2024

UPCAST PROJECT

Draft Document

DELIVERABLE 3.1

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

D3.1: Negotiation and Execution Modules v1

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

Title:	Document version:
D3.1 Negotiation and Execution Modules v1	0.60

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 15, 2024)	O-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).

Responsible:	Organization:	Contributing WP:
Sofoklis Efremidis	MAG	WP3

Contributing Authors (organization):

Abstract:

This document reports on the first version of the technologies and tools that support the negotiation between a dataset provider and a dataset consumer (work that is carried out in T3.1), and also the secure dataset execution and monitoring (work that is carried out in T3.2).

The negotiation process is detailed, along with the developed algorithms and the data models for (a) the usage and access policies and (b) the produced contracts. Moreover, the secure dataset transfer and the dataset execution engines that will be used by the project's pilots are presented along with the monitoring tasks of both the dataset execution and the dataset preparation tasks.

Keywords:

Negotiation, policy, contract, dataset execution, monitoring.

REVISION HISTORY

Revision:	Date:	Description:	Author (Organisation)
v0.10	30/04/2024	First version of the document, with suggested structure and content	Sofoklis Efremidis (MAG)

v0.15	10/05/2024	Updates to the structure of the document after feedback from partners	Sofoklis Efremidis (MAG)
v0.30	19/062024	Updates based on contributions received by partners	Sofoklis Efremidis (MAG)
v0.40	2/7/2024	Updates based on contributions and comments by partners	Sofoklis Efremidis (MAG)
v0.50	12/7/2024	Updates after internal review	Sofoklis Efremidis (MAG)
v0.60	15/7/2024	Updates after internal review	Sofoklis Efremidis (MAG)

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

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Overview

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 3, Negotiation and Execution, which addresses the following project objectives:

- Objective 2: Automate privacy-compliant, fairly-priced negotiation and development of data sharing and processing contracts.
- Objective 3: Enable scalable, safe, secure and verifiable data sharing and trading.
- Objective 4: Enable interoperability of data sharing across different entities, platforms and marketplaces.
- Objective 5: Provide a legal and ethical framework for automated contracts.
- Objective 6: Improve environmental sustainability of data processing workflows.

These project objectives will be achieved by WP3 through the following sub-objectives:

- Objective 3.1: To integrate the privacy and pricing discovery and integration tools from WP2 and deliver a solution supporting an agile and sustainable negotiation process involving the corresponding stakeholders.

- Objective 3.2: To provide and integrate the components fulfilling the security framework upon the requirements and architecture constraints emanating from WP1 and WP2.
- Objective 3.3: To design, model, implement and assess an environmental impact model which drives UPCAST platform to run on an optimal operating point.

1.2 Purpose of the Document

This document reports on work that has been carried out in tasks T3.1 and T3.2 of WP3 that relates to the negotiation tasks between dataset providers and consumers, the data exchange and execution of datasets between the two stakeholders, and the monitoring of the dataset execution and data preparation. The document provides the technical design of the corresponding plugins and execution environments.

1.3 Scope of the Document

This document is based on the work that has been carried out in Tasks T3.1 and T3.2 of WP3. The description of the tasks is as follows:

Task 3.1 Contract Negotiation Module. T3.1 develops the technology and tools to automatically reconcile conflicting requirements, negotiate, recommend, amend, enforce, and update agreements between partners. The task also develops theory, algorithms and tools for planning and optimisation of a workflow. Based on the output of WP2, the tools will re-structure or update the workflow achieving the optimal trade-off between price, computational cost, privacy enforcement and energy efficiency.

Task 3.2 Safe and Secure Execution and monitoring. T3.2 implements the safe and monitored execution plugin by providing a distributed proxy to ensure the safe, traceable and secure data exchange within secure execution environments, on the cloud or on local servers, supported by the Nokia Data Marketplace technology. For monitoring, the task creates workflows' Digital Twins along with the necessary visualizations for all phases (from design to agreement and performance) based on the KPIs identified by the pilot users and the data coming from the connectors developed in Task 4.1.

Deliverable D3.1 reports on the first version of the negotiation, data sharing, execution, and monitoring modules of UPCAST.

1.4 Structure of the Document

The remainder of the document is organised as follows: Chapter 2 gives an overview of the UPCAST Architecture to facilitate the presentation of the data exchange and execution, negotiation and monitoring plugins that are presented in the following chapters and put them in the context of the Architecture. Chapter 3 presents the UPCAST MVP and the typical UPCAST workflow that relates to the preparation, annotation, publishing, discovery, negotiation and execution of a dataset. Chapter 0 presents the UPCAST support for the negotiation actions between a dataset provider and a dataset consumer, which, if successful, results to a contract between them. Moreover, the data models of policies and contracts are presented in this chapter. Chapter 5 presents the data exchange activities between a dataset provider and a dataset consumer as well as the support of UPCAST for the execution of the Data Processing Workflows. Chapter 6 presents the monitoring functions of UPCAST, which collect execution related data for logging key actions during dataset execution and confirming compliance to the agreed contract between the dataset provider and the consumer, and data that relate to the preparation of a dataset by the dataset provider. Finally, Chapter 7 concludes the document.

2 UPCAST Architecture

This chapter presents the UPCAST Architecture that will be used for the developments and integration tasks of the project. The final version of the UPCAST Architecture is an extension of the one presented in [1] and is presented in detail in [2] and is repeated in this chapter for completeness of the document.

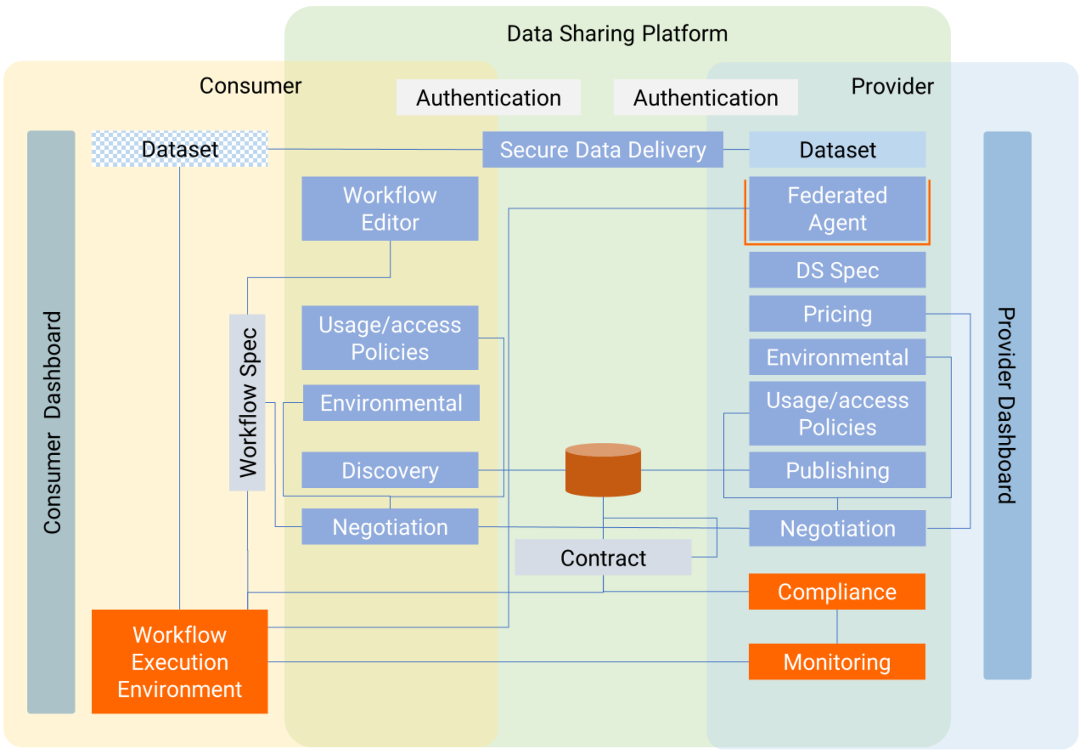


Figure 1: UPCAST Architecture.

Figure 1 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 the Data Marketplace. The Data Sharing Platform allows the authentication of the Providers and Consumers, and also provides the hosting environment to which the components of the architecture (plugins) are deployed. 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 for subsequent publishing it to 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 also 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 make estimates of 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 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 1 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, which the execution has the obligation to emit for the producer to verify compliance with the contract.

Execution takes place in a fully controlled and auditable way, which means 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 that they comply to the terms of the contract that have 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.

3 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 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).

The UPCAST Minimum Viable Product (MVP) is an implementation of the minimum functionality of the UPCAST plugins (described in [2]) 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.

This chapter presents the UPCAST MVP functionality from a user perspective. In the context of this presentation, users of UPCAST are ether dataset providers or dataset consumers. Figure 2 illustrates the core functionality that is included in 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 the actions the dataset provider and the dataset consumer can take and the components that support these actions. This deliverable focuses on the support for Negotiation and Contracting, Secure Data Exchange, and Monitoring. Moreover, the execution environments that will be used by the project pilots are also presented in subsequent chapters.

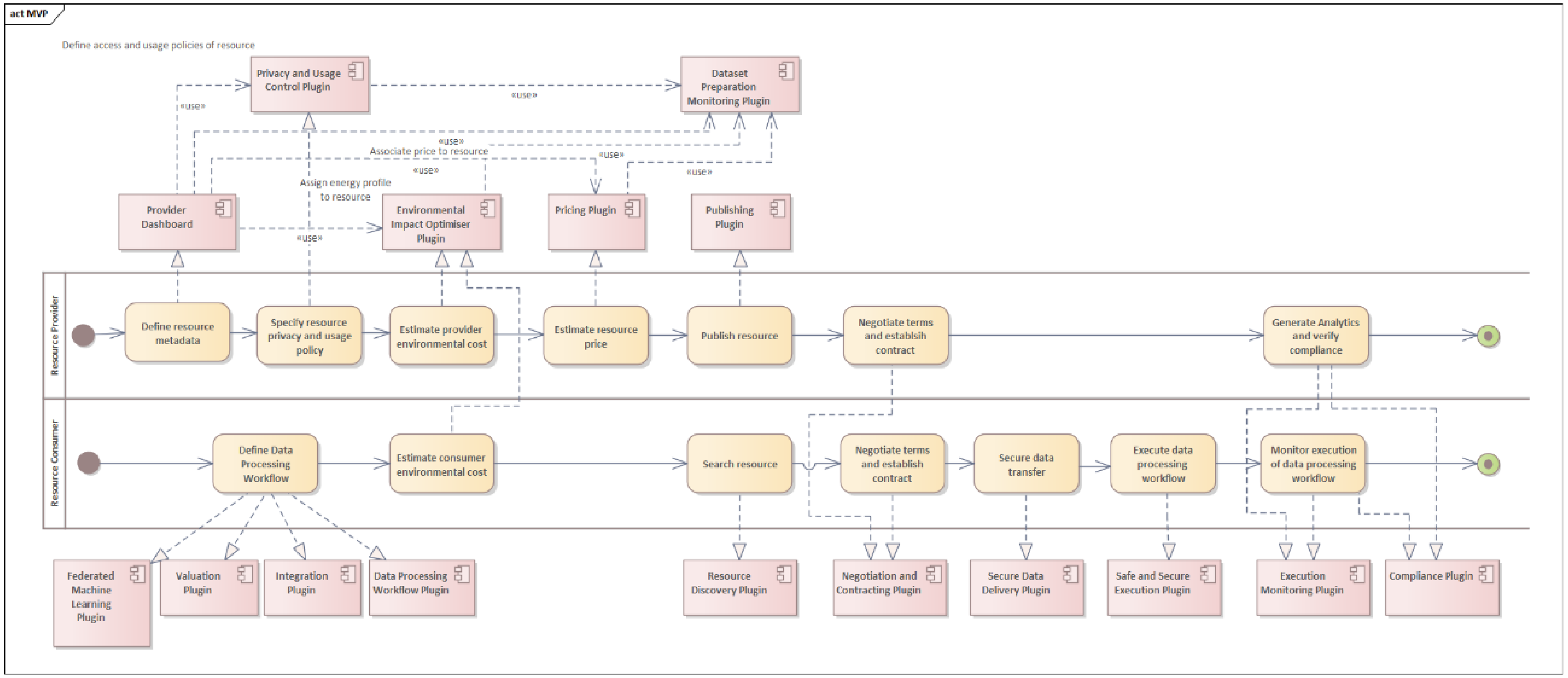

act MVP
Define access and usage policies of resource
Privacy and Usage Control Plugin
Dataset Preparation Monitoring Plugin
«use»
Associate price to resource
Assign energy profile to resource
Provider Dashboard
Environmental Impact Optimiser Plugin
Pricing Plugin
Publishing Plugin
Resource Provider
Define resource metadata
Specify resource privacy and usage policy
Estimate provider environmental cost
Estimate resource price
Publish resource
Negotiate terms and establsih contract
Generate Analytics and verify compliance
Resource Consumer
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
Federated Machine Learning Plugin
Valuation Plugin
Integration Plugin
Data Processing Workflow Plugin
Resource Discovery Plugin
Negotiation and Contracting Plugin
Secure Data Delivery Plugin
Safe and Secure Execution Plugin
Execution Monitoring Plugin
Compliance Plugin

Figure 2: Core functionality of UPCAST MVP.

For the 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 platform) and offer defined functionality to users through the marketplace. Plugins interact with each other and with the marketplace in which they are deployed through well-defined APIs. The users, acting as resource providers or resource consumers, can select a desired plugin from the marketplace and invoke the required functionality through the provided user interface (depicted as dashboard in Figure 2).

Figure 2 shows a representative user journey with activities that involve all of the UPCAST plugins. The upper part of Figure 2 illustrates the actions of a resource provider who wants to 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 is outside the scope of UPCAST. Therefore, the provider experience starts with the dataset annotation in which the provider describes the resource using basic metadata or semantic metadata and defines access and usage policies. The provider may also assign an energy profile to the resource for its generation and storage and also associate a price or price range to the resource 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 Provider Dashboard the provider can define the privacy and usage control policies for the resource that are supported by the Privacy and Usage Control plugin.

RP3. *Estimate provider environmental cost*: Using the Provider Dashboard, the provider can create the energy profile for the resource that relates to the collection and storage of the dataset 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 by using the functions of the Pricing plugin.

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.

RP6. *Negotiate terms and establish contract*. ADD TEXT. The dataset provider and the dataset consumer need to negotiate terms of the policies and requirements that are expressed by both sides. Negotiation is an iterative process, which, if successful, will result into a contract that forms the basis for the dataset workflow execution and verification for compliance.

RP7. Generate Analytics and verify compliance. The dataset provider received in their

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

dashboard monitoring data and corresponding analytics for the dataset execution. Compliance of the execution is checked against the agreed terms of the contract and any violations are notified to the provider.

The lower part of Figure 2 illustrates the actions of a resource consumer who wants to make use of a dataset resource. This consumer starts with "Define Data Processing Workflow" (RC1) which may utilise datasets from several providers, in general. The DPW may contain 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 2, 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 be 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 establish 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 transfer*: With the help of the Secure Data Delivery 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 following chapters give details for the Negotiation and contracting plugin, the Secure Data transfer plugin, the monitoring plugin, as well as the execution environments that will be used in UPCAST for the execution of the DPWs of the project pilots.

4 Negotiation and Contracting

This chapter presents the negotiation and contracting functions of UPCAST. The chapter starts with a description of the problem and the motivation for having such functionalities, and then moves on to give the architecture and design of the corresponding module. A video demo of the negotiation plugin is shown at Negotiation Plugin Demo.

Often, the processing intentions of a data consumer for a dataset of their interest differ from what the data provider is willing to allow. These differences may include the purpose of the processing, the time interval for which the provider is willing to allow access, or the price to pay. Nevertheless, these differences are not necessarily irreconcilable, and both parties can often reach an agreement through negotiation.

The Negotiation and Contracting plugin within the UPCAST, serves as a pivotal component, streamlining the complex processes of negotiation and contract management. With its multifaceted functionality, this plugin facilitates efficient communication and collaboration between data producers and data consumers. It enables users to initiate, track, and finalize negotiations seamlessly, providing a centralised platform for discussing terms, pricing, and specifications. Moreover, the plugin incorporates robust contract management features, allowing stakeholders to create, review, and execute contracts with ease. By automating routine tasks and offering customizable Data Processing Workflows (DPWs), it enhances data sharing while ensuring compliance with regulatory requirements.

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[2]), which in turn uses the Open Digital Rights Language[3] (ODRL), enabling more descriptive and technology-independent contracts. UPCAST contracts also utilise other ontologies such as Data Privacy Vocabulary (DPV[4]), 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.

UPCAST's negotiation plugin serves as a Policy Management Point (PMP) for usage restrictions. It reads machine-readable contracts and checks against information from the privacy and usage control, environmental impact, and pricing plugins, and reaches an agreement if there are no policy conflicts. If conflicts arise, a negotiation will be initiated, allowing the data provider or consumer to present counteroffers. The provider will ultimately decide the negotiation's outcome by agreeing, rejecting, or sending another counteroffer. Additionally, the plugin provides a Policy Administration Point with a user-friendly graphical interface, enabling users to edit policies. This allows users to define restrictions, privacy, and usage policies.

[2] https://docs.internationaldataspaces.org/ids-knowledgebase/v/dataspace-protocol

[3] https://www.w3.org/TR/odrl-model/

[4] https://w3c.github.io/dpv/dpv/

UPCAST combines the strengths of two previously demonstrated technologies: Southampton's toolkit for enabling personal consent (EPCON) to support business rules and the goodFlows tool from ICTabovo, that fosters automated process re-engineering towards compliance with the GDPR, based on comprehensive modelling of the underlying rules.

