly for statistical purposes.
- timestamp: the timestamp of the creation of the JSON object.
- metric: the name of the metric that is reported, e.g. action-start (below).
- value: the value of the metric that is reported, e.g., the name of the action that is started.
- result: any result the metric may have produced. Results are application specific objects that are produced as a result of the completion of an action. Typically they are integer values with 0 indicating normal completion of execution and non-zero indicating completion that resulted in an error. The result of an action may be used for making decisions for following different branches of the workflow or handling errors that may have resulted from the execution of an action. The result filed has meaning for action-end metrics, for the rest its value is None.
- log: log message that contains details of the monitored metric.

Topics are discovered by the monitoring plugin, by continuously polling the Kafka cluster. When a new topic is detected the monitoring plugin creates a new Kafka consumer to read messages from this topic.

Messages that are read by the Kafka consumers are sent to the compliance plugin to check compliance of the execution and are also sent to the UPCAST provider dashboard for monitoring the progress of the execution. When dataset processing ends, a message is sent to the dashboard, to update the state of the execution for running to terminated, and the Kafka consumer terminates. The Kafka topic persists after termination of the execution for providing a record of it for further analysis.

Monitoring Metrics

This section presents the monitoring metrics that will be used in UPCAST. Monitoring metrics are the entities that are emitted by an UPCAST consumer during dataset execution and are streamed to the UPCAST provider. The monitoring metrics are classified in two categories, management and execution.

The management monitoring metrics are the following:

Name of metric	**Meaning**	**Value**	**Emission instance**
start	Start of processing	None	Before start of dataset processing
stop	End of processing	None	After completion of dataset processing
suspend	Suspension of processing	None	After dataset processing suspension

resume	Resumption of processing	None	Before dataset processing resumption

The action monitoring metrics are the following:

Name of metric	Meaning	Value	Emission instance
action-start	Start of processing action	Name of action	Before start of dataset processing action, e.g., join with another dataset, calculation of statistics for an attribute, etc.
action-stop	Completion of processing action	Name of action	After completion of dataset processing action

5.13.2 Plugin Monitoring

This section presents the UPCAST functions for monitoring the execution of the plugins that are used by the dataset provider before any execution of the dataset, i.e., during the preparation and annotation of a dataset, its advertisement and the negotiation between the dataset provider and the dataset consumer. The purpose of the plugin monitoring is to keep a record of all actions that take place before dataset execution that are supported by the UPCAST plugins.

UPCAST plugins are required to generate plugin monitoring event to a Kafka topic UPCAST-plugin. The plugin monitoring events are JSON objects and have the following structure

```
{
  source: [source component, type: string]
  timestamp: [timestamp of event at source, type: datetime]
}
```

Each plugin emits different information for monitoring as shown below.

Negotiation: At each iteration it emits

```
  nid: string
  action: string
  result: object
```

nid is the negotiation id, action is the negotiation action and result is the result of the negotiation action.

Pricing: at the return of its invocation it emits

```
  did: string
  range: tuple
  explanations: object
```

did is the unique dataset id, range is the suggested price range of the dataset and explanations is a representation of the explanation for the suggested price range.

Environmental: at the return of its invocation it emits

```
  did: string
```

```
provider_id: string
consumer_id: string
exec_env_id: string
env_profile: object
```

did is the unique dataset id, provider_id is the id pf the provider, consumer_id is the id of the consumer, exec_env_id is the id of the consumer execution environment, and env_profile is the resulting environmental profile.

Usage policies: at the return of its invocation it emits

```
did: string
policy_id: string
```

did is the unique dataset id and policy_id is the id of the usage and access policy.

Publishing: at the return of its invocation it emits

```
did: string
marketplace_id: string
update: object
```

did is the unique dataset id and marketplace_id is the id of the marketplace. Update is an object that represents any updates that have been made for this dataset, e.g., new suggested price.

5.14 Compliance

The purpose of the compliance plugin is to check if execution of the workflow complies with the terms of the contract that has been agreed between a provider and a consumer. The Compliance plugin is configured to monitor a number of metrics that have been agreed between a provider and a consumer as shown in Figure 11 along with constraints that may have to be respected. During execution of the workflow, the monitoring plugin receives execution monitoring events that reflect the progress of the execution and contain values of metrics that must be verified for compliance. If any deviation from the agreed workflow execution is detected the compliance plugin raises an alert to the dataset provider to signal the violation.

5.15 Safe and Secure Execution

This section gives an overview of the execution environments that may be used in UPCAST for the execution of a Data Processing Workflow. It first presents the execution environments of the project pilots and then gives an overview of the SIMPIPE execution environment which is an alternative execution environment that can be used in UPCAST.

First the details of the execution environments that may be used in UPCAST for executing their DPWs are given. UPCAST pilots use their own infrastructure and execution environments for processing datasets. The resulting diversity of execution infrastructures is seamlessly integrated to the overall UPCAST architecture and is represented in the UPCAST Architecture of Figure 11 as a single component. Moreover, the UPCAST Architecture remains open for the choice of the execution environment that may be used for processing the DPW. SIMPIPE, an alternative execution environment presented in section 5.15.6, may also be seamlessly integrated in the UPCAST architecture. For all pilots, workflow modelling is done with the use of the Data Processing Workflow Modelling plugin (cf. Section 5.8), which is based on ICT abovo goodFlows modeler, while data exchange is performed with the Dawex marketplace platform.

5.15.1 Biomedical and Genomic Data Sharing

The execution environment of NHRF is designed for biomedical and Next-Generation Sequencing (NGS) data analysis. It incorporates Nextflow, a bioinformatics-specialized workflow engine for workflow orchestration and the AWS Batch service for compute resource management. Nextflow coordinates various bioinformatics tools essential for processing NGS data, ensuring reproducibility and scalability of workflows. AWS Batch manages the dynamic provisioning of computational resources, including EC2 instances tailored for high-throughput computing tasks. The system thus handles large input datasets and reference files, typical in NGS analyses, by utilizing AWS S3 for storage and transfer, ensuring data accessibility and durability. This architecture supports the efficient execution of complex bioinformatics workflows, facilitating comprehensive analysis of extensive genomic datasets while maintaining cost-effectiveness and scalability within the cloud infrastructure. In case of less computationally intensive tasks, Nextflow can be easily redirected to utilize the local infrastructure within NHRF premises or academic cloud infrastructure.

5.15.2 Public Administration

In the Public Administration Pilot, MDAT will be able to act both as resource provider and resource consumer. As resource provider, MDAT will support mainly the sharing of public administration datasets and presumably their analysis results with other public entities, research institutions, local NGOs and citizens to drive and collaborate to the data driven environmental policy making of the Municipality of Thessaloniki. As resource consumer, MDAT aims to find additional datasets from public administration organizations and scientific institutions to enrich and enhance the analysis possibilities of the respective data, performing integration scenarios of relevant data and expanding the number of stakeholders within the platform. The user journey utilising UPCAST plugins described in the previous section aligns generally well with the workflows of the Public Administration Pilot. A key difference is that there is no intention of using the Pricing Plugin, as the open data platform developed under the Public Administration pilot will support the open and free publishing and sharing of data.

