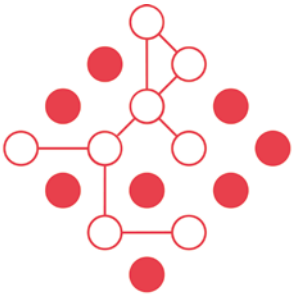

DATA MARKET

AUSTRIA

www.datamarket.at

Data Technology Specification and Development Roadmap

Deliverable number	*D5.1*
Dissemination level	*Public*
Delivery date	*19 June 2017*
Status	*Final (Version 1.2)*
Author(s)	*Bernd Ivanschitz, Ross King, Thomas Lampoltshammer, Sven Schlarb*

The Data Market Austria Project has received funding from the programme “ICT of the Future” of the Austrian Research Promotion Agency (FFG) and the Austrian Ministry for Transport, Innovation and Technology (Project 855404)

Executive Summary

This document presents the DMA Data Technology Specification and Development Roadmap regarding the ongoing work in WP5 of the DMA project. The document relates to deliverable D2.2 “D2.2 Community-driven Data-Services Ecosystem Requirements (1st version)” which formulates the requirements of the DMA community. It will be explained in what way the requirements formulated in D2.2 are going to be addressed. The document gives an overview of the components that are going to be developed in WP5 and how these components are planned to be employed to provide a dataset ingest solution as part of the DMA federated cloud environment. Furthermore, it is explained how the components are exposed as REST services and how these services are going to be combined to provide an integrated solution for dataset ingest and management, preservation, provenance capturing, quality enhancement, semantic enrichment.

Table of Contents

List of Abbreviations

AAL	Active Assisted Living (domain)
BC	Blockchain
DMA	Data Market Austria
DM	Data Management
DQ	Data Quality
ENGY	Energy (domain)
EOS	Earth Observation/Space (domain)
HDFS	Hadoop Distributed File System (Apache Hadoop)
PID	Persistent Identifier
I4.0	Industry 4.0 (domain)
MOB	Mobility (domain)
SC	Service Component

1 Introduction

An essential part of the Data Market Austria platform is the data provisioning, management, and storage environment. It consists of a set of data ingest services which allow publishers to incorporate a great variety of data types available in form of data files of varying size, data streams, or as data consumption endpoints. The ingest services must uniquely identify datasets and their parts, extracting existing and generating additional metadata, providing indicators for the quality of the datasets, storing and preserving the datasets for the long term, and finally providing a trustworthy provenance record of data access or manipulation operations.

This document relates to deliverable D2.2 “D2.2 Community-driven Data-Services Ecosystem Requirements (1st version),” which formulates the requirements of the DMA community. A relation to WP3 exists in relation to the data provenance tracking which is a basic requirement for the business model development. Furthermore, WP5 depends on WP4 which provides the basic infrastructure services. WP5 lays the foundation for WP6 by providing data access services. Cross-work package services for security (authentication/authorization), user management, monitoring, storage as well as central/federated storage services need to be coordinated with WP4. In what concerns the specification and implementation of services, WP5 relates to WP6 and WP7. For this reason, this deliverable will also address architectural aspects which go beyond the scope of WP5’s main objective to deliver dataset ingest services for the DMA platform.

The document is structured as follows: In the following section “Required Capabilities”, WP5 will express its view on several topics which have been covered across different domains in D2.2 “D2.2 Community-driven Data-Services Ecosystem Requirements (1st version)”. The purpose is to highlight the technical topics which have been identified as being relevant from WP5’s perspective and to explain to what extent the requirements are going to be addressed. Section two gives an overview about basic technical frameworks and technologies which will be used to address the requirements discussed before and explains how they the presented technologies are going to contribute to an integrated dataset ingest solution. Section three outlines the basic components which are going to be developed, integrated and deployed by WP5. Section four defines the various services by providing service interface specifications. The section starts with an overview about WP5 services in the context of the DMA federated cloud environment, followed by explaining the principle of how services are going to be combined into composite data processing workflows. The following service specifications will provide concrete insights about what can be expected from the various services developed or integrated by WP5. The last section presents the development, integration and deployment roadmap broken down into individual components and services. The document concludes with a summary and a discussion of open questions and their possible implications.

2 Required Capabilities

2.1 Storing and processing within the cloud

According to the requirements from the DMA user community, as documented in Deliverable D2.2, all domains stated interest in the capability to store and process data in the cloud.

However, the term “cloud” can denote a broad range of remote storage and computing services. In a general sense, a cloud computing environment allows users to make use of applications and storage services on demand in an external network. The cloud environment can be public or private and there is a great variety of business models how these services are being paid by users.

In the context of the DMA project, two aspects of cloud computing are most relevant:

- The ability to "run" a DMA node (cloud compliant software/application stack) in a remote environment of an infrastructure provider.
- The ability to perform massive computations with very large datasets using a computer cluster.