The UPCAST negotiation plugin receives as input machine-readable access and usage constraints of datasets previously defined using the Privacy plugin, and a machine-readable list of processing intentions from a Data Consumer. The plugin provides reasoning mechanisms to facilitate the reaching of an agreement between data providers and data consumers, in particular, support for conflict identification, preparation of counter-proposals based on configurable negotiation ranges, and suggestion of conflict resolution actions to help reaching a contractual agreement. The reasoning mechanism supports negotiation using various prevalence schemes (e.g., most recent rule prevails, deny overrides, stricter rules prevail, Inclusion-Exclusion principle for comparing constraints, pre-actions, and contextual conditions) [3].

The Negotiation functionality within UPCAST is a comprehensive tool designed to facilitate the intricate process of reaching agreements between Resource Providers (RPs) and Resource Consumers (RCs). Triggered by the predefined matching of RP and RC for negotiation, along with the establishment of negotiation terms and preferences encompassing alternative values and interdependent rules, the plugin ensures a systematic approach to the negotiation process. Initially, it verifies the compatibility of DPW against RP constraints, RC intentions, legal frameworks, and organizational policies, while addressing pricing and environmental considerations within the data source description. In the absence of conflicts, an automatic agreement is reached. However, if conflicts arise, the plugin orchestrates a structured negotiation sequence, managing the exchange of offers and requests (more generally, counter-offers) between RP and RC, respectively. Through a user-friendly interface, both parties can propose, modify and accept terms until a mutual agreement is achieved. The negotiation process respects predefined negotiation ranges for each statement in the resource specification, providing flexibility for adjustments. Upon agreement, the plugin proceeds to contract generation, producing both machine-readable and natural language contracts. Throughout the process, transparency is ensured as the negotiation outcome is presented to both parties, empowering them to make informed decisions.

4.1 Negotiation Scenarios

4.1.1 Negotiation Example

A data provider makes an initial offer to share his health data; however, he mentioned that the data must be anonymized and the purpose of data sharing must be Academic Research. A data consumer who seeks the data marketplace, discovers the data provider's health data and finds them useful for processing. He also wants to display the data for research and development purposes. Thus, he sends a request to the provider to express his needs. The data provider agrees the request with a refinement on display action; the consumer can display the data only via print media. If the consumer accepts the counter-offer, an agreement between data provider and data consumer will be established which will be finalised through a contract.

The JSON specification of the above initial offer, request, and counter-offer in ODRL are shown in the following.

Initial offer:

```
{
  "@context":
     "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy:001",
  "profile": "http://example.com/odrl:profile:11",
  "permission": [
     {
        "target": "http://example.com/ProviderHealthDataset",
        "action": "share",
        "duty":{
           "action":"anonymize",
        },
        "constraint":
           {
              "leftOperand": "Purpose",
              "operator": "eq",
              "rightOperandReference": {"@value": "https://w3id.org/dpv#AcademicResearch", "@type": "xsd:uid"}
           }
     }
  ]
}
```

Request:

```
{
  "@context":
     "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy:001",
  "profile": "http://example.com/odrl:profile:11",
  "permission": [
     {
        "target": "http://example.com/ProviderHealthDataset",
        "action": "display",
        "duty":{
           "action":"anonymize",
        },
        "constraint":
           {
              "leftOperand": "Purpose",
              "operator": "eq",
              "rightOperandReference": {"@value": "https://w3id.org/dpv#ResearchAndDevelopment", "@type": "xsd:uid"}
           }
     }
  ]
}
```

Counter-offer:

```
{
  "@context":
    "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy:001",
  "profile": "http://example.com/odrl:profile:11",
  "permission": [
    {
      "target": "http://example.com/ProviderHealthDataset",
      "action": [{
        "rdf:value": { "@id": "odrl:display" },
        "refinement": [{
          "leftOperand": "media",
          "operator": "eq",
          "rightOperand": { "@value": "print", "@type": "xsd:integer" },
        }]
      }],
      "duty":{
        "action":"anonymize",
      },
      "constraint":
        {
          "leftOperand": "Purpose",
          "operator": "eq",
          "rightOperandReference": {"@value": "https://w3id.org/dpv#ResearchAndDevelopment", "@type": "xsd:uid"}
        }
    }
  ]
}
```

4.1.2 Pilot example

This section illustrates a negotiation based on the Nissatech[5] pilot [1], which relates to the collection and processing of fitness-related data. They collect data points, generated through wearable devices, fitness apps, and gym equipment, offer unique insights into individual health and wellness, and help individuals make informed decisions about their fitness routines, diet, and overall lifestyle. Moreover, aggregated fitness data has the potential to contribute to broader public health initiatives and medical research. In this example, the individuals that train at a gym play the role of data provider, while Nissatech plays the role of data consumer that wants to process that data to create a product. Table 1 shows sample input data from the data provider side, the purposes of data consumer, and needed operations.

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

Table 1: Sample of input data, operations, and purpose or output data.

Provider (Input data) – From people / sensors	Operations (Algorithms)	Consumer (Purposes)
• Heart rate data, from a heart rate monitor, such as a chest strap or a wristband. • Motion data, from an accelerometer and a gyroscope, which are embedded in the user's device or wearable • Location data, from a GPS, which is embedded in the user's device or wearable. • Feedback data, from the user's input, such as voice commands, touch gestures, or buttons, to receive the user's feedback, preferences, and goals. • Users' data such as age, height, weight, resting hour, etc. • Users' specific data about their long-term conditions, such as diabetes, cardiovascular disease, and even mental health • VO2 max • Anaerobic zone data/aerobic data • User Preferences	• Statistical analysis • Heart rate analysis to calculate the Avg, Min, Max, and standard deviation of the user's heart rate during a workout, and to compare it with the target heart rate zone based on the user's age, weight, and fitness level. • Mathematical (closed) formulas • Calorie estimation → estimate the number of calories burned by the user during a workout, based on the user's weight, height, age, gender, and activity type and duration • Machine learning and model training • Activity recognition → recognise the type of activity that the user is performing, such as walking, running, cycling, or swimming, based on the sensor data from the accelerometer, gyroscope, and GPS of the user's device • Data visualization • Performance evaluation -> display the user's performance metrics, such as	• Individual-related purposes • Fitness level estimation (age, resting heart rate, and maximum heart rate to estimate the user's fitness level, which is expressed as a percentage of the user's maximum aerobic capacity) • Workout intensity adjustment (fitness level, heart rate zone, and personal goals) → to provide the user with appropriate guidance and feedback • Health risk assessment (the user's vital signs, medical history, and environmental factors) → to identify and predict potential health risks for the user, such as cardiac arrhythmia, dehydration, or overexertion • Recovery time estimation (heart rate variability, muscle fatigue, and sleep quality) • Exercise recommendation (fitness level, VO2 max, resting hour, calories, height,

	speed, distance, elevation, and cadence, in graphs and charts, and to highlight the user's achievements and progress over time.	weight, GPS data, accelerometer data, electrocardiography data) • Diet recommendation (calories, height, weight) • Groups-related purposes • Group comparison and ranking, • Gamification. • Factories-related purposes • Advertising,

Figure 3 depicts the information model of the Nissatech pilot that defines the classes of data items and instances that the policies will refer to.

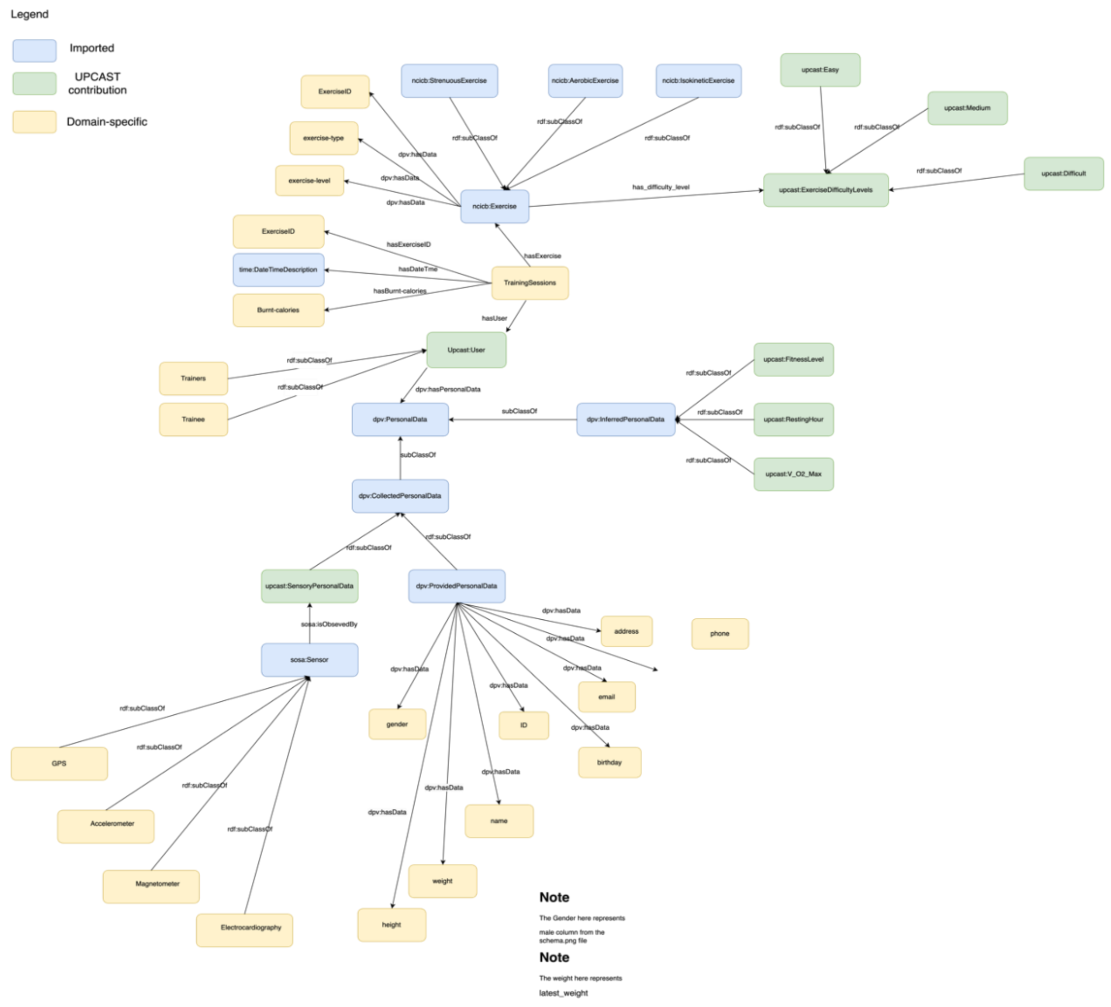


Figure 3: Nissatech information model.

Based on the above, three concrete examples are given in the following.

Example 1

In the first example, an ODRL policy is presented wherein the assigner, who is the data provider, grants permission to share their calorie information with anyone. The policy also specifies constraints regarding the date, exercise level, and exercise type. Specifically, the date must fall within the range of X and Y, the exercise level should be categorised as 'Difficult,' and the exercise type should be 'Aerobic'.

More concretely, the policy allows the sharing of burnt calories with any entity under the conditions that the date of training sessions is between X and Y, the exercise level is 'Difficult,' and the exercise type is 'Aerobic.'

```
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    {
      "dcat": "http://www.w3.org/ns/dcat#",
      "dpv": "https://w3id.org/dpv#",
      "rdf":"https://www.w3.org/TR/rdf12-schema#"
    }
  ],
  "@type": "Policy",
  "uid": "http://example.com/policy:001",
  "profile": "http://example.com/odrl:profile:11",
  "permission": [
    {
      "assigner": "https://example.com/assigners/DataProvider",
      "assignee": "https://example.com/assignees/Nissatech",
      "target": [{
        "@type": "AssetCollection",
        "source":  "http://example.com/TraniningSessions",
        "refinement":
        {
          "leftOperand":"Burnt-calories",
          "Operator":"eq",
          "rightOperand":{"@value":"SELECT ts.burntCalories
                          FROM training_session ts
                          JOIN Exercise ex ON ts.exerciseID = ex.exerciseID
                          WHERE ex.exercise-Type = 'aerobic'
                          AND ex.exercise-Level = 'difficult'
                          AND ts.DateTimeDescription BETWEEN 'X' AND 'Y';", "@type":"xsd:SQL_query"}
        }
    }]
      "action": "share",
      "constraint":
        {
          "leftOperand": "Purpose",
          "operator": "eq",
          "rightOperandReference": {"@value": "https://w3id.org/dpv#AcademicResearch", "@type": "xsd:uid"}
```

```
            }
        }
    ]
}
```

Example 2

In this example trainers are given consent to use a trainee's sessions information such as burnt-calories, exercise type, and exercise difficulty level, under the conditions that the date of training sessions falls between X and Y, with the purpose of generating a personalised plan for the trainee. Note in this case the DPV term "personalisation" is used as a general reference to the purpose.

```
{
    "@context": [
        "http://www.w3.org/ns/odrl.jsonld",
        {
            "dcat": "http://www.w3.org/ns/dcat#",
            "dpv": "https://w3id.org/dpv#",
        }
    ],
    "@type": "Policy",
    "uid": "http://example.com/policy:001",
    "profile": "http://example.com/odrl:profile:11",
    "permission": [
        {
            "assigner": "https://example.com/assigners/DataProvider",
            "assignee":  "https://example.com/Trainers",
            "target": [{
                "@type": "AssetCollection",
                "source":  "http://example.com/TraniningSessions",
                "refinement":
                {
                    "leftOperand":"upcast:queryResult",
                    "operator":"isAllOf",
                    "rightOperand":{"@value":"SELECT ts.Burnt-Calories, ex.exercise-Type, ex.exercise-level
                                          FROM TrainingSessions ts
                                          JOIN Exercise ex ON ts.ExerciseID = ex.ExerciseID
                                          WHERE ts.DateTimeDescription BETWEEN 'X' AND 'Y';", "@type":"xsd:SQL_query"}
                }
        }]
            "action": "Use",
            "constraint":
                {
                    "leftOperand": "Purpose",
                    "operator": "eq",
                    "rightOperandReference": {"@value": "https://w3id.org/dpv#Personalisation", "@type": "xsd:uid"}
                }
        }
    ]
}
```

Example 3

In this example the data provider gives permission to individuals who are both trainee and trainer to grant the use of trainee's information to third parties for research and development purpose, and as a refinement, the purpose must be approved by the ministry of health. (*grantUse* in ODRL vocabulary).

```
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    {
      "dcat": "http://www.w3.org/ns/dcat#",
      "dpv": "https://w3id.org/dpv#",
      "rdf":"https://www.w3.org/TR/rdf12-schema#"
    }
  ],
  "@type": "Policy",
  "uid": "http://example.com/policy:001",
  "profile": "http://example.com/odrl:profile:11",
  "permission":[
    {
      "assigner": "https://example.com/assigners/DataProvider",
      "assignee": [{
        "@type": "PartyCollection",
        "source":  "http://example.com/Trainee",
        "refinement": [
          {
          "leftOperand": "rdf:type",
           "operator": "eq",
          "rightOperandReferenec": { "@value": "http://example.com/Trainer", "@type": "xsd:uid"}
          }]
      }],
      "target":   "http://example.com/TraniningSessions",
      "action": [{
        "rdf:value":{ "@id": "odrl:grantUse" },
        "refinement": {
          "leftOperand": "third-party",
          "operator": "eq",
          "rightOperandReference": { "@value": "http://example.com/ThirdParty", "@type": "xsd:uid" },
        }
      }]

      "constraint": [{
        {
          "leftOperand": "purpose",
          "operator": "eq",
          "rightOperandReference": {"@value": "https://w3id.org/dpv#ResearchAndDevelopement", "@type": "xsd:uid"}
        },
        {
          "leftOperand":"approvedBy",
          "operator":"eq"
          "rightOperandReference": {"@value": "http://example.com/MinistryOfHealth", "@type": "xsd:uid"}

        }
      }]
  }]
}
```

4.2 Negotiation Plugin Architecture

The negotiation and contracting plugin architecture that is used for the developments and integration tasks of the plugin is presented in Figure 4 and detailed below.