In the public administration pilot of MDAT, public organizations, corporations, and citizens can provide their datasets under an open license. These datasets are offered for exchange through an open-source data marketplace being developed for Thessaloniki's regional authorities. Given the open nature of the datasets, execution is not typically monitored, and providers are not involved in the processing workflow. However, third parties might want to offer Nextflow execution services to support scenarios involving calculations with open datasets sourced from various providers.

A special case that might require an execution environment is the Hellenic Statistical Authority. This public organization restricts access to available datasets due to the potential inclusion of personal identification information. When a researcher requests a dataset containing only anonymized information, the statistical authority must first anonymize and then materialize the federated data into a dataset using the authority's processing resources.

Additionally, a more general requirement for execution environment support is for the marketplace itself to provide sample workflows for consumers. These workflows would include the required datasets and processing pipeline scripts as a bundle, promoting engagement with the platform.

5.15.3 Health and Fitness

The deployment and execution of the Nissatech pilot is based on the infrastructure used in the commercial system Zona Zdravlja[13]. The data is collected from wearable devices used by

[13] https://www.zonazdravlja.com

trainees and stored in a MongoDB[14] database. Various types of reports can be generated, providing added value for the fitness coaches who are monitoring the physical activity process of a particular trainee. The collected data will be shared in the Data Marketplace and valuated by the Data Valuation plugin, providing information about the preferences for data from specific types of physical activities. Data will not be shared by individual trainees, but by the Zona Zdravlja Platform provider. Individual trainees will be informed about the data valuation process and encouraged to generate data with higher valuation.

Dawex will be used for sharing the data. Types and numbers of trainees depend on the characteristics of the Dawex data sharing mechanism.

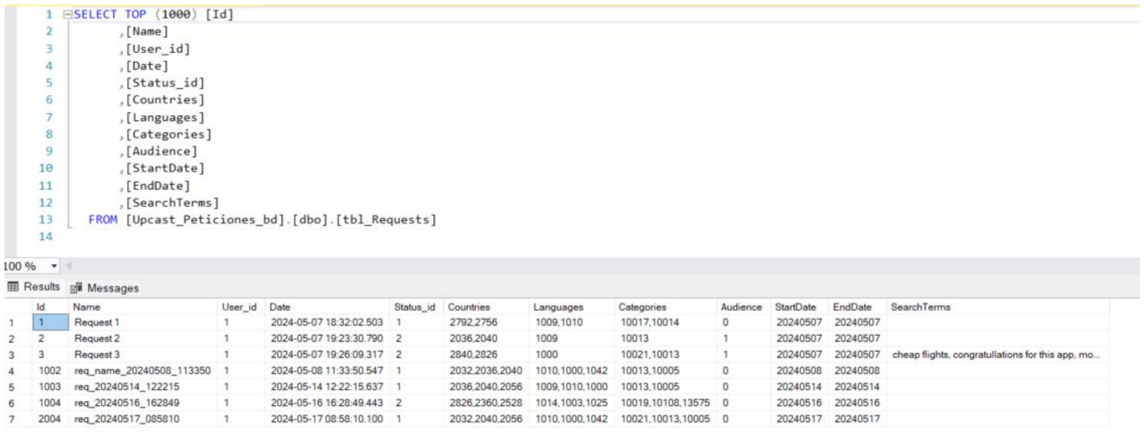
Quantity (how much can be published) depends on the characteristics of the Data Marketplace.

5.15.4 Digital Marketing 1 (JOT)

The deployment and execution for the generation of the marketing data monetisation model of JOT is based on Google Cloud infrastructure. Fully managed by Compute Engine, the pilot is hosted in a Virtual Machine (e2-custom-2-6144 type with x86-64 architecture) with a public IP and SQL Server Express installed to manage the user requests.

As presented in [1] flow orchestration implies the generation of the user request thanks to the development of .NET Blazor Server web app and the publication by means of Microsoft Internet Information Services (IIS).

For this purpose, the generation of the use requested data set is enabled by a ODBC connection to BigQuery and embedded through a sequential storage procedure. First, the data sample containing 100 initial rows, data model and its related information (metadata, description and so on) is obtained.



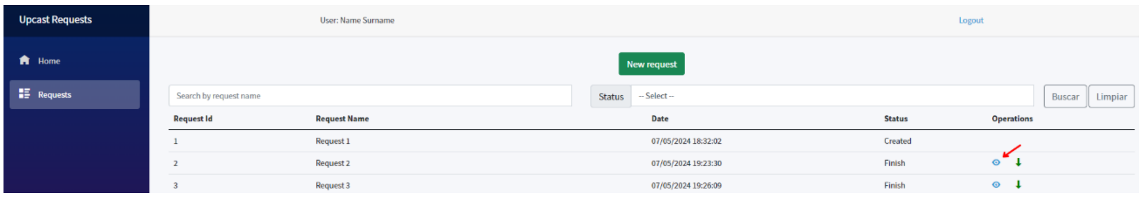
```sql
SELECT TOP (1000) [Id]
      ,[Name]
      ,[User_id]
      ,[Date]
      ,[Status_id]
      ,[Countries]
      ,[Languages]
      ,[Categories]
      ,[Audience]
      ,[StartDate]
      ,[EndDate]
      ,[SearchTerms]
  FROM [Upcast_Peticiones_bd].[dbo].[tbl_Requests]
```

	Id	Name	User_id	Date	Status_id	Countries	Languages	Categories	Audience	StartDate	EndDate	SearchTerms
1	1	Request 1	1	2024-05-07 18:32:02.503	1	2792,2756	1009,1010	10017,10014	0	20240507	20240507	
2	2	Request 2	1	2024-05-07 19:23:30.790	2	2036,2040	1009	10013	1	20240507	20240507	
3	3	Request 3	1	2024-05-07 19:26:09.317	2	2840,2826	1000	10021,10013	1	20240507	20240507	cheap flights, congratullations for this app, mo...
4	1002	req_name_20240508_113350	1	2024-05-08 11:33:50.547	1	2032,2036,2040	1010,1000,1042	10013,10005	0	20240508	20240508	
5	1003	req_20240514_122215	1	2024-05-14 12:22:15.637	1	2036,2040,2056	1009,1010,1000	10013,10005	0	20240514	20240514	
6	1004	req_20240516_162849	1	2024-05-16 16:28:49.443	2	2826,2360,2528	1014,1003,1025	10019,10108,13575	0	20240516	20240516	
7	2004	req_20240517_085810	1	2024-05-17 08:58:10.100	1	2032,2040,2056	1010,1000,1042	10021,10013,10005	0	20240517	20240517	

Figure 58: Table containing the user request for data set generation.

Then, the following steps are sequentially orchestrated to create the final offer to the user. These involve both the pricing and the final negotiation and agreement. Finally, when the agreement is signed, the full data set will be generated and shared with the user with a link to a Google Cloud Storage bucket (as indicated in Figure 59 – red arrow) and related reporting services are executed.

[14] https://www.mongodb.com/



Request Id	Request Name	Date	Status	Operations
1	Request 1	07/05/2024 18:32:02	Created	
2	Request 2	07/05/2024 19:23:30	Finish	👁 ⬇
3	Request 3	07/05/2024 19:26:09	Finish	👁 ⬇

Figure 59: User interface showing the status of the user request.