The DMA community (as defined in D2.2) representing the AAL domain stated interest in the ability to run data analytics directly in the cloud. Other domains did not explicitly mention it, but the high interest in the second aspect related to "big data" allows the assumption that it implies interest in the first aspect as well. The DMA community raised general concerns regarding security and privacy of data. This is of high relevance when the intention is to host data and services in a remote environment.

WP5 will address both aspects, i.e. it will make sure to use cloud compliant technologies (software modules deployed as containers) which allow use of the DMA software stack in environments of a variety of infrastructure providers to choose from. The second aspect is to consider the ability to carry out large scale data processing. The core technology to fulfil this requirement in the DMA environment is to foresee an additional hadoop cluster (MapReduce + HDFS) as part of the DMA architecture. Even though it is not part of the main architecture, the possible need to include a Hadoop Cluster with additional Big Data processing frameworks is taken into consideration.

2.2 Blockchain Technology

Blockchains are a specific type of Distributed Ledgers, which are systems by which a network of actors that do not necessarily trust each other can nevertheless share an agreed-upon history of events or transactions. Blockchains make use of hash functions and asymmetric encryption to build an immutable shared transaction history. In the purest form, a full copy of the blockchain and hence the transaction history is shared among all participants in a decentral peer-to-peer network.

The requirements on Blockchains coming from the DMA user community, as documented in Deliverable D2.2, are present but vague. We learn that Blockchains are discussed intensively in the ENGY Domain, and that there is a high degree of interest in Blockchain technologies coming from the I4.0 and EOS domains, in particular with regard to possibilities for smart billing and smart contracts. Other than concluding the interest (if not the necessity) in integrating Blockchain technologies within the Data Market technology foundation, and the requirement that smart contracts should be possible in this context, we cannot derive further specific requirements from the user community.

However, the project internal requirements, as documented by our project proposal, are considerably more specific. An analysis of the proposal yields the following concrete desired applications of Blockchain technologies expected by the project partners, the work packages, and our funding agency:

- Security
    - Authentication and authorization through blockchain actors
    - Ownership, privacy, and Data Protection
    - Encryption and verification of transactions
- Provenance capturing
    - Providing persistent identifiers and linking them to a blockchain entity
    - Providing a history of data ownership and data transformations through for example quality and enrichment services
    - Monitoring data and service use

- Smart billing and contracting possibilities
    - Controlling data and service access
    - Enforcement of Service Level Agreements
- Resilience through decentralized architecture
    - Applying in turn to provenance, preservation, and security

Some non-functional requirements can be derived indirectly. D2.2 calls for "a mechanism that ensures that only *serious vendors* trade on the DMA." Although the meaning of "serious vendor" can be debated, this in any case calls for some form of controlled or mediated access for new organizations wishing to join and use the DMA foundation. From this we infer that a private (also known as "permissioned") Blockchain will be required in order to enforce access restrictions (public Blockchains like Bitcoin are characterized by being open to anybody in the world who wishes to join the network).

We further infer that Blockchain implementations based on an open source license model will be preferable to closed, proprietary, and or commercial solutions, as the latter would have profound limiting effects on the business and deployment model for the platform.

2.3 Data set ingest

Regarding the data set ingest, deliverable D2.2 refers to the several functional aspects, such as the guided input process, data set profiling, metadata validation, metadata mapping, semi-automatic metadata extraction and generation (also called semantic enrichment), and validation of ownership. High importance was assigned by the MOB and ENGY domains to all of these aspects whereas I4.0, AAL and EOS have generally attributed medium importance.

Starting with the "guided input process" function, WP5 will provide alternative ways to start the ingest process. For file based datasets, there will be the following options to make datasets available in the DMA marketplace:

- Guided, form-based data upload.
- Machine readable instructions to initiate data crawling.
- Create data and metadata according to the data/metadata specification defined by DMA (which might require mapping external metadata schema to DMA metadata schema).

Regarding data consumption services provided in form of an API, there will be the option of a guided, form-based registration.

The "profiling" function will be mainly covered by WP7 (tasks 7.1 and 7.2). However, WP5 can support in integrating the information extraction services into the data set ingest pipeline in order to make the necessary metadata available to the matchmaking framework (WP7, task 7.1).

The "metadata validation" function will be provided as part of the data set ingest pipeline. The minimum functionality is a format completeness validation which can be extended to consider interoperability aspects, i.e. verifying the availability of information entities required by other DMA services. DMA will also support "metadata mapping" (WP6, T6.3) to map external metadata schemas to DMA metadata schema (before ingesting such metadata together with the data set).

Similar to the "profiling" function, there will be "semi-automatic metadata extraction and generation", also called "semantic enrichment", function developed by WP6 (task 6.3), WP5 can support this function by integrating the service into the data ingest pipeline.

The "validation of ownership" function is planned to be provided by integrating the DMA identity services with blockchain services.