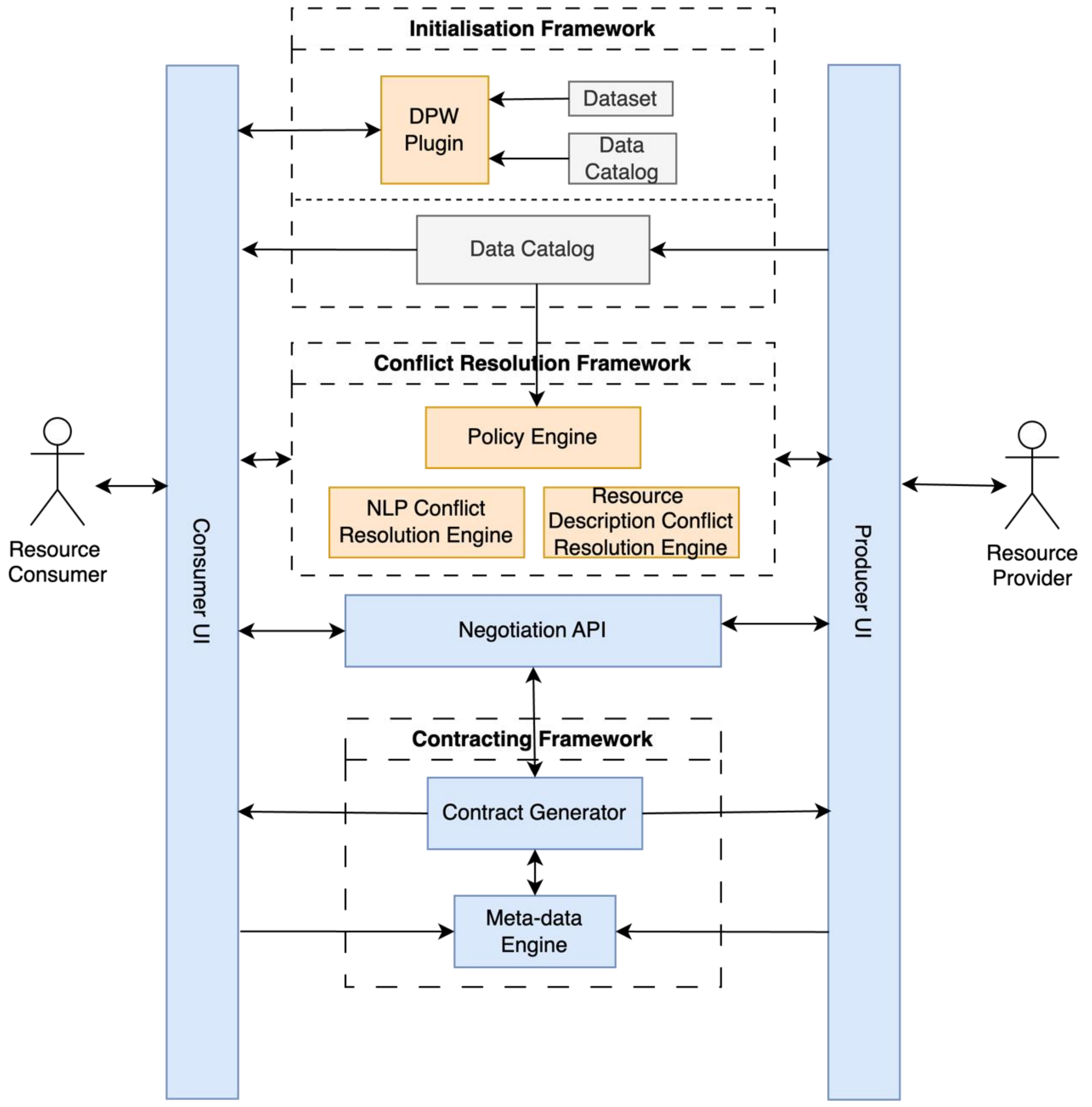


Figure 4: Negotiation and Contracting Plugin Architecture.

- Consumer/Producer UI: Provides user-friendly interfaces for initiating and managing negotiations.
- Initialisation Framework: There are two primary scenarios to start a negotiation, separated by a dashed link in the framework.
    - DPW plugin: Allows consumers to define their data processing requirements.
    - Dataset: This component, in alignment with the Data Catalog, provides data sources for the DPW plugin.
    - Data Catalog: Contains a list of advertised datasets. Providers register their datasets and initial offers here. Consumers can also browse the Catalog to find desirable datasets when the DPW plugin is not utilised.
- Conflict Resolution Framework
    - Policy Engine: Ensures compliance with DPW and ODRL rules, resolving any detected conflicts.
    - NLP Conflict Resolution Engine: Resolves conflicts in the natural language parts of the negotiation.
    - Resource Description Conflict Resolution Engine: Addresses conflicts in resource descriptions, such as price and environmental impacts.
- Negotiation API: Manages the entire negotiation process from initiation to termination or finalization.
- Contracting Framework
    - Contract Generator: Automates the creation of contracts, integrating all relevant details from the negotiation.
    - Meta-data Engine: Supports digital signatures, important data, and any contract-related information.

4.3 Usage Policy Data Model for Resource Provider

The usage policy is defined by the data provider. In this section, we describe the UPCAST Offer from the resource provider's perspective, which aligns with the usage policy data model. Figure 5 illustrates the UPCAST Offer data model.

In UPCAST, the resource provider specifies a resource along with the associated access and usage constraints. If a consumer finds the resource appealing but needs additional access, they generate an UPCAST Request. The provider may accept the request outright or reject it due to conflicts with their own policies. In response, the provider creates an UPCAST Offer, which is a message related to the initial UPCAST Request; it is also called a counter-offer.

The UPCAST Offer message consists of four parts: an ODRL Offer, a resource specification, a Data Processing Workflow Pattern, and a Natural Language Part. The following sections provide detailed explanations of these four components.

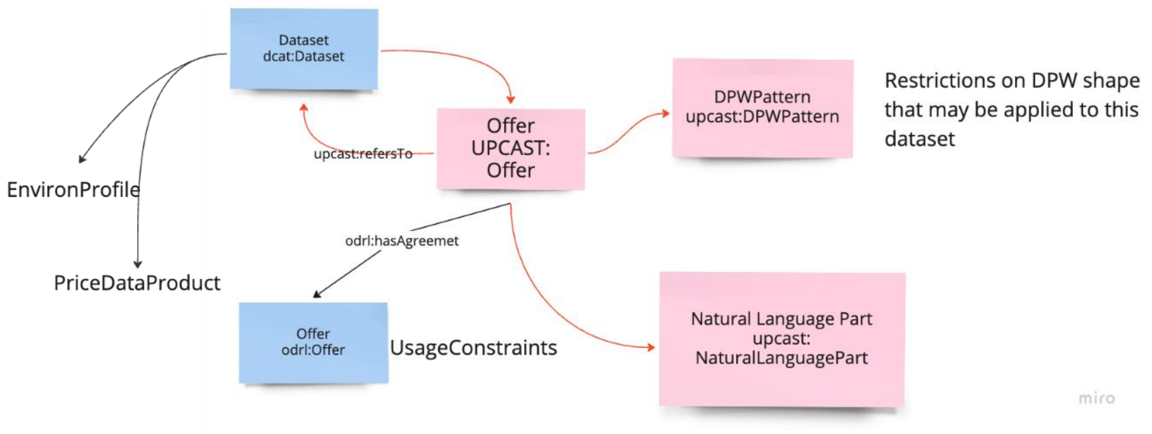


Figure 5: UPCAST Offer Data Model.

4.3.1 ODRL Offer

The following shows the structure of a provider odrl:Offer.

sent by	Data Provider	
format	ODRL, IDSA	
properties	Odrl:Rule	Permission with none, one, or more duties as the property of the Permission. Prohibition with none, one, or more remedies as the property of the Prohibition.
	odrl:Action	May include refinement(s)
	[odrl:Asset]	It defines as *target* in the request and it is one or many data source that consumer wants to access. It may also include none, one or many refinement(s)
	[odrl:Party] [idsa:Participants]	It defines as assignee and assigner in the request; refer to someone who get the permission and someone who give the permission, respectively. It may also include none, one or many refinement(s)
	[odrl:Constraint]	It defines any constraint on the rule based on the usage policies. Logical constraint can be also included.
	[odrl:uid]	It identifies the rule.

The UPCAST Offer message is sent by a provider to initiate a contracting negotiation or to respond to an UPCAST Request message sent by a consumer.

The odrl:Offer contains a *Rule* (odrl:Rule), which can be Permission, Prohibition, and Duty. Since providers usually submit initial Offers to control the usage of their data sources, they may offer a rule giving a Permission/conditional Permission based on a duty or defining a Prohibition and its remedies.

A *uid* identifies a Rule. According to the IDSA specification, a consumer must include a request property, which itself must have a @id property. If the odrl:Offer includes a providerPid property, the request will be associated with an existing contract negotiation and a consumer's UPCAST Request will be created using either the offer or offer.@id properties. If the odrl:Offer does not include a providerPid, a new contract negotiation will be created on provider side using either the offer or offer.@id properties and the provider selects an appropriate providerPid. An offer.@id will generally refer to an UPCAST Offer contained in a Catalog. If the provider is not aware of the offer.@id value, it must respond with an error message. A Catalog or data marketplace in UPCAST is a collection of entries representing datasets and their initial UPCAST Offers that is advertised by a provider participant.

The dataset id is not required but can be included when the provider initiates a contracting and negotiation. Different to a dataset (see DCAT Vocabulary Mapping), the odrl:Offer inside an UPCAST Offer message must have an odrl:target attribute. However, it's contained Rules must not have any odrl:target attributes to prevent inconsistencies with the ODRL inferencing rules for compact policies.

4.3.2 Dataset

Dataset in the UPCAST Offer is an explicit pointer to the target dataset; and is applied to resource description. To do this, the provider can create the energy profile for the resource with the resource environmental cost suggested by the Environmental Impact Optimiser Plugin. The environmental profile of the provider relates to the energy consumption for the collection of the dataset and its storage. Moreover, the provider can assign a price to the resource manually, or with the support of the Pricing plugin. Therefore, pricing and declaring energy and environmental issues are done in Dataset of the ODRL offer.

4.3.3 Data Processing workflow Pattern

Users generally define a series of actions related to pre-processing and processing of datasets using UPCAST DPW plugin. Moreover, a user, specifically a provider, may define access and usage control constraints at the level of the whole DPW as a pattern, in case the latter is planned to be advertised as a resource to be used in other DPWs (more information is given in section 4.6.2).

4.3.4 Natural Language component

The UPCAST negotiation and contracting plugin facilitates the creation of both machine-readable and natural language contracts when the resource provider and consumer reach an agreement. A natural language component is essential in the UPCAST Offer, as it provides a human-readable representation of the terms. This ensures clarity and understanding between the parties involved. Leveraging Large Language Models (LLMs) enhances this process by generating accurate and contextually relevant natural language summaries of the contract terms, improving communication and reducing misunderstandings.

4.4 Intentions Data Model for Resource Consumer

This section describes the UPCAST Request Data Model which is depicted in Figure 6.

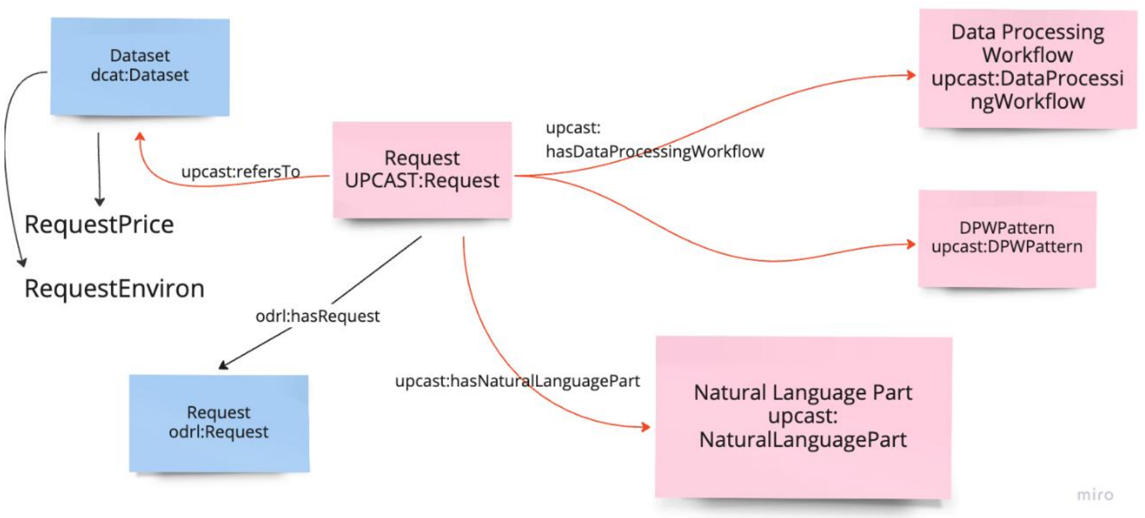


Figure 6: UPCAST Request Data Model.

The UPCAST negotiation and contracting plug-in is triggered by sending an UPCAST Request message. The UPCAST Request message has also several parts; an ODRL request, a Dataset, a Data Processing Workflow, a Data Processing Workflow Pattern, and a Natural Processing Part, which are explained in the following subsections.

4.4.1 ODRL Request

sent by	Data consumer	
format	ODRL, IDSA	
properties	odrl:Rule	Permission with none, one, or more duties as the property of the Permission.
	odrl:Action	May include refinement(s).
	[odrl:Asset]	It defines as *target* in the request and it is one or many data source that consumer wants to access. It may also include none, one or many refinement(s).
	[odrl:Party] [idsa:Participants]	It defines as assignee and assigner in the request; refer to someone who gets the permission and someone who gives the permission, respectively. It may also include none, one or many refinement(s).
	[odrl:Constraint]	It defines any constraint on the rule based on the usage policies. Logical constraint can also be included.
	[odrl:uid]	It identifies the rule.
	Purpose	The consumer must specify the purpose of the request; for what the requested data source will be used.

The UPCAST Request Message is sent by a Consumer to initiate a contracting negotiation or to respond to an UPCAST Offer message sent by a provider. The odrl:Request within the message contains a *Rule* (odrl:Rule) which can be Permission, Prohibition, and Duty. Since a consumer usually submits a Request to access a data source, we focus on Permission.

The definition of the rule and its properties are the same as the ODRL Offer. Moreover, the consumer should define his purpose of the request. A *Purpose* or goal, based on DPV definition, describes the intention or objective of why the data is being used, and should be broader than mere technical descriptions of achieving a capability. For example, "Analyse Data" is an abstract purpose with no indication of what the analyses is for as compared to a purpose such as "Marketing" or "Service Provision" which provide clarity and comprehension of the 'purpose' and can be enhanced with additional descriptions. Such modelling is in line with regulatory requirements regarding the specificity of purposes, for example in GDPR.

To express the Purpose, consumers add a constraint to the rule; here, the *purpose* is an instance of leftOperand property in ODRL and it is described as a defined purpose for exercising the action of the rule. To be compliant with Purpose class in DPV, odrl:*purpose* isA dpv:Purpose;

Based on the IDSA specifications, if the message includes a consumerPid property, the request will be associated with an existing contracting and negotiation process. If the message does not include a consumerPid, a new contracting and negotiation process will be created on consumer side and the consumer selects an appropriate consumerPid.

4.4.2 Dataset

The same as the UPCAST Offer, the UPCAST Request also needs a resource description part in which the consumer may get/submit a suggested price or price range of the resource and understand how the price is formed. Environmental issues of the request are also defined in Dataset part by the consumer.

4.4.3 Resource consumer intentions modelling and compliance by design

A DPW is a model of dataset processing and compliance of consumer intentions. As the negotiation and contracting processes heavily depend on it, they both have to be clear and concrete.

Formally, a DPW is a graph where tasks are the nodes, and the edges define the sequence of tasks, as well as the overall data and control flow. Edges carry information that defines the nature of data to be transferred from one task to the next. This information may refer to a specific dataset or describe the transferred data in abstract terms, using 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 transitions.

A resource consumer may specify an UPCAST Data Processing Workflow (DPW) leading to the formation of one or more UPCAST requests. Users generally define a series of actions related to the pre-processing and processing of datasets using the UPCAST DPW plugin. A consumer specifies a DPW offering desired functionalities; specifically, it defines intended data-centric processes alongside specific access and usage intentions and/or requirements. The DPW is also used to produce the processing specification for the dataset, which will be further converted to an execution specification.

Moreover, the environmental impact optimiser will generate energy consumption metrics for processes applied to the resource in the DPW and aggregate the energy consumption.

4.4.3.1 Data Processing Workflow model

In general terms, a workflow describes a series of actions with well-defined sequential relations and information dependencies among them. A workflow under execution is a *workflow instance*, whereas its specification is provided by a *workflow model*. In UPCAST, modelling of Data Processing Workflows adopts the approach elaborated in the context of H2020 BPR4GDPR[6] that leverages semantic technologies towards comprehensive modelling of workflows with inherent support of constraints specification in a workflow design. The underlying semantic ontology is referred to as the Workflow Model Ontology (WMO).

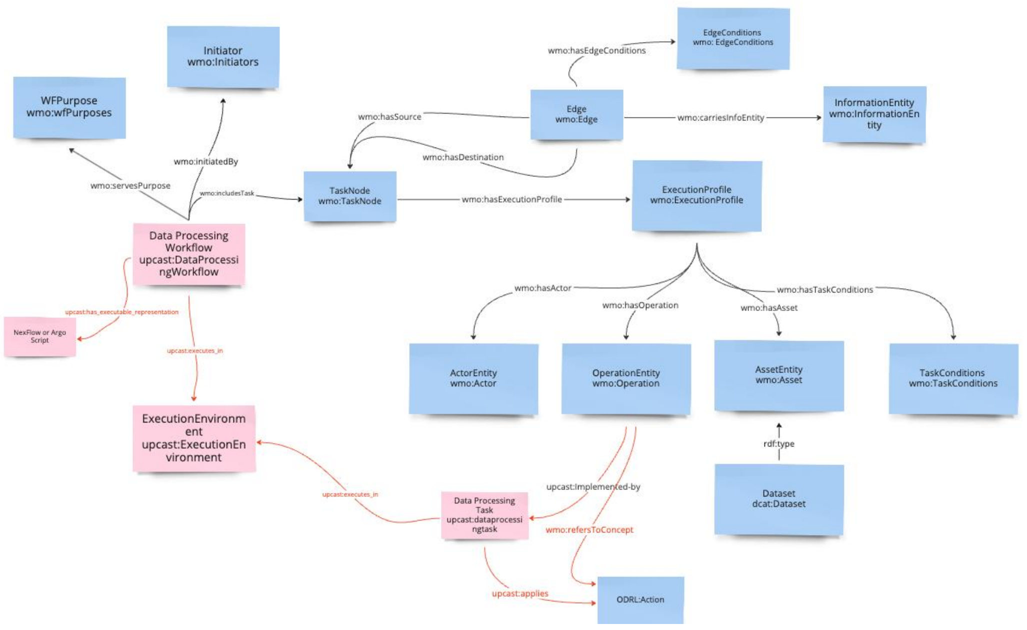


Figure 7: UPCAST Data Processing Workflow Model.