5.15.5 Digital Marketing 2 (CACTUS)

The deployment and execution of the marketing data monetization model for CACTUS will be managed as follows. Google Cloud APIs will be utilized to gather all necessary data, ensuring a comprehensive and efficient data collection process. This data will then be imported into the CACTUS custom-made CRM, designed to meet CACTUS specific needs and enhance data management capabilities. The CACTUS CRM is hosted on a MySQL Server provided by Digital Ocean[15], leveraging its robust infrastructure for reliable performance and scalability. This approach not only streamlines data collection and management but also integrates seamlessly with existing systems, thereby optimizing the overall data monetization strategy.

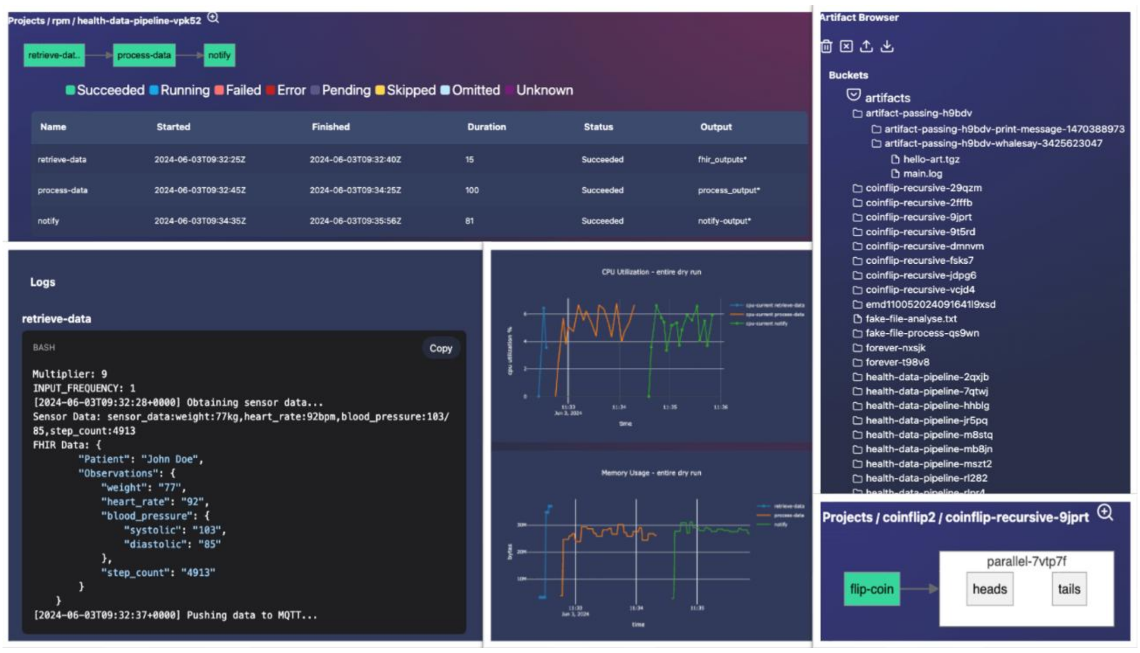
5.15.6 SIMPIPE Execution Environment

SIMPIPE is an open source software[16], developed and maintained by SINTEF. SIMPIPE provides an accurate and secure sandbox environment for execution of pipelines using Argo Workflows for orchestrating data pipeline jobs on a Kubernetes cluster. SIMPIPE runs a preconfigured local Kubernetes cluster that runs the containerized steps of the workflow in pods. Kubernetes is an opensource platform designed for managing automated deployment and scaling of containerized applications, securing high resiliency by handling of complex tasks such as automatic failover, horizontal and vertical scaling and continuous deployment.

The execution of a workflow pipeline in SIMPIPE consists of two stages. In the first stage a project is defined by a project name and an Argo Workflow YAML file. The YAML file describes the steps of the pipeline, which can conveniently also be a directed acyclic graph (DAG). This enables complex workflows containing for example while loops and pipeline dependencies such as the outcome of a previous step. In the second stage a dry run is defined by setting input parameters for the steps of the pipeline. Submitting the dry run will schedule the execution of the workflow. In the graphical user interface (GUI) of SIMPIPE the user can monitor the status and progress of the pipeline along with output logs and resource metrics of the entire pipeline and for individual steps.

[15] https://www.digitalocean.com

[16] https://github.com/DataCloud-project/SIM-PIPE


Projects / rpm / health-data-pipeline-vpk52
retrieve-dat..
process-data
notify
🟩 Succeeded 🟦 Running 🟥 Failed 🟥 Error ▨ Pending 🟨 Skipped 🟦 Omitted 🟪 Unknown
CPU Utilization - entire dry run
Memory Usage - entire dry run
time
bytes
Projects / coinflip2 / coinflip-recursive-9jprt
flip-coin
parallel-7vtp7f
heads
tails

Name	Started	Finished	Duration	Status	Output
retrieve-data	2024-06-03T09:32:25Z	2024-06-03T09:32:40Z	15	Succeeded	fhir_outputs*
process-data	2024-06-03T09:32:45Z	2024-06-03T09:34:25Z	100	Succeeded	process_output*
notify	2024-06-03T09:34:35Z	2024-06-03T09:35:56Z	81	Succeeded	notify-output*

Logs

retrieve-data

```
Multiplier: 9
INPUT_FREQUENCY: 1
[2024-06-03T09:32:28+0000] Obtaining sensor data...
Sensor Data: sensor_data:weight:77kg,heart_rate:92bpm,blood_pressure:103/
85,step_count:4913
FHIR Data: {
        "Patient": "John Doe",
        "Observations": {
            "weight": "77",
            "heart_rate": "92",
            "blood_pressure": {
                "systolic": "103",
                "diastolic": "85"
            },
            "step_count": "4913"
        }
    }
[2024-06-03T09:32:37+0000] Pushing data to MQTT...
```

Artifact Browser

Buckets

```
artifacts
  artifact-passing-h9bdv
    artifact-passing-h9bdv-print-message-1470388973
    artifact-passing-h9bdv-whalesay-3425623047
      hello-art.tgz
      main.log
  coinflip-recursive-29qzm
  coinflip-recursive-2fffb
  coinflip-recursive-9jprt
  coinflip-recursive-9t5rd
  coinflip-recursive-dmnvm
  coinflip-recursive-fsks7
  coinflip-recursive-jdpg6
  coinflip-recursive-vcjd4
  emd110052024091641l9xsd
  fake-file-analyse.txt
  fake-file-process-qs9wn
  forever-nxsjk
  forever-t98v8
  health-data-pipeline-2qxjb
  health-data-pipeline-7qtwj
  health-data-pipeline-hhblg
  health-data-pipeline-jr5pq
  health-data-pipeline-m8stq
  health-data-pipeline-mb8jn
  health-data-pipeline-mszt2
  health-data-pipeline-rl282
```

Figure 60: Frontend interface elements of SIMPIPE.

Figure 60 shows screenshots of some graphical user interface elements from SIMPIPE. The upper left panel shows pipeline step information. Lower left panels show logs and metrics graphs respectively. Upper right panel show artefact browser for uploaded and produced artefacts. Lower right panel show pipeline steps for a parallel loop

Clicking on the magnifying glass sign brings the user to a page with many different metrics graphs. Logs are scrollable and can be downloaded. Logs and outputs can be found and downloaded also from the artefact browser. In the artefact browser, the user can create new buckets and upload files. The artefacts are stored in a local Minio instance, and can be browsed and referenced as input files for new pipeline dry runs. Several dry runs of a pipeline can be selected to make predictions for the scaling of the pipeline.

A containerized step in the workflow pipeline requires an image to be run. Local and public images can be referenced as well as private image repositories. This is important in order to use custom built images for the different steps of a pipeline.

SIMPIPE is a powerful tool for executing data pipelines in a safe and secure sandbox environment, enabled by the integration of widely used opensource software for running and managing data pipelines in cloud environments.

6 UPCAST Pilot Descriptions

This section gives a detailed description and update on the UPCAST pilots.

6.1 Biomedical and Genomic Data Sharing

6.1.1 Requirements and technical challenges

Genomic and clinical data sharing raises challenges mainly due to the sensitive and heterogeneous nature of such data. Protecting data privacy and ensuring appropriate informed consent is obtained by clinical partners are critical ethical and legal considerations that will be addressed by UPCAST plugins. In addition, complying with data protection standards, implementing encryption, and enforcing secure data transfer and storage are critical challenges concerning genomic and biomedical data processing. Finally, processed genomic data are highly heterogeneous as a result of different analytical pipelines and annotations applied. For this reason, integration of data from different resources is technically challenging and requires representation of data with community-accepted standards to enable interoperability with external systems or databases.

6.1.2 Data Processing workflows

NHRF acts both as resource provider and resource consumer. As resource provider NHRF aims to share analysis results with other research groups and exploit them as the basis of a new collaboration. As resource consumer, NHRF seeks to acquire data from research partners and data repositories to perform integrative analysis on cancer genomics. The user journey utilising UPCAST plugins described in the previous section, aligns well with the workflows of NHRF. The difference that should be noted concerns the Pricing plugin, since NHRF currently does not monetise any datasets. This could be possibly applied only to datasets derived from *in-vitro* experiments and not to datasets orienting from patient data.

NHRF implements two bioinformatics workflows:

1. **Cancer Genomics:** Analysis of Next Generation Sequencing (NGS) data from cancer tissue of patients together with the relevant clinical data for the derivation of a curated list of variants including variants with clinical interest. This workflow necessitates collaboration with clinical partners to acquire clinical and genomic data. Genomic data are either provided by the clinical/research partner or generated by NGS performed at NHRF Genomic Unit from the provided biomaterial (Figure 61).
2. ***In-vitro* gene expression**: Analysis of NGS data from *in-vitro* experiments performed in NHRF for the derivation of a list of genes with differential expression. These experiments concern the analysis of alterations in gene expression after treatment of cancer cell lines with bioactive molecules that could represent candidate drugs (Figure 62).

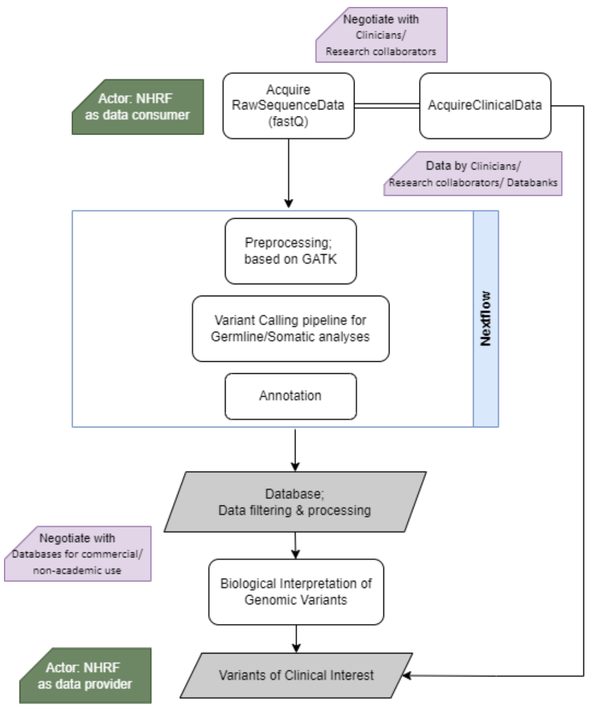

Negotiate with Clinicians/ Research collaborators
Actor: NHRF as data consumer
Acquire RawSequenceData (fastQ)
AcquireClinicalData
Data by Clinicians/ Research collaborators/ Databanks
Preprocessing; based on GATK
Variant Calling pipeline for Germline/Somatic analyses
Annotation
Nextflow
Database; Data filtering & processing
Negotiate with Databases for commercial/ non-academic use
Biological Interpretation of Genomic Variants
Actor: NHRF as data provider
Variants of Clinical Interest

Figure 61: Cancer Genomics workflow

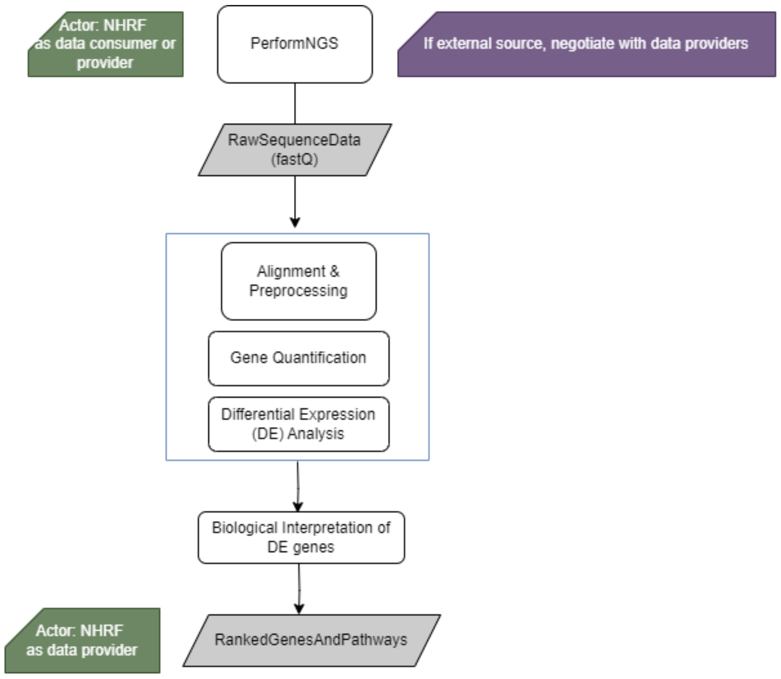
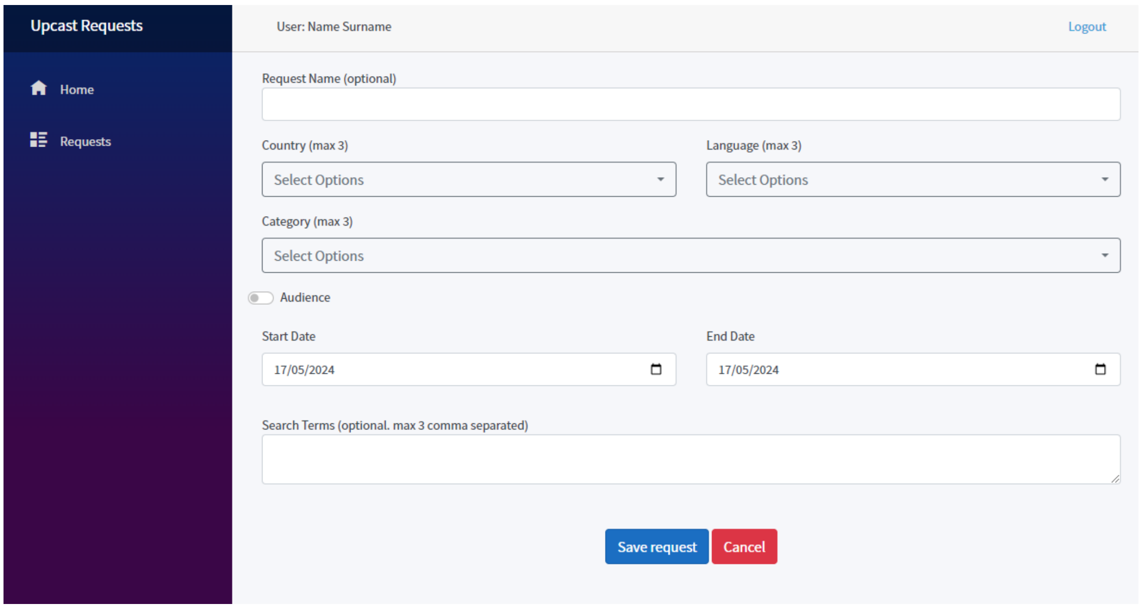