The most fundamental artefacts of a workflow model are *tasks* and *flows*. The former represent actions to be executed within the workflow, each describing the *operation* performed by an *actor* on an *asset*. Flows express dependencies between tasks, are represented through directed edges and are of two types: *control* and *data*. A control flow dependency between two tasks t_A and t_B means that t_B is executed only after the execution of t_A is completed; what the edge transfers is the thread of control, potentially accompanied by the necessary control parameters, if any. On the contrary, a data flow dependency assumes actual data of interest are exchanged (i.e., to be accessed and processed by the destination task), denoting explicit data dependencies. Further, a

[6] https://www.bpr4gdpr.eu

workflow model is complemented by the operational *purposes* it is meant to serve, and the potential *initiators*, denoting entities authorised to initiate the workflow. Therefore, a *workflow model* can be defined as a tuple $\langle T, F_C, F_D, Init, WFPu \rangle$, such that: T is a finite set of tasks $\langle t_1, t_2, \ldots, t_n \rangle$; F_C and F_D are sets of directed edges, expressing the control flow and data flow relations among tasks; *Init* is the set of human actors that, according to the given specification, are allowed to trigger the workflow execution; *WFPu* denotes the set of purposes for which the workflow is intended to be executed. Tasks, edges, purposes and initiators are instantiated as individuals of wmo:TaskNode, wmo:Edge, wmo:wfPurposes and wmo:Initiators classes, respectively.

The core constituents of tasks are actors, operations and assets, while for flows, the exchanged information is essential for edges definition. To adequately capture the core workflow perspectives (control, data, and resource [4]) a comprehensive approach for modelling these elements is adopted, centred around the notion of *enhanced entities*; the latter describe elements that their definition is either concrete, or abstract and constrained over attributes and/or subconcepts.

In this context, 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.

Similarly, tasks' operation, actors and assets are instantiated by the wmo:OperationEntity, wmo:ActorEntity and wmo:AssetEntity classes. However, the ontological specification of tasks includes an intermediate concept, that of *execution profiles*, enabling the specification of variations regarding the execution of a task, unlike the typically "monolithic" tasks' definition of other approaches. This concerns two aspects: differentiated execution based on some *conditions*, and capturing the dependencies between the task's actors, assets and operation constraints, that is, precisely defining their valid combinations.

Execution profiles are modelled through individuals of the wmo:ExecutionProfile class, appropriately linked to wmo:OperationEntity, wmo:ActorEntity and wmo:AssetEntity instances, or, in the case of actors and assets, to logical structures thereof. Further, they may be associated to wmo:TaskConditions, i.e., expressions defining conditions for the profile to be executed. Task conditions describe real-time constraints external to the workflow specification (e.g., contextual factors), or spanning beyond task boundaries, that cannot be expressed on the basis of referenced entities' attributes alone.

4.4.3.2 From DPW modelling to negotiation

For modelling Data Processing Workflows leveraging the WMO, UPCAST makes use of the goodFlows [7] prototype provided by ICT Abovo. goodFlows allows graphical specification of workflows along with its elements described above. Figure 8 provides an overview of the respective environment.

[77] https://www.ict-abovo.gr/goodflows

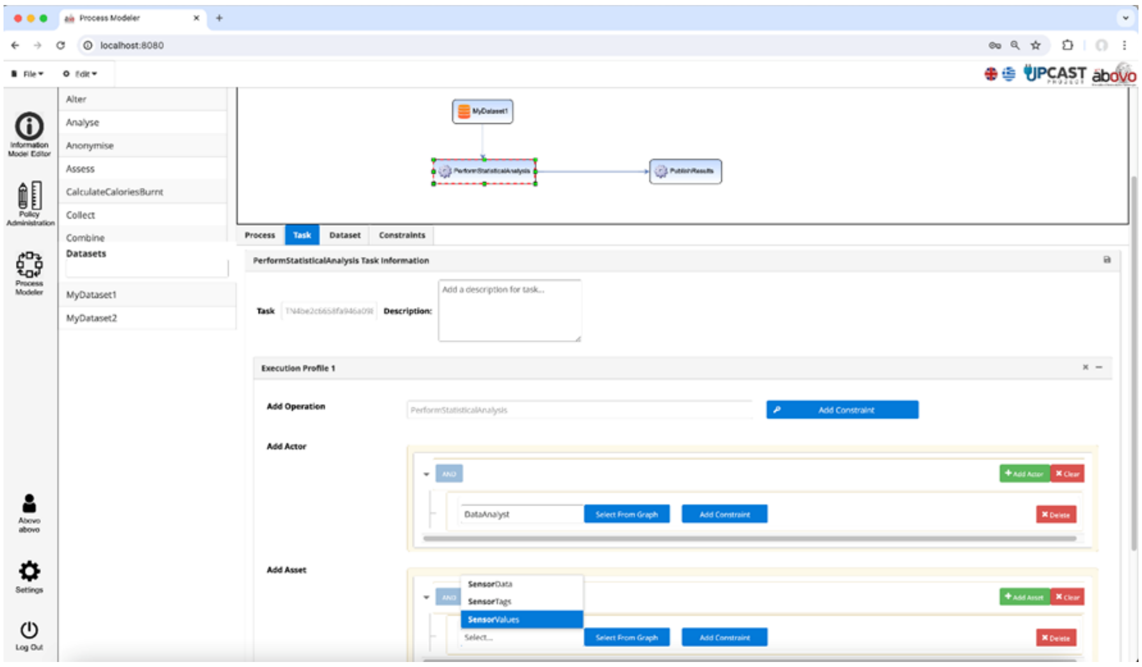


Figure 8: Data Processing Workflows modelling using goodFlows.

In the context of UPCAST, goodFlows is subject to several adaptations to implement essential project functionalities. As regards negotiation, goodFlows is evolving across several axes. First, the workflow modelling functionality shall constitute the consumer's gateway towards the discovery of datasets, as well as negotiating their acquisition. To this end, Figure 9 illustrates dataset discovery from within the workflow modelling environment, Figure 10 assumes that a dataset has been discovered and depicts its properties, whereas Figure 11 showcases the conflicts identified that will result in the initiation of the negation process.

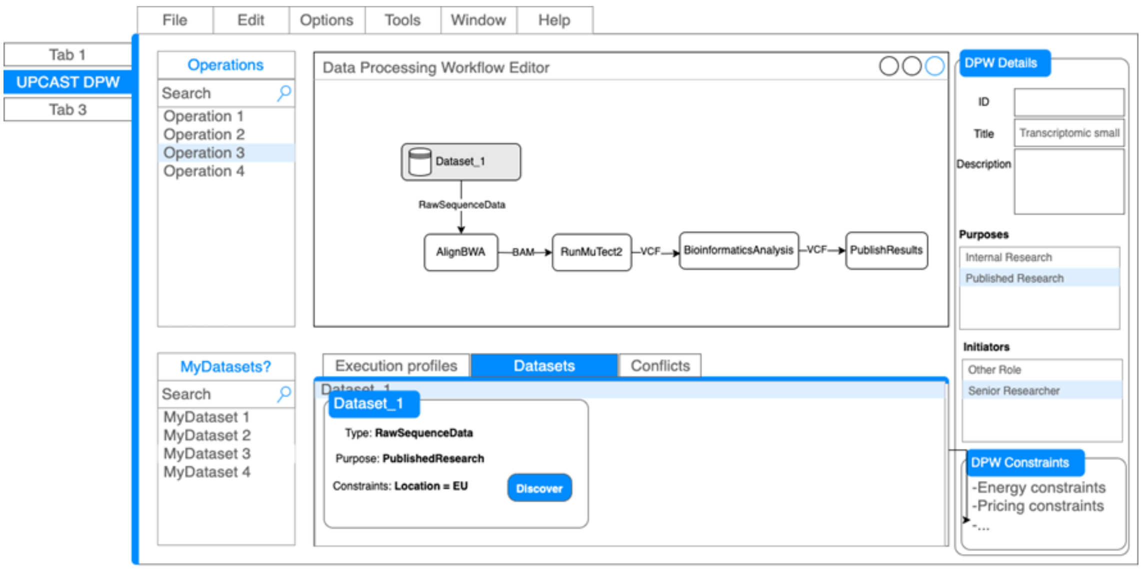
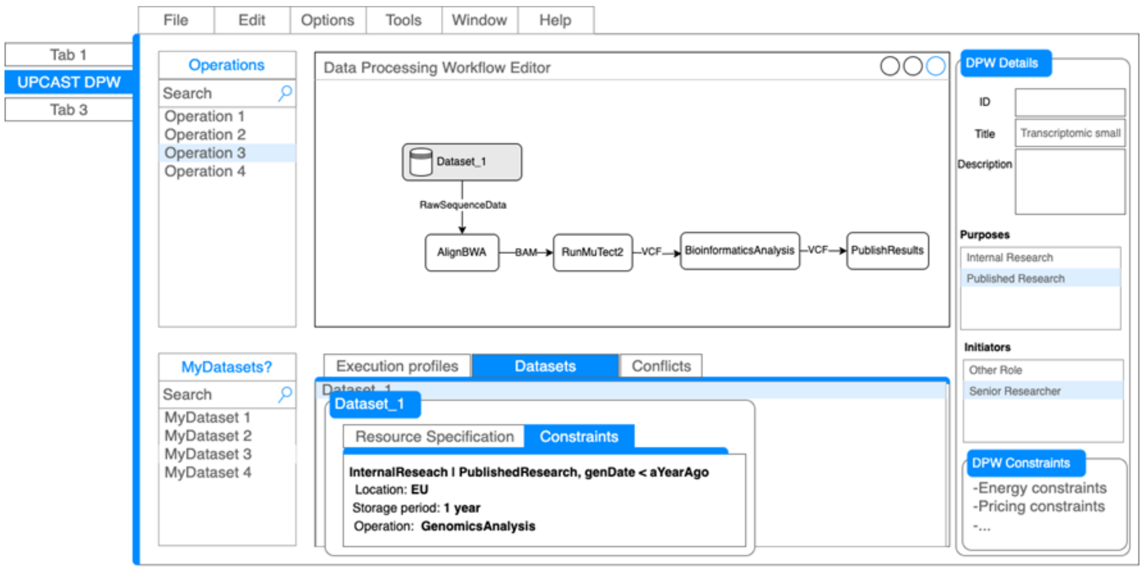
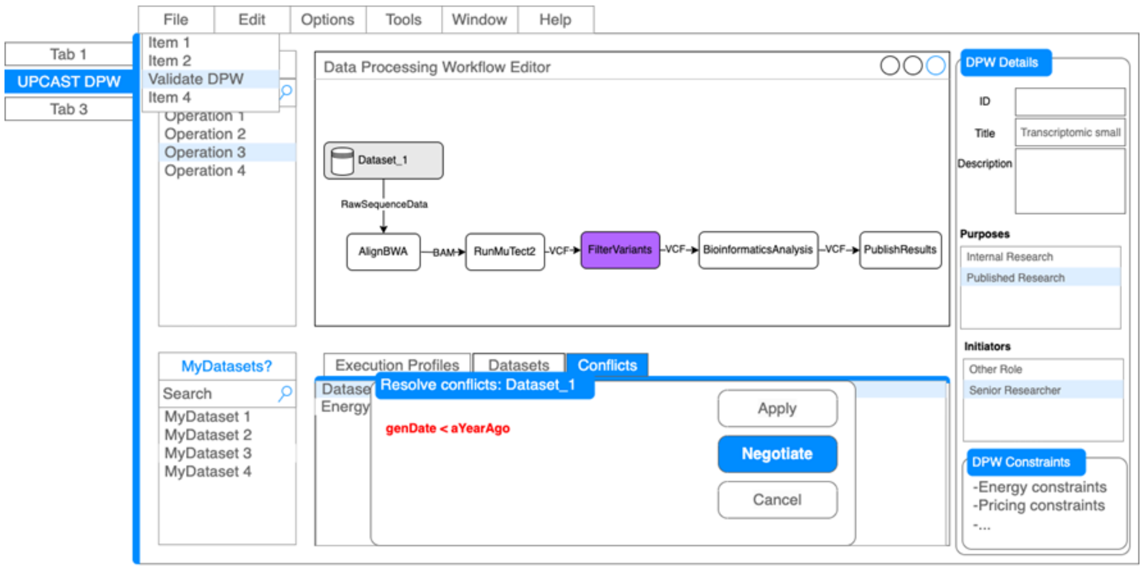


Figure 9: Dataset discovery through goodFlows.



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

Figure 10: Properties of the discovered dataset.



genDate < aYearAgo

Figure 11: Conflict identification prior to negotiation.

Conflict identification between constraints defined by the resource provider and the workflow specification created by the resource consumer comprises another important functionality current added to goodFlows. To this end, the underlying rules' base containing organisational policies, regulatory provisions, etc., is extended with the constraints set by the provider, in order to identify the conflicts and possible resolutions.

Conflicts identification leverages a mechanism for automated workflow model verification against the underlying rules and its re-engineering towards becoming compatible with these rules. This is based on a set of directives, provided by the rule engine; the next two sections outline respectively, the directives and the mechanism for model verification and re-engineering.

4.4.3.3 Compliance directives

The verification of a workflow model with respect to the various underlying constraints is performed based on its ontological representation and on a set of so-called *Compliance Directives*, that indicate the terms under which the workflow in question is compatible with the constraints, and thereby acceptable. Directives are generated through reasoning over the Policy Model reflecting internal policies, regulatory provisions, etc., as well as, in the context of negotiation, the constraints set by the resource provider; its main elements are access control *rules*, used for defining *permissions*, *prohibitions* and *obligations* over *actions*, i.e., structures that, similar to tasks, indicate an operation performed by an actor on an asset [3].

Directives creation takes place on the basis of the pairs of all interacting tasks within the workflow, along with their corresponding interactions (i.e., connecting edges) themselves, what is being referred to as the *Bilateral Associations* (BA) of the workflow. The reason for choosing this pair-wise fragmentation for the initial processing at the level of Directives generation, is that a BA essentially constitutes the elementary unit of flow. Thus, the instructions received are richer in semantics, since it is not only the tasks that matter, but also their interrelations. The main types of Directives considered are:

- *Bilateral Validity Directive (BVD)*: A directive of this type refers to one BA, indicating, for a given purpose and initiator among the specified ones, one valid actor--operation--asset combination for each task and a valid specification of the relationship connecting the two tasks; the latter may refer to the edge as has been defined by the designer, a different edge specification, or even one or more tasks that must be inserted in between the two tasks, so that the control or data flow between them is consistent. All other types of directives presented below refer, in most cases, to a specific BVD, reflecting the fact that requirements and prohibitions may depend on the existence and the different valid specifications of the tasks originally appearing in the workflow.
- *Input Requirement Directive (IRD)*: A task, though being “in principle” accepted, needs to receive some additional input, not included in the BA under consideration. The directive specifies the required input, (optionally) its source, and the task within a valid BA that needs to receive it.
- *Output Requirement Directive (ORD)*: A task specified within a valid BA must provide (some of) its output to a certain task (or structure thereof). An ORD defines the task in a valid BA that must communicate the data, the data themselves and the task structure that must receive them.
- *Task Presence Directive (TPD)*: A task structure must execute, complementing reference BA tasks. If applicable, a TPD also indicates the relative position or data association with respect to the BA task the required one(s) must be found in; for example, a task may require that another has preceded at some point in the workflow.
- *Task Forbiddance Directive (TFD)*: A task must not be executed in the context of a workflow, either at any point or within certain parts of the flow. Each of these directives refers to a task defined by a BVD, specifying the task structure with which the task under consideration is not allowed to coexist, along with their relative position, if applicable.

- *Flow Forbiddance Directive (FFD):* A task is not allowed to have read access to two or more types of information during a single execution instance. Given a valid BA, such a directive prescribes a forbidden additional incoming flow, by specifying the data it is not allowed to receive but also, potentially, a particular task that they must not come from.

All types of directives may optionally be associated with a contextual condition under which the indicated specification, requirement or forbiddance must apply. Furthermore, a precondition or a postcondition may be defined, denoting the fact that said directive is enforceable if a task, or structure thereof, precede, respectively follow.

4.4.3.4 Workflow Verification

On the basis of the above-described directives, the verification of workflows takes place, through a procedure summarised in Figure 12, and briefly explained in what follows.

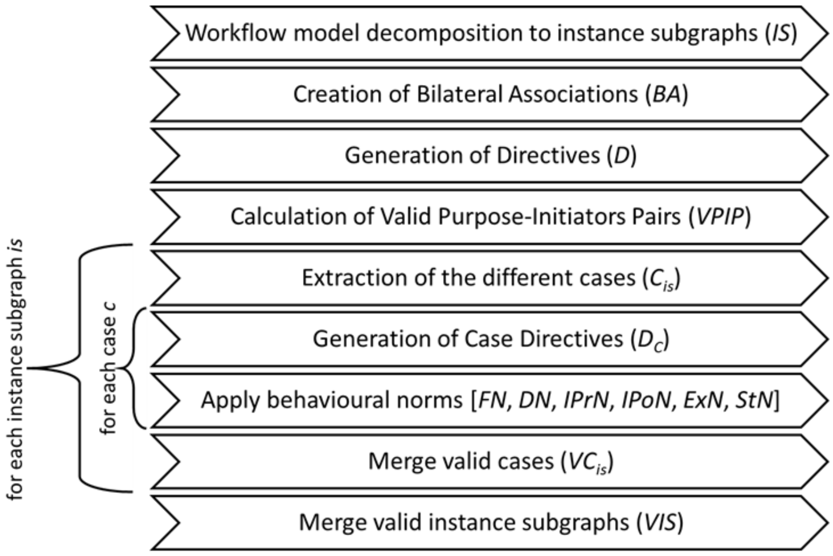

Workflow model decomposition to instance subgraphs (*IS*)
Creation of Bilateral Associations (*BA*)
Generation of Directives (*D*)
Calculation of Valid Purpose-Initiators Pairs (*VPIP*)
Extraction of the different cases (C_{is})
Generation of Case Directives (D_C)
Apply behavioural norms [*FN, DN, IPrN, IPoN, ExN, StN*]
Merge valid cases (VC_{is})
Merge valid instance subgraphs (*VIS*)
for each instance subgraph *is*
for each case *c*

Figure 12: Overview of the verification process.