Actor: NHRF as data consumer or provider
PerformNGS
If external source, negotiate with data providers
RawSequenceData (fastQ)
Alignment & Preprocessing
Gene Quantification
Differential Expression (DE) Analysis
Biological Interpretation of DE genes
Actor: NHRF as data provider
RankedGenesAndPathways

Figure 62: "In-vitro gene expression" workflow.

6.2 Digital Marketing Data and Resources

6.2.1 Business Case

The scope and goals of the business case are the same as defined in [1] and shown in Figure 63. Thanks to the access to the data and performance indicators of the digital marketing campaigns launched and managed by JOT, the company is developing a new revenue stream by offering dynamic and customizable analysis of the main user interest in different locations and business verticals.


Upcast Requests

Home
Requests

User: Name Surname
Logout

Request Name (optional)

Country (max 3): Select Options
Language (max 3): Select Options
Category (max 3): Select Options
Audience
Start Date: 17/05/2024
End Date: 17/05/2024
Search Terms (optional. max 3 comma separated)

Save request
Cancel

Figure 63: Digital Marketing data specifications.

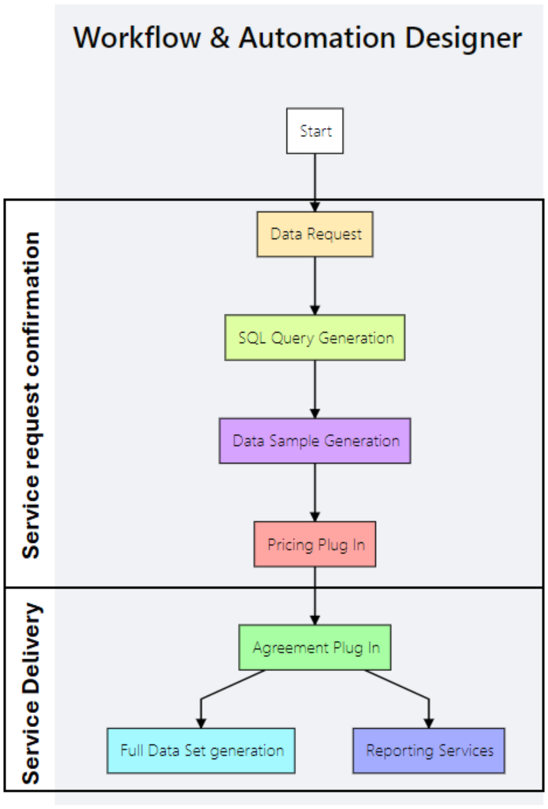


Figure 64: Business Case workflow model.

For the monetization of the marketing data, and due to the volume of data, it is required that the service request and delivery are performed in a sequential approach. Main goal is to ensure that all the requirements from the user are included in the final data monetization service. Also, it avoids the generation of full data sets, incurring in processing time and cost that do not meet the data consumer expectations. Every data set is generated by direct connection to the BigQuery repository in Google Cloud,

Figure 64 shows the business case model using Data Processing Workflow Modelling UPCAST plugin. As the business service is executed the requested plugins calls are carried out, mainly for data set pricing (which is very dependent on the features defined by the data consumer in the initial request) and the negotiation one, which states the service delivery conditions.

There are additional plugins that can be integrated in this workflow such as the monitoring and resource allocation, but they have not been included in the picture to facilitate the workflow comprehension and how the value is created in each step.

6.2.2 Datasets

The datasets used to implement this data monetization model are based on the performance of the digital marketing campaigns managed at massive scale by JOT. The large volume of campaigns covering almost every country and all the business verticals enables the generation of value to a wide diversity of market segments.

In order to facilitate the definition of the data set request, the pilot has defined a user interface with a limited number of degrees of freedom: country, business verticals (category), language, date and short free text. There is also an optional feature in case the data consumer would like to know the most frequent user persona looking for the information/interest requested.

With the aim of providing useful information the data will be aggregated at week or monthly level.

Data sets are shared with the data consumers (clients) in *.csv or *txt format by means of a sharing space or link for direct download.

6.2.3 Requirements and technical challenges

A detailed review of the requirements and technical challenges described in D1.1 (sections 4.4.2 and 4.4.5) has been carried out. Considering the current development status and the difficulties addressed, Tables 12, 13 and 14 of that deliverable is still fully relevant

6.2.4 Processing workflows

Two workflows are defined, as follows (Figure 65):

1. **Service Negotiation**: It covers all the steps needed till the data monetization contract signature. This workflow comprises the following steps:
    a. Data set request definition
    b. Generation of the query to the central data base
    c. Calculation of the price based on the benchmark (according to data set core features) and additional adjustments considering JOT internal variables.
    d. Data set sample verification by the data consumer
    e. Contract generation and signature
2. **Service Delivery**: In this case, the workflow covers the development of both the generation of the requested final data set and the creation of the static and dynamic reports containing the main insights embedded in the data.

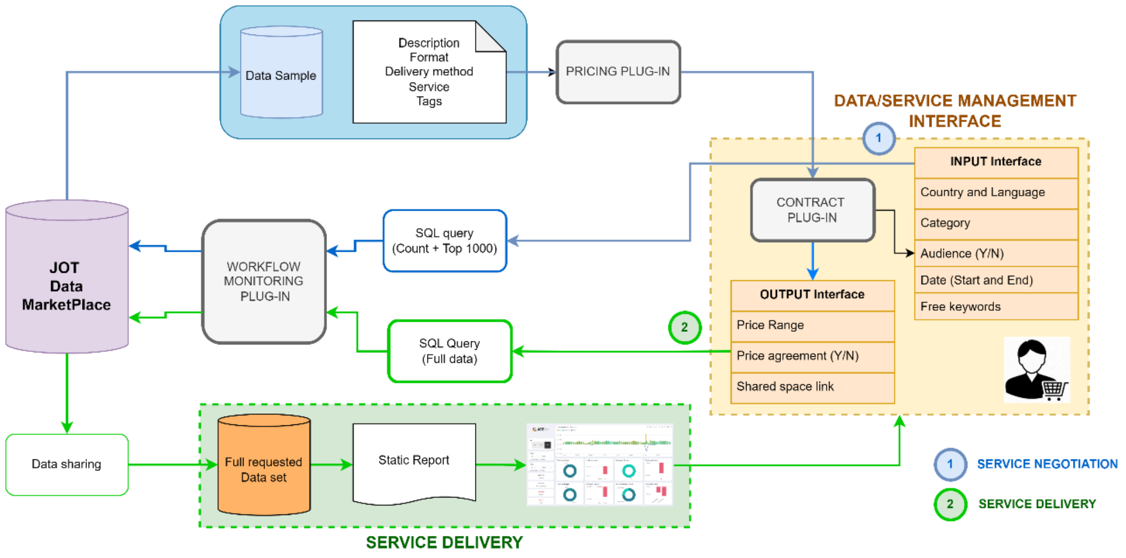


Figure 65: Digital Marketing workflows.

6.3 Digital Marketing Data and Resources

6.3.1 Business Case

With the evolution of digital marketing over the years, leveraging social media, mobile devices, data analytics, and personalized targeting, Cactus utilizes client data, primarily from Google and Meta Analytics, to identify the optimal digital marketing tools tailored to each client. Additionally, financial data is considered to develop a comprehensive marketing strategy that aligns with the client's overall business objectives.

6.3.2 Datasets

The data used are from the following channels: Google Analytics, Meta Analytics, Google Ads, Sales Data, and the P&L statement. Specifically, the data collected from Google Analytics include Visitors, ROAS, Click-through Rate (CTR), Click per Cost (CPC), Quality Score, Bounce Rate, Channels, and Conversion Rate. From Meta Analytics, Cactus consider Budget, ROAS, Landing Page View, Cost per Lead (CPL), Reach, Impressions, and Frequency. Cactus also gathers data from Google Ads, including Quality Score, ROAS, Budget, Click-through Rate (CTR), and Click per Cost. Cactus has the ability to incorporate Sales data and Profit and Loss statements to determine the best digital marketing tools for their clients. By analyzing all these data, Cactus can identify the weaknesses and strengths of their clients and determine the tools that will help boost their sales.

6.3.3 Requirements and technical challenges

Cactus' goal is to automate its business procedures, and while pursuing this objective, there are several important challenges that need to be addressed:

- Data Sharing: Establishing a seamless and secure process for clients to share their data with Cactus is crucial. Implementing secure data transfer protocols and providing user-friendly interfaces will facilitate efficient data exchange.
- Data Editing: Cactus needs to develop effective mechanisms to edit client data accurately and efficiently. Implementing robust editing tools and workflows will streamline the process and ensure data accuracy.
- User-Friendly Environment: Creating a user-friendly environment is essential for clients. This includes mobile responsiveness, minimizing downtime, and adhering to

stringent security protocols. Prioritizing intuitive interfaces and responsive designs will enhance the overall user experience.

- Multilingual features: Given the geographical diversity of clients, it is important to consider language requirements. Contracts should be available in the client's language all the other data and information could be in English.

6.4 Sharing Public Administration for Climate

6.4.1 Business Case

The aim of this pilot is to support data driven environmental policy making within the Metropolitan Area of Thessaloniki, by using the UPCAST plugins to streamline the process of integration and exchange of environmental data between the various stakeholders. These include public administration organizations, statistical authorities, research institutions, civil initiatives as data providers and municipalities, authorized organizations, researchers, citizens as data consumers.

Main goals are:

- Characterize the various uses of environmental data in Thessaloniki for decision-making and monitoring policy progress.
- Define the local ecosystem of actors involved in managing environmental data.
- Test organizational and governance challenges in managing and presenting environmental data to stakeholders.

6.4.2 Datasets

A number of diverse datasets from various data sources that could cover in an adequate degree the urban and environmental landscape of the Municipality of Thessaloniki have been identified through the Public Administration pilot. These datasets can be grouped into general categories such as i) environmental, ii) demographics, iii) urban, iv) traffic and v) other and come mainly from public statistical authorities as Eurostat and ELSTAT (Hellenic Statistical Authority).

Some examples of these datasets (have been described in D1.1, section 4.2.4) are air pollution measurements, traffic conditions, Urban Audit Indicators, transport statistics, dwellings energy sources and availabilities, population, gender and occupation distributions.

The Pilot aims to be able to support all data relevant to these categories to enrich its available resources, encourage data exchange and fill in some critical pieces of a data driven puzzle of environmental policy making.

Moreover, a generic data model that consists of more specific data models that describe the aforementioned data categories has been developed, in order to have a common representation and understanding of related data resources and subsequently facilitate search and integration processes. Resource providers of the platform will be able to adopt the pilot data model to describe their resources and transform their data into RDF format, leveraging the advantages of this representation.

Finally, resource consumers will be able to get data also in RDF format (where this transformation has taken place), besides more common data formats (e.g., csv files).

6.4.3 Requirements and technical challenges

No significant updates regarding the requirements and technical challenges of the pilot have been identified, as they have been described in Deliverable D1.1 (section 4.2).

6.4.4 Processing workflows

Two workflows have been identified, which are presented in this section.

1. **Share public administration data**. Data Publishers publish datasets specified using domain-specific vocabularies and ontologies that will be transferred securely to data users. This workflow may include the following steps:
    a. use the data definition plugin to create a structured definition for the dataset, ensuring clarity for further processing by plugins for privacy and discovery.
    b. upload datasets and their variations to meet the specifications and a potential DPW request.
    c. use the environmental cost estimation plugin to evaluate and optimize datasets for ecological impact.
2. **Integrate and aggregate public administration data**. Data consumers integrate and aggregate the data based on a defined data processing workflow. Using UPCAST plugins, they can negotiate with data providers in an automated way and at the same time respect data providers' privacy conditions and requirements. This workflow may include the following steps:
    a. Develop a DPW and publish it as a lead request for potential publishers.
    b. Discover suitable datasets using the Resource Discovery plugin. The pilot consumer evaluates these datasets and negotiates if suitable options are found.
    c. Use the negotiation plugin to start the negotiation and meet the pilot consumer's needs and any legal constraints for each dataset.
    d. Optional: Dataset producers can propose a service price for dataset preparation.
    e. Upon agreement, the pilot consumer purchases and gains access to the customized dataset variation.
    f. Use the UPCAST integration functionality to integrate the purchased datasets and proceed to data analysis as defined in the DPW.