As mentioned above, the directives are generated on the basis of Bilateral Associations (BA). For the latter to be extracted, first the workflow model is decomposed to *instance subgraphs* (*IS*); these correspond to the different variants the workflow may take, based on the values assigned to all constraints associated with its flow. That is, when the execution of a task implies conditional branching of the consequent flows based on edge constraints, the mutually exclusive constraint spaces of outgoing edges are separately considered, resulting in an execution tree; its leaves represent the space of instance subgraphs. The concept of instance subgraphs has been often used in workflow science, since it makes easier to handle by breaking up the workflow into manageable components [5].

Based on *IS*, the Bilateral Associations (BA) are created, and thereupon verified leveraging the associated functionality offered by the Privacy and Usage Control module, particularly the goodFlows' Policy Decision Point, resulting in the set *D* of Directives. The cartesian combination of purposes and initiators (*PIP*) is then reduced to those pairs that appear to be valid (*VPIP*) according to *D*, whereas *D* drives the verification of each $is \in IS$.

The first step in the verification of an instance subgraph *is* concerns the extraction of the different *cases* (C_{is}), derived from the Bilateral Validity Directives (BVD) associated with *is*. Each case *c* reflects an execution variant of *is*, where each task can be executed in a unique manner and all edges between tasks are the ones prescribed by the corresponding BVDs. To make this clearer, for every BA $<t_i, e_k, t_{i+1}>$, each derived BVD comprises a structure $<t_i^*, e_k^*, t_{i+1}^*>$ where t_i^* and t_{i+1}^* incorporate exactly one execution profile each, and e_k^* is the edge appropriately adapted. It is important to stress that e_k^* may include additional tasks mediating t_i^* and t_{i+1}^*; this is often the case, e.g., with tasks performing data anonymisation or encryption. Eventually, each case *c* is a projection of *is*, according to a valid combination of $<t_i^*, e_k^*, t_{i+1}^*>$ structures, derived from the BVDs.

The generation of cases C_{is} is followed by their verification and appropriate transformation, considering also the rest of Directives. In this context, the behavioural *norm* of each task *t* in a case *c* is extracted by the directives D_c pertaining to the case. Essentially, norms comprise groups of compliance patterns that span across all Directives' types and can be verified together for *t*.

Forbiddance Norms (*FN*) reflect requirements implied by TFD and FFD. Provisions described by *Direct Norms* (*DN*) concern tasks that should be present in the workflow directly connected with *t* via an edge, either incoming or outgoing. On the other hand, *Indirect Pre-* (*IPrN*) and *Indirect Post-Norms* (*IPoN*) indicate tasks that should precede, respectively follow, the execution of *t*, with relative position other than direct connection, whereas an *Existence Norm* (ExN) implies the need for a task to exist in the workflow at any position. *State Norms* (*StN*), derived from BVD, reflect requirements related to data state. Finally, norms can be *conditional* or *definite*, depending on whether the corresponding Directives are associated with pre- and/or post-conditions, or not.

All tasks comprising the case are verified against the associated norms. Therefore, tasks are topologically sorted [6], providing for both forward and backward traversal, and the application of the norms for the progressive transformation of the case *c* takes place, resulting to its verified version *vc* (or to failure). The procedure begins and finishes with the application of forbiddance provisions; the reason is that, on the one hand, the case may be rejected at the very beginning due to some conflict implied by *FN*, while, on the other hand, checking against forbiddances is deemed necessary following any transformations that may have happened due to the application of the other types of norms.

The latter takes place in three phases; first, the definite provisions are applied, followed by the conditional ones. In each phase, direct norms precede indirect pre- and post-norms; the reason why indirect norms are not applied together, as is the case with direct, is that post- norms require traversing the tasks of the case in a backward manner. Third, norms related with data state are applied, in order to perform the corresponding verification and transformation after all other norms have been applied and, consequently, all task additions and flow modifications they imply have already been enforced.

After this loop has been executed over all cases C_{is}, the cases VC_{is} found to be valid are being merged, providing the verified instance subgraph *vis*; merging concerns the aggregation of the tasks representing the same activity in the different cases, and the unification of the corresponding edges. Similarly, when verification of all instance subgraphs is complete, the verified ones (*VIS*) are merged providing the final verified workflow model WM_V. In other words, similarly to the decomposition of the initial workflow model to instance subgraphs and cases, the final WM_V is assembled from its elementary parts, i.e., its cases and verified instance subgraphs, into a unified specification. Intuitively, in order for the workflow verification to be successful, there should be at least one verified case *vc* resulting from the procedure.

The basic scheme presented above has some variants related to the repetitive execution of certain parts, so that a case, a subgraph, or the model as a whole, to be verified again; so that to capture potential privacy flaws that the modification may have introduced. For instance, two new tasks, introduced during verification, may conflict with each other, which cannot be captured by the initial directives. Hence, repetition of some procedures is necessary, until the workflow "converges" to a definitive structure.

4.5 Negotiation Protocol

The UPCAST Negotiation Protocol is a set of interactions between a provider and a consumer that establishes an UPCAST contract based on an ODRL agreement. The UPCAST negotiation information model is presented in Figure 13.

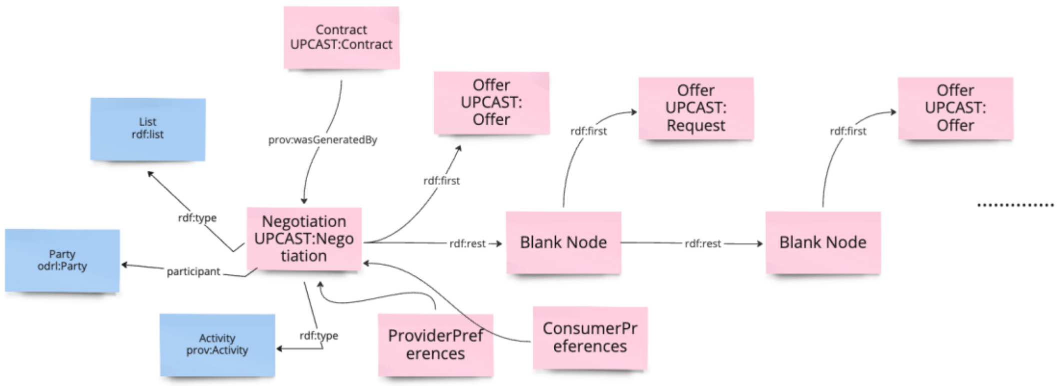
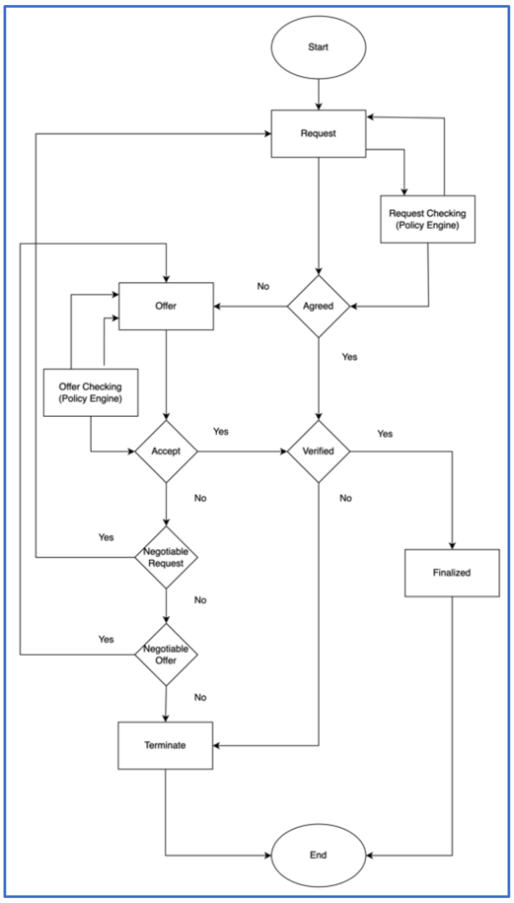


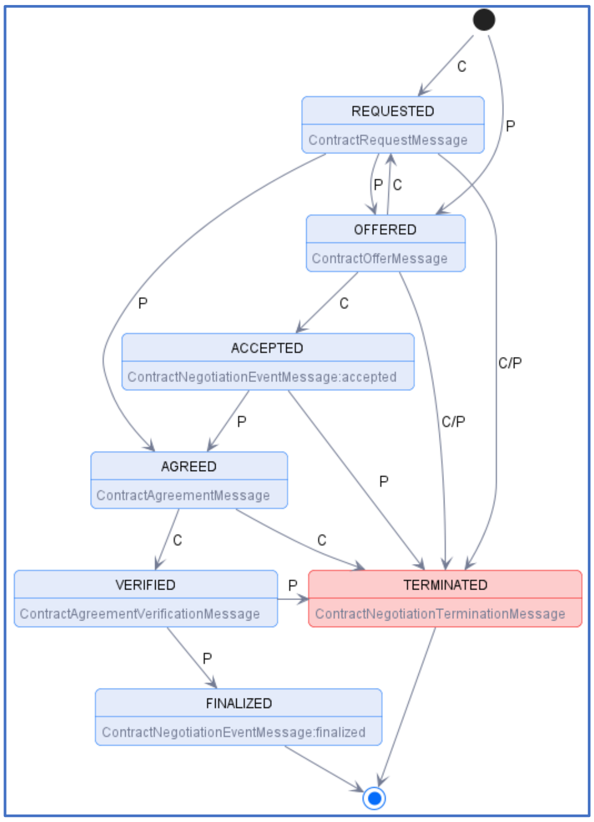
Figure 13: UPCAST Negotiation Data Model.

Once the provider and the consumer have been matched for negotiation and agreement, the UPCAST Negotiation Plugin verifies the DPW and consumer's intentions against the provider's constraints, legal constraints, organization-specific policies, pricing, and environmental impact constraints. If no conflict is identified, an agreement is automatically reached. Otherwise, the system highlights the conflicts and tries to find an optimal offer/counter offer, which is then sent back to the consumer. Following this, a negotiation process is initiated, consisting of a sequence of counter-offers exchanged between the provider and the consumer. The consumer may choose one of the alternatives presented by the system, manually edit the terms, or request a new alternative. This counter-offer can then be accepted or modified by the provider. If the provider accepts the counter-offer, an agreement is reached, and the system proceeds to contracting. If not, the provider must present a counter-offer, assuming the role of the consumer in the previous step. This back-and-forth continues until the provider agrees with an offer. Ultimately, the provider has the final say on whether the negotiation proceeds by accepting, rejecting, or sending another counter-offer. Through the negotiation and contracting plugin UI, the provider defines the negotiation range for each statement in the resource specification, while the consumer may also fine-tune the DPW specification to reflect their negotiation preferences (explained in Section 4.5.2). The case concludes with the generation of machine-readable and natural language contracts if an agreement is reached. In any case, the negotiation outcome is presented to both parties.

UPCAST negotiation builds upon IDSA standards; Figure 14 (a) shows the UPCAST negotiation flowchart, which is compatible with the state machine of the IDSA Contract Negotiation Protocol represented in Figure 14 (b).



(a) UPCAST Negotiation Flowchart.


REQUESTED
ContractRequestMessage
OFFERED
ContractOfferMessage
ACCEPTED
ContractNegotiationEventMessage:accepted
AGREED
ContractAgreementMessage
VERIFIED
ContractAgreementVerificationMessage
TERMINATED
ContractNegotiationTerminationMessage
FINALIZED
ContractNegotiationEventMessage:finalized
C
P
C/P

(b) IDSA Contract Negotiation Protocol State Machine.

Figure 14: Negotiation Flowchart and State Machine.

4.5.1 Negotiation process

The negotiation process detailed below is fully compatible with the IDSA negotiation protocol. The following abbreviations are used in the presentation:

- RP: resource provider,
- RC: resource consumer,
- DPW: Data Process Workflow

Preconditions:

RPs have submitted their data source and primary ODRL offers in a data market place.

Description:

1. RC initiates a negotiation through one of the following scenarios:
    - Scenario 1: Using goodFlows, after defining a DPW, the consumer can generate an UPCAST request, which includes an ODRL request based on internal policies, a resource specification, a natural language part, and negotiation preferences;
    - Scenario 2: The consumer searches the Data Catalog for a resource. Upon finding a suitable resource, the consumer fetches the resource specification and its ODRL offer from the Data Catalog. The consumer then generates an UPCAST request, which includes an ODRL request, a resource specification, a DPW, a natural language part, and negotiation preferences;

    (The UPCAST request is explained in Section 4.4)
2. Before sending the request to the RP, the RC's request should be examined for conflicts:
    a. Conflicts within the ODRL request and the DPW are detected by Policy Engine.
    b. Conflicts in resource specification and natural language part are found by Resource Description Conflict Resolution Engine and NLP Conflict Resolution Engine, respectively.
3. If any conflict is detected, RC may:
    a. Revise the request and resend it for conflict checking.
    b. Confirm the existing request, despite its conflict(s).
4. RP may:
    a. Agree to the UPCAST request, in which case the process continues from step 9.
    b. Release a counteroffer called an UPCAST offer, which includes an ODRL offer, a resource specification, a DPW pattern, a natural language part, and their negotiation preferences.

    (The UPCAST offer is explained in Section 4.3).
5. If the RP releases an UPCAST offer, the offer should be examined for conflicts:
    a. Conflicts in the ODRL offer and DPW pattern are detected by the Policy Engine.
    b. Conflicts in the resource specification and natural language part are found by the Resource Description Conflict Resolution Engine and the NLP Conflict Resolution Engine, respectively.
6. If any conflict is detected, the RP may:
    a. Revise the offer and resend it for conflict checking.
    b. Confirm the existing offer despite its conflicts.
7. The RC may:
    a. Accept the offer and send an acceptance message to the RP.
    b. Generate a new UPCAST request, in which case the process continues from step 2.
8. If the RC accepts the offer, the RP may respond with an agreement.

9. RC may verify the agreement.
10. An UPCAST contract is generated, including the final version of the ODRL agreement, resource specification, DPW, and natural language part. Metadata such as the start date and the validation period of the contract is also added.
11. RC signs the contract by adding the date and his dpv:DigitalSignature;
12. RP signs the contract by adding the date and his dpv:DigitalSignature;

Postconditions:

The UPCAST contract is finalised.

It should be noted that

– RC can terminate the negotiation at steps 3, 7, and 9.
– RP can terminate the negotiation at steps 4, 6, and 8.

4.5.2 Negotiation Terms and Preferences

In a negotiation process between a data provider and a data consumer for a specific dataset, both parties typically have preferences and objectives that they aim to achieve. Negotiation preferences can vary depending on the specific context and requirements of the dataset involved.

Both providers and consumers may specify their preferences before starting the negotiation. They define some conditions and also some ranges to specify the upper and the lower bounds of some variables. They may inform the other party of some or all their preferences; it depends on the game theory model that is applied for negotiation.

Following are some common negotiation preferences for both the data provider and the data consumer, along with examples.

Data Provider's Preferences:

1. Fair Compensation/Price: The data provider may seek fair compensation for providing access to the dataset. This compensation could be monetary or non-monetary, such as reciprocal access to services. In the case of monetary compensation, the provider may define a minimum value that he will accept.

   Example: The data provider may offer a dataset for a lower fee if the data consumer requests usage of the dataset for research purposes rather than general purposes.

   Example: The data provider sets a minimum for the price, and if the consumer makes a request of less than that amount, the provider leaves the negotiation.

2. Data Access Restrictions: The data provider may have preferences regarding how the dataset can be accessed, used, and shared to protect its property rights and ensure data security and privacy. Some related ranges are:
   - Access control; whether it is full data access, limited data access or anonymized data access.
   - Conditional access; whether it is time-bound access or purpose-bound access.
   - Data usage restriction; it can be single-use, multi-use, or aggregate-use.
   - Privacy and security measures; for example, data masking, encryption, access logs, and differential privacy.

- Compliance requirements; it can be regularity compliance or audit rights.
- Monitoring and reporting; for instance, usage reports or breach notification.

Example: The data provider may require the data consumer to agree to specific usage restrictions and data protection measures, such as: *"Anonymize the data before sharing to a third-party"*, *"Do not share attribute X and attribute Y, simultaneously"*, *"Use data whose collected date is between X and Y"*.

3. Environmental Impact: The data provider may have preferences regarding the environmental impact of data processing workflows that use their data.

   Example: The data provider may require the data processing workflow of a data consumer to have a carbon consumption under a certain range, but is willing to negotiate for a higher carbon consumption.

Data Consumer's Preferences:

1. Cost-effectiveness: The data consumer seeks to obtain the dataset at a reasonable cost or within their budget constraints to ensure the overall viability of their project or business objectives.

   Example: the data consumer may specify the maximum value that he will accept to pay for data.

2. Data Relevance and Quality: The data consumer's primary preference may be to access a dataset that meets their specific needs and requirements. They prioritize data relevance, accuracy, and reliability.

   Example: the data consumer may require a specific collection of the attributes such as, daily step counts, heart rate, calories burned, and workout duration in Nissatech pilot.

3. Data Access and Usage Rights: The data consumer may also have preferences regarding the terms of data access, usage, and redistribution rights to ensure flexibility and compatibility with their intended use cases.

   Example: A research institution negotiating for access to scientific data may require non-exclusive usage rights to analyze and publish research findings derived from the dataset. The third parties to which these research findings will be shared are usually determined post-negotiation at the consumer's discretion. These third parties could be other research organizations or different entities, depending on the consumer's preference.

4. Timeliness and Availability: Timely access to the dataset can be crucial for the data consumer's project timelines and deliverables. They prioritize negotiating agreements that ensure prompt access to the dataset.

   Example: The data consumer may prioritize agreements that guarantee immediate data access and updates.

These negotiation preferences serve as guiding principles for both the data provider and the data consumer during contract negotiations, helping them reach mutually beneficial agreements that address their respective interests and objectives.

Examples of Specific Negotiations

Provider-Defined:

- The producer decides to provide only anonymized data access to ensure privacy.
- The producer allows access to the data for six months only.
- The producer mandates that all sensitive data fields must be masked before sharing.
- The producer requires the consumer to comply with GDPR regulations.

Consumer-Defined:

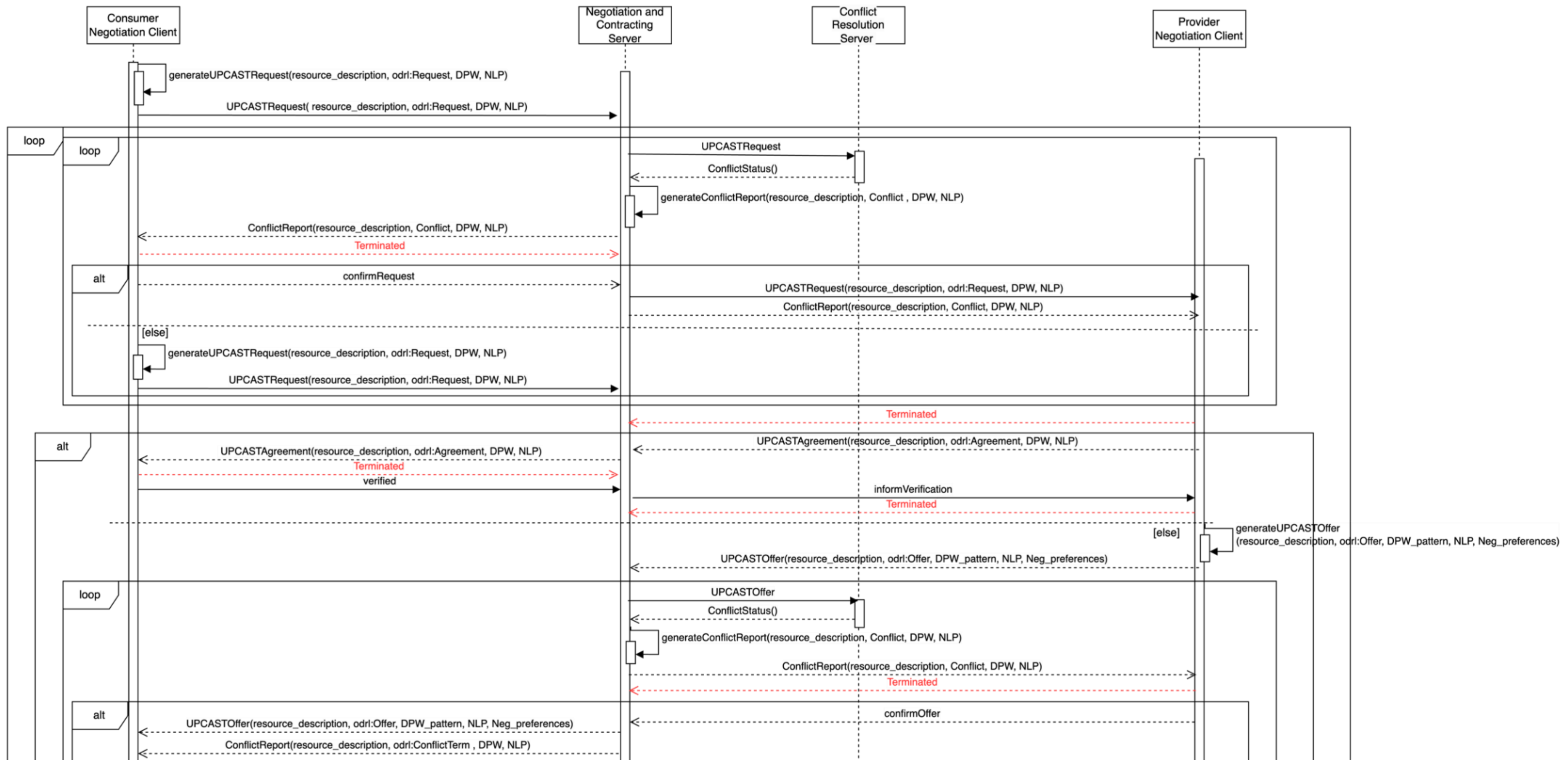
- The consumer specifies that they need the data for developing a new machine learning model.
- The consumer agrees to pay a specified fee for data access.
- The consumer proposes that any new datasets derived from the original data will be shared back with the producer.
- The consumer suggests a revenue-sharing model where the producer receives a percentage of profits generated from the data usage.

4.5.3 Negotiation sequence diagrams

This section presents the sequence diagrams for the negotiation process.

First, the negotiation process in examined, where resource providers and resource consumers act as clients, while negotiations are managed by a central negotiation and contracting server and conflict resolution is handled by the Conflict Resolution Engine, which consists of three components: the Policy Engine, the NLP Conflict Resolution Engine, and the Resource Description Conflict Resolution Engine. Figure 15 illustrates the negotiation sequence diagram. As depicted, consumers always initiate a negotiation by sending a request. Resource providers then have the option to agree to the request, send back a counteroffer, or terminate the negotiation. If a provider responds to a consumer's request with an offer, the consumer can choose to accept the offer, send a new request, or terminate the entire negotiation. Ultimately, when an agreement is reached, a contract is generated and sent to the parties for their signatures.

Next, the negotiation process from the perspective of the negotiation API is illustrated. In this scenario, both resource providers and resource consumers utilize the API to initiate, manage, finalize, or terminate negotiations. The API is responsible for tracking all active negotiations, allowing both parties to access up-to-date information. Additionally, the API handles conflict resolution and oversees the contracting process, ensuring seamless and efficient negotiation /management. Figure 16 shows the API-based version of the negotiation sequence diagram.


Consumer Negotiation Client
Negotiation and Contracting Server
Conflict Resolution Server
Provider Negotiation Client
generateUPCASTRequest(resource_description, odrl:Request, DPW, NLP)
UPCASTRequest(resource_description, odrl:Request, DPW, NLP)
loop
loop
UPCASTRequest
ConflictStatus()
generateConflictReport(resource_description, Conflict , DPW, NLP)
ConflictReport(resource_description, Conflict, DPW, NLP)
Terminated
alt
confirmRequest
UPCASTRequest(resource_description, odrl:Request, DPW, NLP)
ConflictReport(resource_description, Conflict, DPW, NLP)
[else]
generateUPCASTRequest(resource_description, odrl:Request, DPW, NLP)
UPCASTRequest(resource_description, odrl:Request, DPW, NLP)
Terminated
alt
UPCASTAgreement(resource_description, odrl:Agreement, DPW, NLP)
UPCASTAgreement(resource_description, odrl:Agreement, DPW, NLP)
Terminated
verified
informVerification
Terminated
[else]
generateUPCASTOffer (resource_description, odrl:Offer, DPW_pattern, NLP, Neg_preferences)
UPCASTOffer(resource_description, odrl:Offer, DPW_pattern, NLP, Neg_preferences)
loop
UPCASTOffer
ConflictStatus()
generateConflictReport(resource_description, Conflict, DPW, NLP)
ConflictReport(resource_description, Conflict, DPW, NLP)
Terminated
alt
confirmOffer
UPCASTOffer(resource_description, odrl:Offer, DPW_pattern, NLP, Neg_preferences)
ConflictReport(resource_description, odrl:ConflictTerm , DPW, NLP)

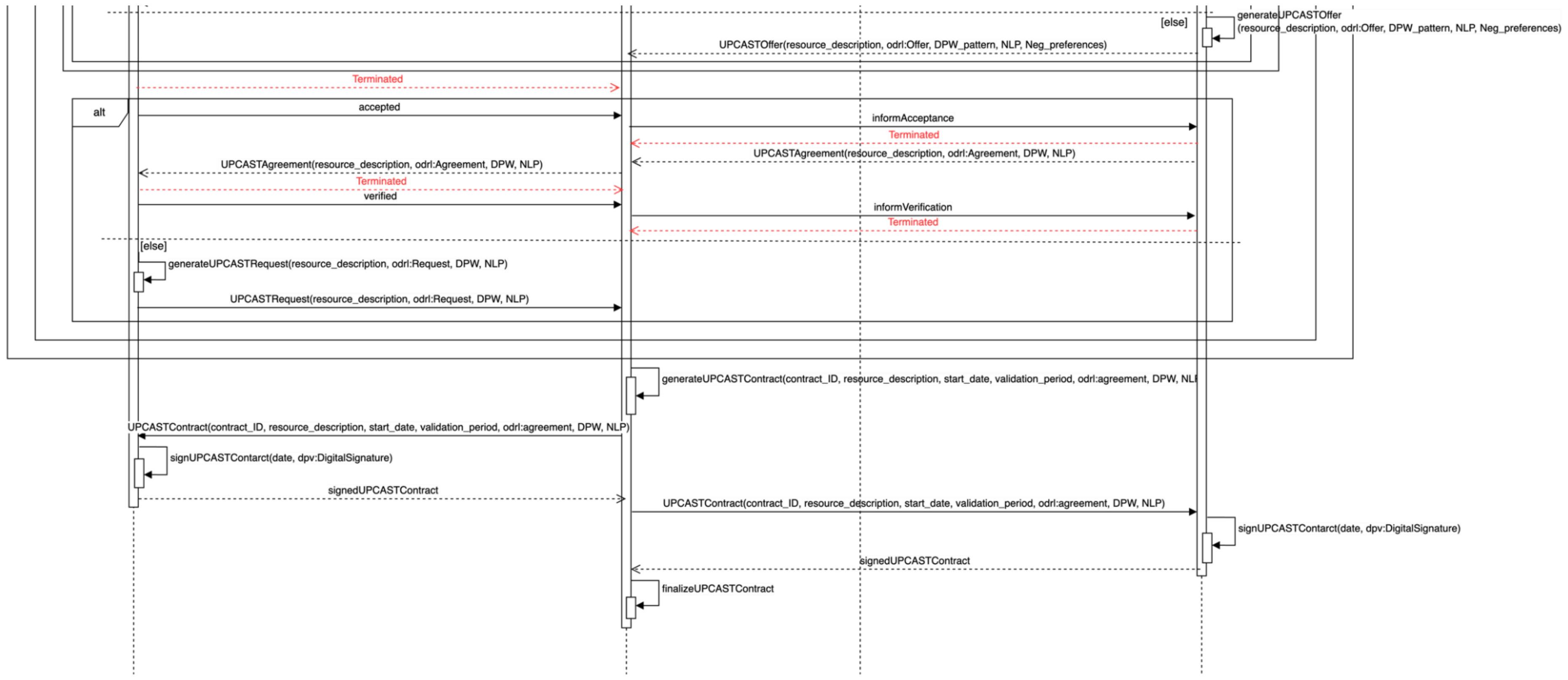

generateUPCASTOffer
(resource_description, odrl:Offer, DPW_pattern, NLP, Neg_preferences)
[else]
UPCASTOffer(resource_description, odrl:Offer, DPW_pattern, NLP, Neg_preferences)
Terminated
alt
accepted
informAcceptance
Terminated
UPCASTAgreement(resource_description, odrl:Agreement, DPW, NLP)
UPCASTAgreement(resource_description, odrl:Agreement, DPW, NLP)
Terminated
verified
informVerification
Terminated
[else]
generateUPCASTRequest(resource_description, odrl:Request, DPW, NLP)
UPCASTRequest(resource_description, odrl:Request, DPW, NLP)
generateUPCASTContract(contract_ID, resource_description, start_date, validation_period, odrl:agreement, DPW, NL
UPCASTContract(contract_ID, resource_description, start_date, validation_period, odrl:agreement, DPW, NLP)
signUPCASTContarct(date, dpv:DigitalSignature)
signedUPCASTContract
UPCASTContract(contract_ID, resource_description, start_date, validation_period, odrl:agreement, DPW, NLP)
signUPCASTContarct(date, dpv:DigitalSignature)
signedUPCASTContract
finalizeUPCASTContract

Figure 15: Simplified Sequence Diagram.

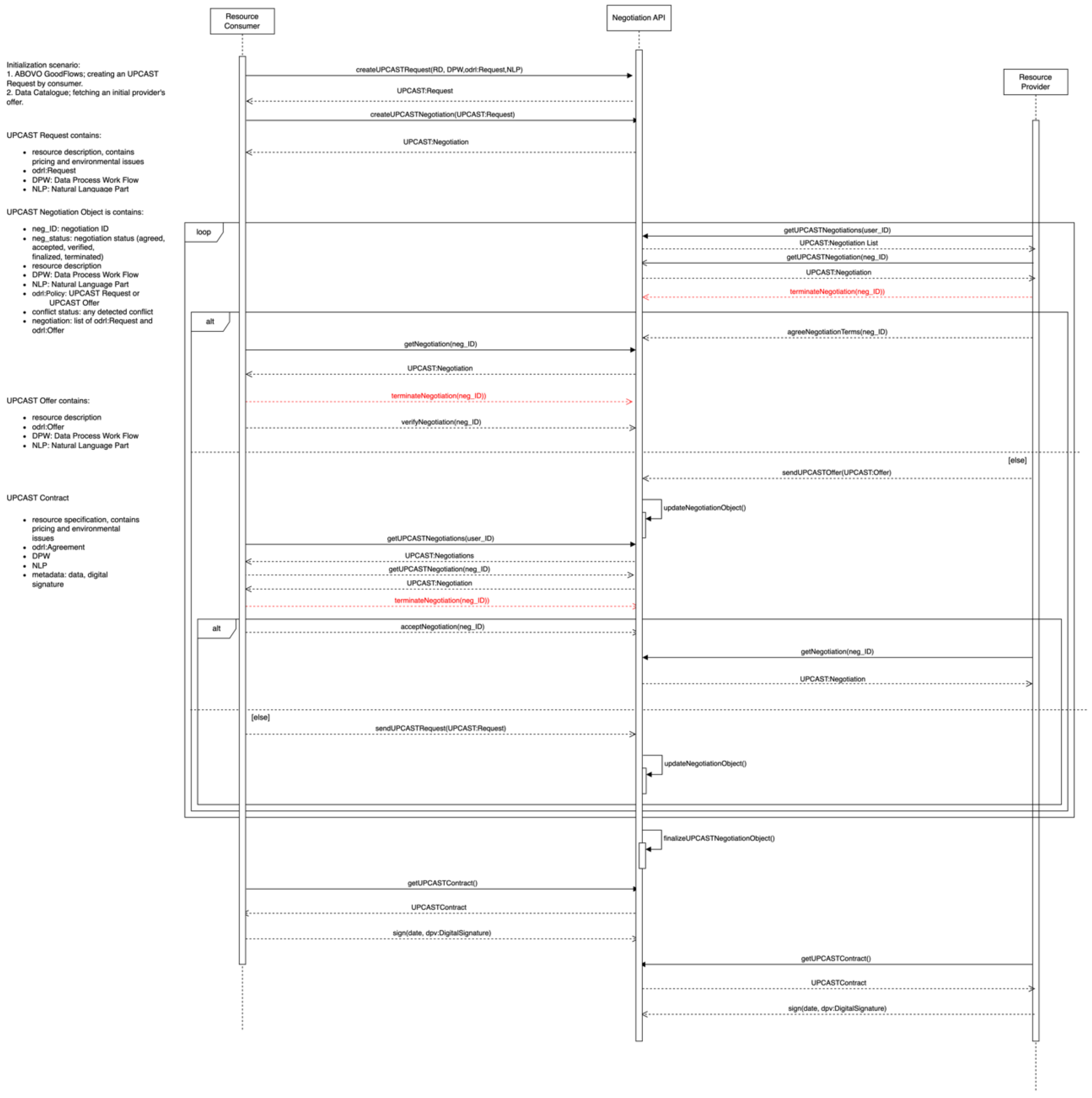

Resource Consumer
Negotiation API
Resource Provider
Initialization scenario:
1. ABOVO GoodFlows; creating an UPCAST Request by consumer.
2. Data Catalogue; fetching an initial provider's offer.
UPCAST Request contains:
• resource description, contains pricing and environmental issues
• odrl:Request
• DPW: Data Process Work Flow
• NLP: Natural Language Part
UPCAST Negotiation Object is contains:
• neg_ID: negotiation ID
• neg_status: negotiation status (agreed, accepted, verified, finalized, terminated)
• resource description
• DPW: Data Process Work Flow
• NLP: Natural Language Part
• odrl:Policy: UPCAST Request or UPCAST Offer
• conflict status: any detected conflict
• negotiation: list of odrl:Request and odrl:Offer
UPCAST Offer contains:
• resource description
• odrl:Offer
• DPW: Data Process Work Flow
• NLP: Natural Language Part
UPCAST Contract
• resource specification, contains pricing and environmental issues
• odrl:Agreement
• DPW
• NLP
• metadata: data, digital signature
createUPCASTRequest(RD, DPW,odrl:Request,NLP)
UPCAST:Request
createUPCASTNegotiation(UPCAST:Request)
UPCAST:Negotiation
loop
getUPCASTNegotiations(user_ID)
UPCAST:Negotiation List
getUPCASTNegotiation(neg_ID)
UPCAST:Negotiation
terminateNegotiation(neg_ID))
alt
agreeNegotiationTerms(neg_ID)
getNegotiation(neg_ID)
UPCAST:Negotiation
terminateNegotiation(neg_ID))
verifyNegotiation(neg_ID)
[else]
sendUPCASTOffer(UPCAST:Offer)
updateNegotiationObject()
getUPCASTNegotiations(user_ID)
UPCAST:Negotiations
getUPCASTNegotiation(neg_ID)
UPCAST:Negotiation
terminateNegotiation(neg_ID))
alt
acceptNegotiation(neg_ID)
getNegotiation(neg_ID)
UPCAST:Negotiation
[else]
sendUPCASTRequest(UPCAST:Request)
updateNegotiationObject()
finalizeUPCASTNegotiationObject()
getUPCASTContract()
UPCASTContract
sign(date, dpv:DigitalSignature)
getUPCASTContract()
UPCASTContract
sign(date, dpv:DigitalSignature)

Figure 16: Negotiation Sequence Diagram.