6.5 Health and Fitness Data Trading

6.5.1 Business Case

Millions of people are sharing data in various fitness apps with the help of devices like wearables and IOT-enabled fitness equipment, creating very large datasets and streams. Personal fitness/health data, collected during various physical activities is extremely valuable for both the data producer (trainee), service providers (fitness, healthcare, wellbeing) and product vendors (e.g., vendors of the fitness equipment, nutrition supplements). However, wearables and fitness equipment are often used in “isolation”, meaning they are tailored to scenarios that benefit a single trainee. In this pilot, the UPCAST plugins will be used to valuate, share and trade data streams related to health and fitness data.

Smart4Fit[17] (TRL9) is a system for real-time monitoring for fitness based on personal wearables integrated in a bigger IoT environment (fitness club with plenty of connected fitness device) and is used in collaborative scenarios, like group training and collaborative gamification.

On the other hand, personal fitness/health data, collected during various physical activities has a good value not only for the data producer (trainee), but also for many service providers (fitness, healthcare, wellbeing) and product vendors (e.g., vendors of the fitness equipment, different supplements).

The use case is resolving challenges for an efficient and secure monetarization of such data. It explains the need for sharing the data and monetarize its value properly.

[17] https://smart4fit.nissatech.com/

6.5.2 Datasets

There are two main types of datasets:

Heart rate monitoring data: Heart rate data collected by monitoring users while training (fitness, medical): HR: Heart rate data refers to the measurements or recordings of a person's heart rate over a period. The heart rate is a measure of the number of times the heart beats per minute (bpm) and is commonly used as an indicator of a person's cardiovascular health and physical exertion. It is measured in beats per minute (BPM). Smart4Fit collects data at a frequency of 0.5 Hz, which means a new heart rate measurement is recorded every 2 seconds. Data is obtained from Bluetooth sensors during training sessions, allowing us to track and monitor the trainees' heart rate throughout their workout. Data is further analyzed in other **smart** analytical services to gain insights into the trainees' physiological response, intensity of the exercise, recovery patterns and overall cardiovascular fitness.

Acceleration monitoring data: Monitoring Acceleration data from accelerometer for trainee's better performance: Acceleration data from an accelerometer refers to the measurements or recordings of the acceleration experienced by an object or body in three-dimensional space. Unlike a gyroscope that measures rotational movement, an accelerometer specifically detects linear acceleration, including both static and dynamic acceleration. The accelerometer provides acceleration data in three axes: x, y, and z. Each axis represents a different direction or dimension of linear movement. The data collected from these axes allows tracking of changes in velocity or speed of the object in those directions. The sensor can be configured to collect data at various frequencies: 5, 52, 208 and 416 Hz. Higher frequencies provide more detailed data and capture rapid changes in acceleration, while lower frequencies may be suitable for capturing slower movements or conserving battery life. By collecting acceleration data from the accelerometer, one can analyze a trainee's movement patterns, assess the intensity of physical activities, detect impacts or sudden changes in velocity, and monitor body dynamics during training sessions. This data can be used to evaluate exercise techniques, quantify physical exertion, identify areas for improvement, and enhance the overall training outcome for the trainees.

6.5.3 Requirements and technical challenges

A detailed review of the requirements and technical challenges described in D1.1 (sections 4.3.2 and 4.3.5) has been carried out. Considering the current development status and the difficulties addressed, Tables 8, 9 and 10 of that deliverable is still fully relevant.

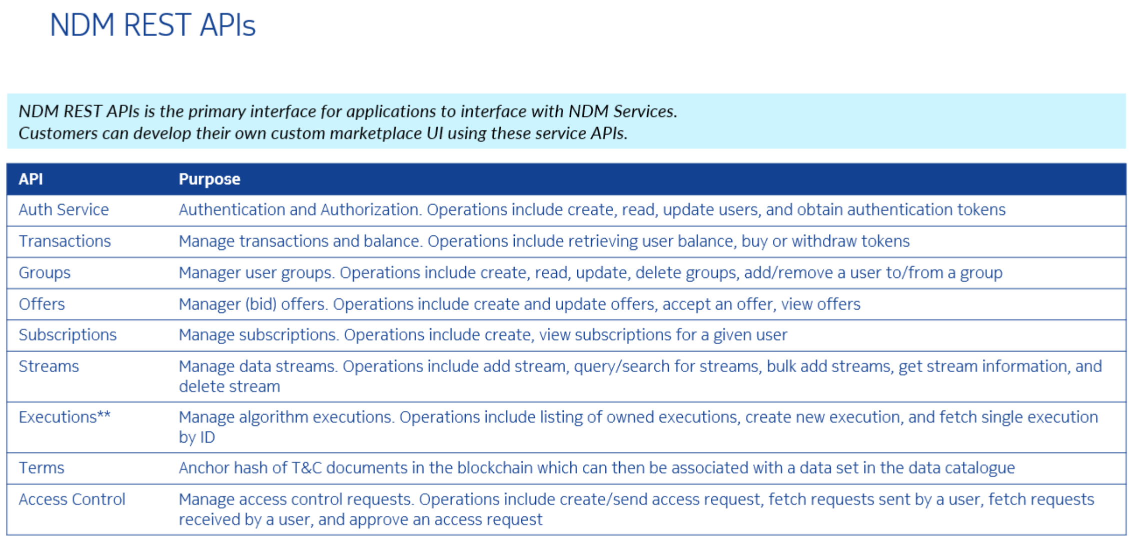
7 UPCAST Integration with Marketplaces

This chapter describes the mechanisms that will be used for the deployment of the plugins to the project's market places and integration with them.

7.1 Nokia Marketplace

7.1.1 Technical description

Each of the microservices in the NDM platform provides a RESTful API by which it can be used in any platform or customer application. An overview of the NDM Rest API is shown in Figure 66.



NDM REST APIs

NDM REST APIs is the primary interface for applications to interface with NDM Services.
Customers can develop their own custom marketplace UI using these service APIs.

API	Purpose
Auth Service	Authentication and Authorization. Operations include create, read, update users, and obtain authentication tokens
Transactions	Manage transactions and balance. Operations include retrieving user balance, buy or withdraw tokens
Groups	Manager user groups. Operations include create, read, update, delete groups, add/remove a user to/from a group
Offers	Manager (bid) offers. Operations include create and update offers, accept an offer, view offers
Subscriptions	Manage subscriptions. Operations include create, view subscriptions for a given user
Streams	Manage data streams. Operations include add stream, query/search for streams, bulk add streams, get stream information, and delete stream
Executions**	Manage algorithm executions. Operations include listing of owned executions, create new execution, and fetch single execution by ID
Terms	Anchor hash of T&C documents in the blockchain which can then be associated with a data set in the data catalogue
Access Control	Manage access control requests. Operations include create/send access request, fetch requests sent by a user, fetch requests received by a user, and approve an access request

Figure 66: NDM REST API.

To use any service within NDM, the user should authenticate himself using username/password and Auth service API. It will get an access token which should be pass in all the API in the header:

Create access token:

```
curl --location 'https://ukpn.dataexchange.nokia.com//auth/tokens' \
--header 'Content-Type: application/json' \
--data-raw '{
    "email": "buyer@ukpn.com",
    "password": "Buyer@ukpn123"
}'
```

7.1.2 Support for UPCAST Architecture

NDM (Nokia Data Marketplace)[18] is based on microservice architecture and is deployed on k8s cluster. So any plugin should come with helm chart[19] so it could be deployed within that cluster.

Although the deployment would be easy but communicating of the plugin to the other services within the cluster should be checked for each plugin separately as the functionalities could be overlapped with the existing and core functionalities of the NDM.

Moreover, for the other services to be able to work and communicate to the plugin, there may

[18] https://www.nokia.com/networks/bss-oss/data-marketplace/

[19] https://helm.sh/docs/topics/charts/

be a requirement for code development and code modification.

7.2 Dawex Data Exchange

7.2.1 Technical description and API Specs

In the context of the project, Dawex will provide a dedicated Data Exchange environment for pilots to provide and acquire data, using the UPCAST plugins when relevant, according to the scenarios validated in the Deliverable 5.1 [5].

Dawex's objective is to reference the plugins in the Data Exchange environment catalog so that they can be used on the platform by participants, after the contracting process.

There will be three main scenarios for plugin integration:

1. A scenario where the plugin is provided under a downloadable app (apk) packaged in a data product offering in the Data Marketplace,
2. A scenario where the plugin is exposed via a service offering in the Data Marketplace, to be called up and consumed via an API
3. A scenario where the plugin provider offers a data processing service in the Data Marketplace, that is next acquired by the users (pilots) to be used outside the platform.

7.2.2 Support for UPCAST Architecture

In the context of the project, Dawex will provide a dedicated Data Exchange environment for pilots to provide and acquire data, using the UPCAST plugins when relevant, according to the scenarios validated in Deliverable 5.1 [5].

Dawex's objective is to reference the plugins in the Data Exchange Platform catalog so that they can be used on the platform by participants, after the contracting process.

We have validated three main scenarios for plugin integration:

- A scenario where the plugin is provided under a downloadable app (apk) packaged in a data product offering in the Data Marketplace,
- A scenario where the plugin is exposed via a service offering in the Data Marketplace, to be called up and consumed via an API
- A scenario where the plugin provider offers a data processing service in, the Data Marketplace, that is next acquired by the users (pilots) to be used outside the platform.

To do so, the plugins must comply with the following standards/best practices

- W3C Verifiable Credential Data Model standard
- W3C DCAT v3 standard (Data Catalog Vocabulary)
- REST API oriented architecture conforming to the Open API standard (3.0 spec) to provide programming language-agnostic interface description
- OpenID Connect (OIDC) standard as an authentication layer on top the OAuth 2.0 framework to allows identity verification through any OIDC authorization server (Azure, GCP, AWS, …) in a REST-like manner and using JSON as a data format
- Pull APIs for asynchronous processing

8 Conclusions

D1.3 reports the final version of the UPCAST Architecture, which has been formed after analysis of pilot requirements, partners' technologies, integration, and deployment requirements to the project's marketplaces. The UPCAST architecture presented in this deliverable is a refinement of a the one that has been defined in the early stage of the project. The UPCAST architecture is versatile and extendible as it allows plugins to be integrated according to the needs of dataset providers and consumers. The UPCAST architecture that is given in the document will be the basis for the developments in the project, their integration in the project's marketplaces and the set up for the execution of the project pilots.

In addition to the high-level UPCAST architecture, the document gives an update of the plugins that are designed and developed in the project, along with their interface specifications, and typical uses of them. Moreover, the dashboards of the data provider and the data consumer that are used for specifying, publishing, discovering and processing datasets are presented.

Along with the UPCAST architecture, the deliverable gives also the updated workflow of using the UPCAST plugins from the dataset annotation to its execution and monitoring.

Finally, the document gives an update of the workflows and business case of the five project pilots.

9 References

[1] UPCAST, “D1.1: Project concept requirements setup,” 2023.

[2] UPCAST, “D1.2: MVP definition and architecture,” 2023.

[3] A. Azcoitia, C. Iordanou and N. Laoutaris, “Measuring the Price of Data in Commercial Data Marketplaces,” in *ACM Data Economy Workshop*, 2022.

[4] A. Azcoitia, C. Iordanou and N. Loutaris, “Understanding the Price of Data in Commercial Data Marketplaces,” in *IDCE*, 2023.

[5] A. Alhazmi, T. Blount and G. Konstantinidis, “Forbackbench: A benchmark for chasing vs. query-rewriting.,” in *VLDB Endowment 15*, Sydney, 2022.

[6] R. Fagin, P. G. Kolaitis and L. Ropa, “Data exchange: getting to the core.,” *ACM Transactions on Database Systems (TODS),* vol. 30, no. 1, pp. 174-210, 2005.

[7] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro and E. Tsamoura, “Benchmarking the chase.,” in *36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, Chicago, IL, USA, 2017.

[8] A. Artale, D. Calvanese, R. Kontchakov and M. Zakharyaschev, “The DL-Lite family and relations.,” *Journal of artificial intelligence research,* vol. 36, pp. 1-69., 2009.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati and G. Veter, “DL-Lite: Practical reasoning for rich DLs.,” in *International Workshop on Description Logics*, 2004.

[10] S.-. Das, “R2RML: RDB to RDF mapping language,” 2011. [Online]. Available: http://www. w3. org/TR/r2rml/.

[11] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Van de Walle, "RML: A generic language for integrated RDF mappings of heterogeneous data," in *LDOW*, 2014.

[12] A. Dimou, "R2RML and RML comparison for RDF generation, their rules validation and inconsistency resolution.," arXiv:2005.06293, 2020.

[13] T. Bagosi, D. Calvanese, J. Hardi, S. Komla-Ebri, D. Lanti, M. Rezk, M. Rodríguez-Muro, M. Slusnys and G. Xiao, "Bagosi, Timea & Calvanese, Diego & Hardi, Josef & Komla-Ebri, Sarah & Lanti, Davide & Rezk, Martín & Rodríguez-Muro, Mariano & Slusnys, Mindaugas & Xiao, Guohui. The Ontop Framework for Ontology Based Data Access.," *Communications in Computer and Information Science,* pp. 67-77, 2014.

[14] UPCAST, "D5.1: Monitoring and Evaluation Methodology," 2024.

[15] UPCAST, "D1.3: Updated project concept and architecture," 2024.

[16] A. Artale, D. Calvanese, R. Kontchakov and M. Zakharyaschev, "The DL-Lite family and relations.," *Journal of Artificial Intelligence Research (JAIR),* vol. 36, no. 1-69, 2009.

ACRONYMS

Acronym	Explanation
AI	Artificial Intelligence
CPU	Central Processing Unit
DPV	Data Privacy Vocabulary
DPW	Data Processing Workflow
DSL	Domain-Specific Language
EIO	Environmental Impact Optimiser
GDPR	General Data Protection Regulation
HPC	High Performance Computing
IDSA	International Data Spaces Association
LLM	Large Language Model
MVP	Minimum Viable Product
NLP	Natural Language Processing
ODRL	Open Digital Rights Language
PDP	Policy Decision Point
PMP	Policy Management Point
PUC	Privacy and Usage Control
RC	Resource Consumer
RP	Resource Provider
UI	User Interface
WMO	Workflow Model Ontology