4.6 Contract Data Model

Upcast contract is the result of a successful negotiation between a provider and a consumer when they reach a complete agreement. It contains several main parts. Firstly, an ODRL Agreement makes the foundation of the contract. Then, an UPCAST contract utilises the IDSA contract schema. In addition, it requires DPW to specify contract-related activities and sequences and Natural Language Part. Eventually, some metadata is needed to finalise the contract; for example, a contract must contain signatures of participants, related dates, etc. Figure 17 shows the information model of the UPCAST contract.

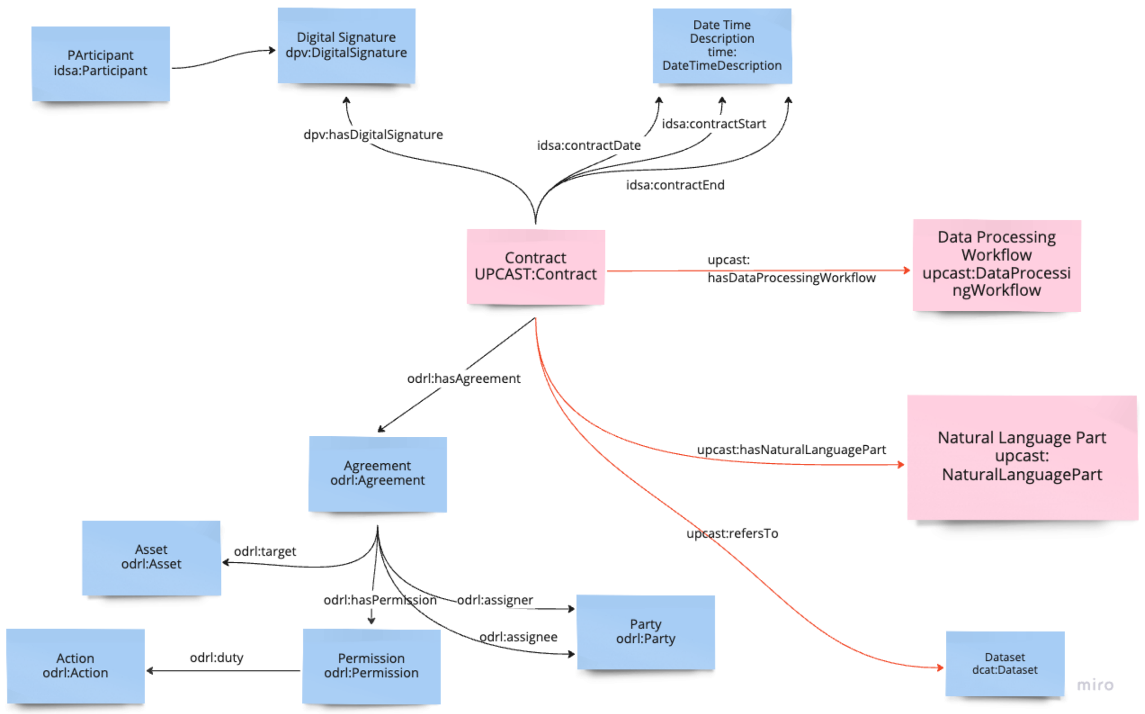


Figure 17: UPCAST Contract Data Model.

4.6.1 ODRL Agreement

An Agreement is a concrete policy associated with a specific dataset that has been accepted by both the provider and consumer parties. An Agreement is a result of a negotiation and is associated with *exactly one* Dataset.

An ODRL Agreement Policy *must* contain at least one Permission or Prohibition rule, a party with assigner function, and a party with assignee function (in the same permission or prohibition). The Agreement Policy will grant the terms of the policy from the assigner to the assignee. It must also contain a target property. The complete ODRL Agreement is sent through a Contract Agreement Message by a provider when it agrees to a contract.

sent by	Data Provider	
format	ODRL, IDSA	
properties	Odrl:Rule	Permission with none, one, or more duties as the property of the Permission. Prohibition with none, one, or more remedies as the property of the Prohibition.
	odrl:uid	It identifies the policy.
	odrl:Party idsa:Participants	It defines assignee and assigner; the contents of these properties are a unique identifier of the Agreement parties. These identifiers are not necessarily the same as the identifiers of the Parties negotiating the contract.

	odrl:Asset	It defines as target; None of its Rules, however, must have any odrl:target attributes to prevent inconsistencies with the ODRL inferencing rules for compact policies.
	[odrl:Constraint]	It defines any constraint on the rule based on the usage policies. Logical constraint can be also included.
	[odrl:Profile]	It defines an ODRL rule that this agreement complies with.

The ODRL agreement may also have none, one, or many *profile* values to identify the ODRL Profile that this Agreement conforms to. It may have none, one, or many *inheritFrom* values. It may have none or one *conflict* values (of type ConflictTerm) for Conflict Strategy Preferences indicating how to handle Policy conflicts.

An ODRL agreement may also declare properties which are shared and common to all its Rules. Specifically, action properties, sub-properties of relation (such as target), and sub-properties of function (such as assigner and assignee).

An ODRL agreement must either:

- Only use terms defined in the *ODRL Core Vocabulary* [odrl-vocab], or
- Use an ODRL Profile that declares the supported vocabulary used by expressions in the Policy.

4.6.2 Data Processing Workflow

An ODRL Agreement does not contain the sequence and the order of the actions; thus, DPW is needed to specify it. Each UPCAST contract contains a specific DPW in which the exact sequence of the actions in ODRL agreement is defined.

4.6.3 Dataset

Based on the UPCAST contract, a resource specification is created and the resource is annotated with basic metadata, using UPCAST vocabulary and domain-specific vocabularies. Pricing and environmental issue related to the contract is identified here.

4.6.4 Natural Language component

Users must be able to generate contracts that contain and use boilerplate text whenever the contract contains clauses that can only be expressed through natural language. Therefore, UPCAST Negotiation and Contracting plugin has been developed to automatically generate contracts in both machine-readable formats, under standards such as the Open Digital Rights Language (ODRL), and natural language using boilerplate text or even Large Language Models (LLMs).

4.6.5 MetaData

An UPCAST Contract *must* contain timestamps that define contract sign date-time, contract start date-time, and contract end date-time. The agreed version of an UPCAST Contract will also be signed by both the provider and the consumer; thus, the contract *must* include a signature. A DPV Digital Signature which is an expression and authentication of identity through digital information containing cryptographic signatures will be used for this purpose.

4.6.6 Human Readable Contract Terms

The UPCAST negotiation and contracting plugin generates outcomes that include both machine-readable and human-readable contracts when an agreement is reached between the provider and consumer. The human-readable component of these contracts is essential for ensuring that all parties fully understand the terms and conditions, facilitating transparency and reducing the likelihood of disputes. This component provides a clear, concise summary of the agreement in natural language, making it accessible to individuals regardless of their technical expertise.

Leveraging Large Language Models (LLMs) significantly enhances the creation of human-readable contract terms. LLMs can automatically generate accurate and contextually relevant summaries of complex contractual agreements. By interpreting and translating the technical and legal jargon into plain language, LLMs ensure that every stakeholder, from legal professionals to non-specialists, can easily comprehend the terms. This capability not only streamlines the negotiation process but also promotes fairness and clarity, ensuring that all parties have a shared understanding of the agreement.

4.7 Legal Terms

The following legal terms are critical for the Negotiation and Contracting plugin in UPCAST.

Data Ownership and Intellectual Property Rights

- Clearly define the ownership of data and any intellectual property rights associated with the data and derived works; idsa:Participants, odrl:Party in ODRL Agreement, and dpv:DigitalSignature.
- Specify conditions under which data can be shared, licensed, or sold, including any rights to modifications or improvements made to the data by the consumer; odrl:Constraints and odrl:duty in ODRL Agreement.

Permissions and Restrictions

- Permissions: Outline specific permissions granted to the data consumer, such as access rights, usage rights, and sharing rights; odrl:Permission in ODRL Agreement.
- Prohibitions: Define explicit prohibitions, such as restrictions on data usage, limitations on further dissemination, and any activities that are expressly forbidden; odrl:Prohibition in ODRL Agreement.
- Obligations: Specify obligations imposed on the data consumer, such as requirements to maintain data confidentiality, report data usage, and comply with relevant laws and regulations; odrl:Duty in ODRL Agreement.

Data Processing Workflow

- Description: Detailed steps of how the data will be collected, processed, stored, and used.
- Technical Requirements: Specifications that must be followed during data processing.

Privacy and Data Protection

- Ensure compliance with data protection laws such as GDPR (General Data Protection Regulation).
- Define the measures to be taken to protect personal data, including anonymization, encryption, and access controls; odrl:Action in ODRL Agreement.
- Include clauses that specify the data producer's and consumer's responsibilities in protecting personal data and handling data breaches; odrl:duty in ODRL Agreement.

Data Usage and Processing

- Detail the specific purposes for which the data can be used and any processing activities that are permitted; odrl:Constraint in ODRL Agreement.
- Include conditions for data minimization, ensuring that only the necessary data is used for the specified purposes.

Liability and Indemnification

- Establish the liabilities of each party in case of data misuse, breaches, or non-compliance with the contract terms; odrl:remedy and odrl:consequences in ODRL Agreement.

Compliance and Auditing

- Outline the requirements for compliance with applicable laws, industry standards, and best practices.
- Include provisions for regular audits to ensure adherence to the terms of the contract and the effectiveness of data protection measures.

Dispute Resolution

- Specify the mechanisms for resolving disputes that may arise from the data sharing agreement, such as mediation, arbitration, or litigation.
- Include the jurisdiction and governing law that will apply to the contract.

Termination and Exit Strategy

- Specify the contract validity duration by start time and end time/duration; time:DateTimeDescription.
- Define the conditions under which the contract can be terminated by either party.
- Include provisions for the return or destruction of data upon termination of the agreement.

Natural Language Component

- Define the Natural Language representation of the contract align with the legal terms; for example:
  - "The Data Producer retains all rights, title, and interest in and to the data provided under this agreement. The Data Consumer is granted a non-exclusive, non-transferable license to use the data solely for the purposes specified herein."
  - "The Data Consumer agrees to process personal data in compliance with GDPR and to implement appropriate technical and organizational measures to protect the data against unauthorised access, disclosure, alteration, or destruction."

  - "The Data Consumer is permitted to use the data for research and analysis purposes only. Any further distribution or commercial use of the data is strictly prohibited unless explicitly authorised by the Data Producer."
  - "The Data Consumer shall be liable for any damages resulting from the misuse of the data and agrees to indemnify the Data Producer against any claims, damages, or liabilities arising from such misuse."
  - "This agreement may be terminated by either party with 30 days written notice. Upon termination, the Data Consumer shall cease all use of the data and, at the Data Producer's discretion, either return or destroy all copies of the data."

By incorporating these legal terms and examples into the negotiation and contracting process, UPCAST ensures that data sharing is conducted in a secure, compliant, and mutually beneficial manner.

5 Data Exchange and Execution

This chapter gives the details of the Data Exchange and the dataset execution functions of the UPCAST platform. Data Exchange relates to functions for the secure transfer of the dataset from the provider to the consumer. These functions are implemented by the Secure Data Delivery plugin that is detailed in section 5.1. Section 5.2 gives the details of the execution environments that can be used in UPCAST for the execution of the project's pilots. As all pilots plan to use the Nextflow system for their workflows, an overview of Nextflow is given in section 5.2.2. Finally, section 5.2.3 gives an overview of SIMPIPE, which is an alternative execution environment that may be used for the execution of consumer workflows.

5.1 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 1.

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[8] 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,

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

- 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.

5.1.1 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 18, 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.

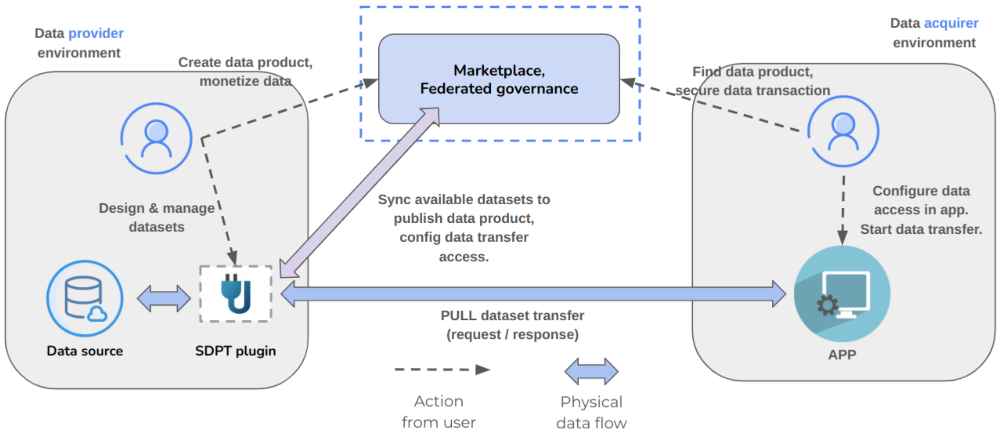

Data provider environment
Create data product, monetize data
Marketplace, Federated governance
Data acquirer environment
Find data product, secure data transaction
Design & manage datasets
Sync available datasets to publish data product, config data transfer access.
Configure data access in app. Start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

Figure 18: One to one data exchange scenario.

One to many data exchange

In this scenario, depicted in Figure 19, 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.

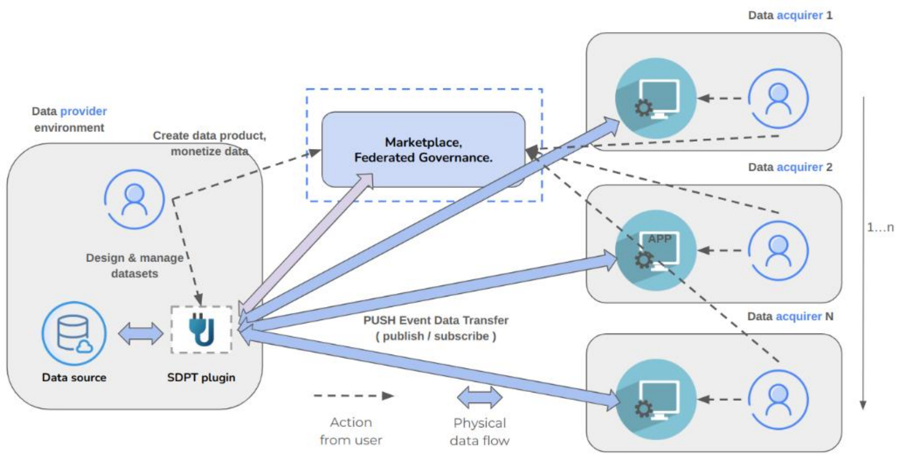

Data provider environment
Create data product, monetize data
Marketplace, Federated Governance.
Design & manage datasets
Data source
SDPT plugin
PUSH Event Data Transfer (publish / subscribe)
Action from user
Physical data flow
Data acquirer 1
Data acquirer 2
Data acquirer N
APP
1...n

Figure 19: One to many data exchange scenario.

5.1.2 Capabilities

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

Table 2: 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

5.1.3 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 20 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.

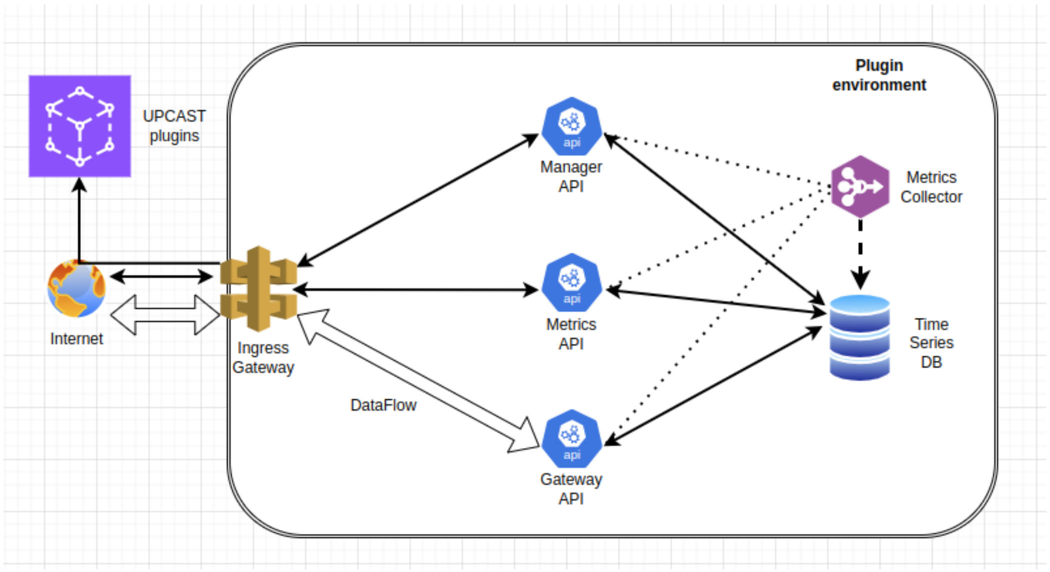


Figure 20: Safe data delivery Plugin Architecture.

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.

5.1.4 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 Chapter 6.

5.2 UPCAST 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 Nextflow management system and the SIMPIPE execution environment.

5.2.1 Pilot Execution Environments

This section gives the details of the execution environments that may be used in UPCAST for executing the DPWs. 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 1 as a single component, the Workflow Execution Environment. 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 that is presented in section 5.2.3, may also be seamlessly integrated in the UPCAST architecture. For all pilots, workflow modelling is done with the use of ICTabovo goodFlows modeler while data exchange is performed with the Dawex marketplace platform.

5.2.1.1 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 computational intensive tasks, Nextflow can be easily redirected to utilize the local infrastructure within NHRF premises or academic cloud infrastructure.

5.2.1.2 Public Administration

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.

5.2.1.3 Health and Fitness

The deployment and execution of the Nissatech pilot is based on the infrastructure used in the commercial system Zona Zdravlja[9]. The data is collected from wearable devices used by trainees and stored in a MongoDB[10] 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.

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

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

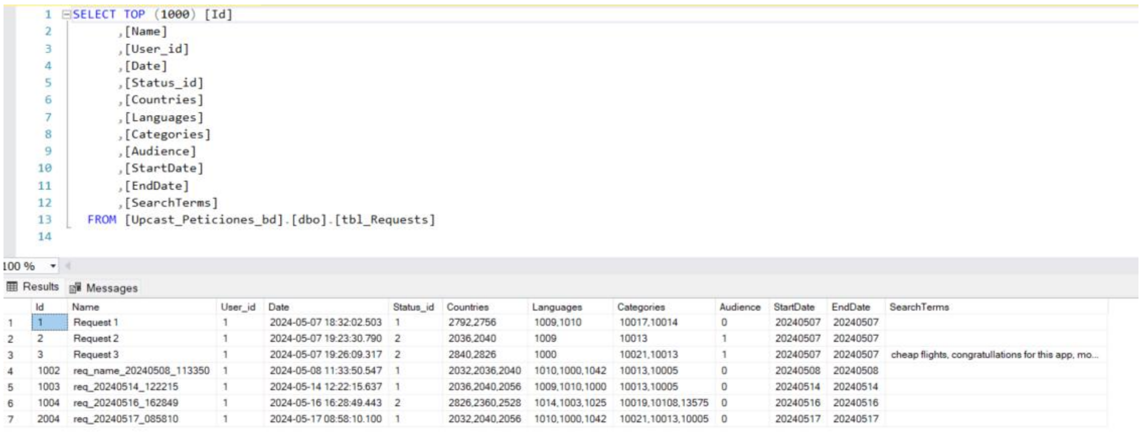
Quantity (how much can be published) depends on the characteristics of the Data Marketplace.

5.2.1.4 Digital Marketing 1

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 [2], 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.

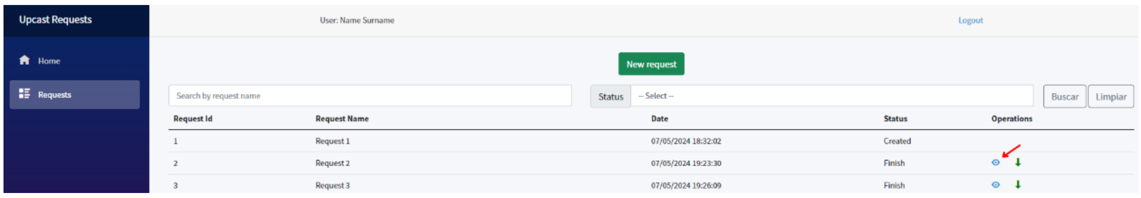


```sql
SELECT TOP (1000) [Id]
      ,[Name]
      ,[User_id]
      ,[Date]
      ,[Status_id]
      ,[Countries]
      ,[Languages]
      ,[Categories]
      ,[Audience]
      ,[StartDate]
      ,[EndDate]
      ,[SearchTerms]
  FROM [Upcast_Peticiones_bd].[dbo].[tbl_Requests]
```

	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 21 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 22 – red arrow) and related reporting services are executed.



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 22 User interface showing the status of the user request.

5.2.1.5 Digital Marketing 2

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[11], 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.

5.2.2 Nextflow

This section gives an overview of Nextflow[12], which is a workflow management system designed for the development and execution of computational pipelines. Nextflow facilitates the integration and orchestration of various tools and scripts, enabling reproducible and scalable analysis. Workflows are composed of processes, each encapsulating a computational task with defined inputs, outputs, and scripts. Channels connect processes, enabling data flow between them. They can be used to define complex data dependencies and parallelism.

Nextflow supports Docker[13], Singularity[14], and other container technologies, ensuring that pipelines can run in isolated environments with all dependencies. It also supports cloud integration and can seamlessly run on various cloud platforms (AWS, Google Cloud, Azure), with leveraging of their auto-scaling capabilities. Besides cloud, it can run on high-performance computing (HPC) clusters using traditional schedulers like SLURM[15].

Nextflow uses a domain-specific language (DSL) based on Groovy [16] to define computational workflows. Groovy is a powerful programming language for the Java virtual machine. The Nextflow syntax has been specialized to ease the writing of computational pipelines in a declarative manner. The recent DSL2 introduces modular pipeline development, enabling reusability and better organization. Processes are the fundamental units in a Nextflow pipeline, defining tasks with inputs, outputs, and a script to execute:

```
process myProcess {
    input:
    path 'input.txt'

    output:
    path 'output.txt'

```

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

[12] https://nextflow.io/

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

[14] https://sylabs.io

[15] https://slurm.schedmd.com/documentation.html

[16] https://groovy-lang.org

```
    script:
    """
    myCommand input.txt > output.txt
    """
}
```

If a process is containerized, a “container” parameter must be added in the process:

```
process myProcess {
    container 'myDockerImage:latest'
    ...
}
```

Channels connect processes by transferring data between them, enabling parallel execution and complex data dependency management. They support various types of data, including files, values, and collections. Channels can be created and manipulated using built-in methods:

```
Channel.fromPath('data/*.txt').set { inputFiles }
```

5.2.2.1 Nextflow plugins

Nextflow supports plugins that extend its capabilities. The Kafka[17] plugin provides an extension to implement built-in support for Kafka systems and manipulation in Nextflow scripts. It provides the ability to create a Nextflow channel to listening from topics as send message.

A snippet must be added to a nextflow.config file:

```
plugins {
  id 'nf-kafa@0.0.1'
}
```

5.2.2.2 Nextflow Executors

In the Nextflow framework architecture, the executor is the component that determines the system where a pipeline process is run and supervises its execution. The executor provides an abstraction between the pipeline processes and the underlying execution system. This allows to write the pipeline functional logic independently from the actual processing platform. In other words, the pipeline script can be written once and have it running on a computer, a cluster resource manager, or the cloud, simply by changing the executor definition in the Nextflow configuration file. For instance, to execute the pipeline script with the AWS Batch service, the following snippets may be used:

```
# enables nf-amazon plugin
plugins {
    id 'nf-amazon'
}
```

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

```
# defines as executor an existing AWS Batch queue
process {
  executor = 'awsbatch'
  queue = 'aws - queue1'
}
```

5.2.3 SIMPIPE Execution Environment

This section gives an overview of SIMPIPE[18], which is a software for executing data pipelines in a secure sandbox environment. The environment is a Kubernetes[19] cluster in which SIMPIPE leverages Argo Workflows[20] to orchestrate pipeline workflows. The pipeline workflows are defined using Argo Workflows YAML[21] format, in which each pipeline step references a containerized application image. In UPCAST, SIMPIPE will be used to automate pipelines for data sharing and processing agreements.

[18] https://www.sintef.no/en/software/sim-pipe

[19] https://kubernetes.io

[20] https://argoproj.github.io/workflows

[21] https://yaml.org

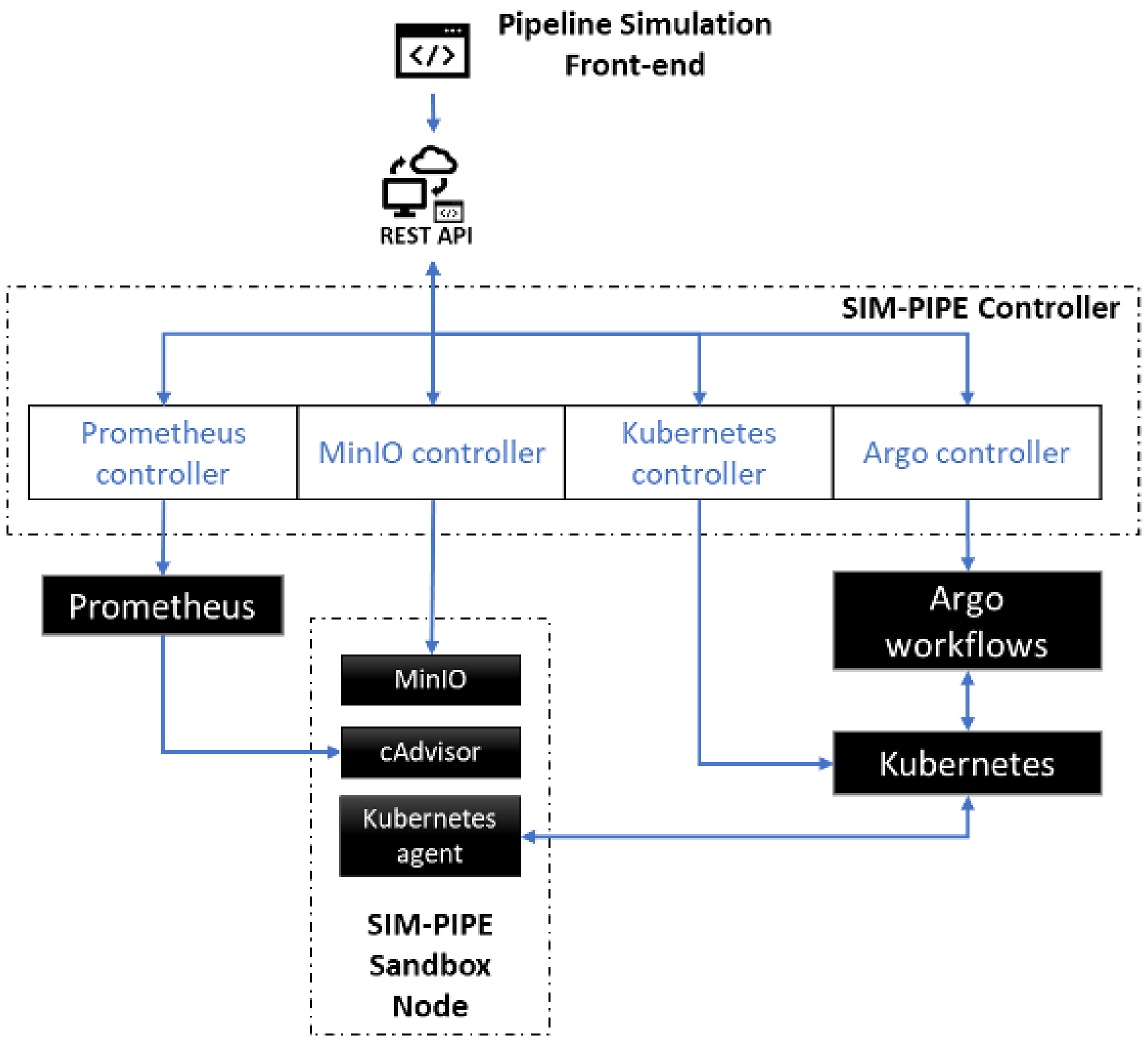


Figure 23: SIMPIPE Architecture.

Figure 23[22] shows the architecture of SIMPIPE, where the controller backend is a stateless NodeJS environment which provides a GraphQL API leveraged by the frontend graphical user interface. Furthermore, the API can be used to integrate SIMPIPE into other software. The SIMPIPE controller consists of subcontrollers responsible for talking to Kubernetes, Argo Workflows, Prometheus and Minio. Argo Workflows is responsible for assigning pipelines (also known as workflows) to the Kubernetes cluster. The pipeline is defined using a declarative language written in YAML file format. Argo Workflows allow the user to specify the pipeline tasks and dependencies and supports complex job orchestration using Directed Acyclic Graphs (DAG). Tasks can be run in parallel, in sequence, or based on conditions. The Kubernetes controller is used to manage projects and assign workflows. Prometheus is used for storing and querying metrics such as resource consumption from the execution of workflows, whereas Minio

[22] https://datacloudproject.eu/, Deliverable D3.4, Figure 9.

is used as an object storage. The Minio controller manages workflow artifacts and can be used to upload files that in turn can be referenced and used as input files for data pipelines.

When a data pipeline is defined as an Argo Workflow, with each of the steps in the workflow having their own containerized image, the pipeline can be uploaded as a project to SIMPIPE. Executions of the pipeline can be run by creating a dry run of the pipeline. Input parameters and input data can be customized when creating a new dry run. It is important to note that the containerized images must be made available for the pipeline to run successfully. Images from local or public registries can be referenced, or private image repositories can be used. If private image registries are used, a registry secret can be added to the workflow in order to successfully authenticate against the private image repository.

SIMPIPE requires a Linux-based operating system because it depends on Kubernetes which in turn relies on a Linux kernel to isolate resources and processes. SIMPIPE is easiest installable on macOS but can be installed on any Linux-based operating system including Windows Subsystem for Linux (WSL).

6 Monitoring

This chapter 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) plugin monitoring that allows providers to have an overview of the actions taken by the various plugins during dataset annotation, pricing, negotiation, etc.

Execution monitoring is supported by the monitoring plugin. The interface of the plugin and the means of streaming monitoring events are presented in section 6.1, whereas monitoring of UPCAST plugins is presented in section 6.2.

6.1 Execution Monitoring

This section presents the execution monitoring plugin of UPCAST, which is responsible for monitoring the execution of a dataset. Dataset execution takes place in a control manner and under the terms of a contract that has been agreed and signed between the provider and the consumer. Execution monitoring is used for two reasons: (a) presentation of execution related data to the dataset provider along with relevant statistics and analytics through the Dashboard, and (b) verifying the compliance of the dataset execution with the terms of the contract that has been agreed between the dataset provider and the consumer by the compliance plugin. The following sections give details for the monitoring process and the metrics that are used by it.

6.1.1 Monitoring Process

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 by the use of standard streaming platforms. In UPCAST Apache Kafka[23] 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 streaming messages between publishers of messages (dataset consumers in the case of UPCAST) and subscribers to messages (dataset providers 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. Producers of messages (UPCAST dataset consumers) send monitoring events to a topic, whereas consumer of messages (UPCAST dataset providers) read those events from a topic. A given topic may have several producers and several consumers. Several publishers can publish

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

messages concurrently and several consumers may consume messages concurrently. The consumption of a message by concurrent consumers depends on if these consumers belong to the same consumer group. Consumer groups are collections of consumers. All consumers in a consumer group share the messages of a topic, which means that each message will be read by exactly one consumer. Consumers in different groups consume the same topic data, which means that a message will be consumer multiple times by consumers that belong to different groups. 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.

Kafka topics will be created by UPCAST consumers by using code like the following.

```
from kafka.admin import KafkaAdminClient, NewTopic


topic_name = ...
topic_list = [NewTopic(name=topic_name, num_partitions=1,
replication_factor=1))]
admin_client.create_topics(new_topics=topic_list, validate_only=False)
```

Once a topic has been created, it can be used for sending monitoring data between UPCAST consumers and providers. Sending a message to a topic can be done with code like the following

```
from kafka import KafkaProducer


client = KafkaProducer(bootstrap_servers=[IP_addr]:9092’])
topic_name = ...
event = ...
f = client.send(topic_name, event)
```

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]
  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 (section 6.1.2).
- 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.

6.1.2 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

Below are some examples of monitoring events in JSON format.

Example 1: Start of an FFT on an image by the image_analysis component.

```
{
  source: "image_analysis",
  timestamp: "2024-05-31T05:21:36Z",
  metric: "action-start",
  value: "do_FFT",
  result: None,
  log: "FFT on the image"
}
```

Example 2: Start of checking validity of raw data that are read by the read_raw_data component.

```
{
  source: "read_raw_data",
  timestamp: "2024-06-02T21:15:45Z",
  metric: "action-start",
  value: "check_validity",
  result: None,
  log: "Check validity."
}
```

Example 3: End of validity check of the raw data by the read_raw_data component with a failed outcome, and an explanation of the reason.

```
{
  source: " read_raw_data",
  timestamp: "2024-06-02T21:16:27Z",
  metric: "action-end",
  value: "check_validity",
  result: -1,
  log: "Validity check failed due to incorrect formatting."
}
```

6.2 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 of 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.

7 Conclusions

Deliverable D3.1 presents some of the key technologies of UPCAST, their interfaces and early designs. In particular, the document presents the negotiation plugin that is used between providers and consumers to negotiation the terms of the processing of the dataset and agree on a contract, the execution modules that are used to execute an agreed workflow and the monitoring plugin that is used to collect data of the dataset execution for presenting them to the dataset provider and the feeding them to the compliance plugin. Moreover, the data exchange functions of UPCAST are presented that are used for exchanging the dataset between a provider and a consumer in a secure way. The document complements the early designs of UPCAST plugins as they are reported in Deliverables D2.1 [5], D2.2 [6], D3.3 [7]. The final version of the Negotiation and Execution modules of UPCAST will be reported in D3.2 [8].

8 References

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

[2] UPCAST, “D1.3: Updated project concept and architecture,” 2024.

[3] UPCAST, “D2.2: Privacy and Usage Control Modules,” 2024.

[4] S. Jablonski and C. Bussler, Workflow Management: Modeling, Concepts, Architecture and Implementation, London: International Thomson Computer Press, 1996.

[5] H. Meda, A. Sen and A. Bagchi, “On detecting data flow errors in workflows,” *Journal of Data and Information Quality,* vol. 2, no. 1, pp. 4:1-4:31, 2010.

[6] A. Kahn, “Topological sorting of large networks,” *Communications of the ACM,* vol. 5, no. 11, pp. 558-562, 1962.

[7] UPCAST, “D2.1: Pricing and Discovery Modules v1,” 2024.

[8] UPCAST, “D2.2: Privacy and Usage Control Modules v1,” 2024.

[9] UPCAST, “D3.3: Environmental Module v1,” 2024.

[10] UPCAST, “D3.2: Negotiation and Execution modules v2,” TBP 2025.

9 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