2.4 Standards for long-term-preservation

In a general sense, long-term preservation capability was agreed as an important aspect by the DMA user community, as documented in Deliverable D2.2. This allows inferring the general requirement that there must be means to ensure long-term preservation of data assets by making sure that datasets and related metadata which are being stored in the DMA federated platform must be accessible and interpretable by the designated community on the long-term.

One of the core aspects that will be addressed by WP5 related to the area of long-term preservation is that a basic structure and storage format for a DMA data asset will be designed. Particular attention will be paid to making sure that the format is adequate for storing it in the DMA cloud environment and for transferring between DMA nodes. This allows replicating data in the DMA federated cloud environment, efficiently providing it to a consumer service, or supporting backup procedures.

Furthermore, the fact that long-term preservation is considered as being of general importance permits deriving the need for additional services to take adequate digital preservation measures. This means that it must be possible for collection curators and administrators to take appropriate actions and to address specific risks which are being identified.

Specifically this means that data characterization services must be employed as part of the data ingest pipeline. It must be possible to extract technical metadata (e.g. file formats and additional technical properties) which allow analyzing to what extent certain risks are applicable and if measures need to be taken. Identification of file formats is the most fundamental requirement to enable informed decisions about adequate preservation actions and it is the basis for being able to decide if data files which are part of a data set submission are accepted “as is” or if a transformation is required to make sure the data set can be used by the target community in a sensible way. A central requirement regarding this functionality is therefore to be able to define a machine-actionable policy for data migration. It closely relates to the quality enhancement services developed as part of WP5 because the purpose of these services is also to improve usability and interpretability of the datasets.

2.5 Security

In the broad sense, basic security functionality is a general requirement across all the different domains which have been asked to provide input for deliverable D.2.2. Medium to high relevance was attributed to the security aspect and there is general agreement, that there must be mechanisms to ensure secure data transfer and e.g. versioning, i.e. data provenance. Tracking the usage of data and services including basic security and access control can be seen as a specific function of the provenance capturing functionality and is rated as being of medium importance by the different domains.

Additionally, specific security aspects were highlighted by some domains, such as the following:

- In the AAL domain there is a focus on concerns related to privacy loss and data leaks. The reason for this is that there is personal information involved which are subject to national and international data privacy policies and data protection directives. As a consequence, the implementation of a high level of confidentiality and privacy measures is demanded by this domain.
- In the EOS domain it is stated that there is a need for security in trading and using the data, e.g. mechanisms that must to be put in place to ensure that data is not exploited by non-authorised parties.

- Related to the MOB domain it is stated that provenance and “security of data” plays an important role.

As a cross-work package requirement, the security aspect is relevant for all technical work packages and central security and access control services are being developed by WP4 as part of the basic infrastructure services (task 4.1). However, a significant contribution to the security concept is also to be expected by WP5, especially regarding task 5.2: “Blockchains for Security and Provenance”. A close coordination between task 5.2 and 4.1 is therefore required to enable the integration of the basic services for user management and authentication, service use authorization, and provenance capturing with the blockchain technology. To give a concrete example, this means that the user management service provided in the context of WP4 must trigger an event for the blockchain service to register the user as an entity in the blockchain. Subsequent events, such as a data set transformation operations, can then reference this user entity.

A security concept underlying the DMA basic platform services is still being developed. For the time being it is assumed that the DMA services landscape is constituted by loosely coupled services using the REST architectural approach. In the prototype development phase[1], users will have to authenticate using a central user authentication service (for example, based on OAuth 2.0 with SSL encrypted connections). In the final release phase it is planned to extend this service. First, an identity service allows protecting against unauthorized use of data access or manipulation operations. The the blockchain service is used to verify if a user is correctly registered in the blockchain.

WP5 will put the focus on provenance capturing using blockchain technology. The data set ingest will be conceived as data set transformation and manipulation pipelines where each individual modification step can be recorded in a tamper-proof way.

3 Framework Technologies Overview

3.1 Persistence/Storage

As the focus of this document lies on data ingest services, it will not do a comparison or evaluation of persistence/storage technologies to be used in DMA as this falls primarily into the scope of WP4. The purpose of this section is to explain which storage backend technology WP5 is planning to use in for implementing data ingest services (even if the technological choices made so far are preliminary).

For the purpose of implementing data ingest services, the persistence/storage technology is of crucial importance because the essential outcome of the process is that datasets as well as existing or generated metadata are persistently stored in appropriate formats and that metadata can be made available to search, retrieval, data analysis, and other data consumption services.

Based on the expertise and evaluations carried out by partners of the DMA project (mainly Catalysts) – at the time of the creation of this document – the decision was to use the Red Hat Ceph Storage solution[2] as the main storage backend which is particularly suitable for use in cloud infrastructures.

[1] See section “Development and Deployment roadmap”.

[2] https://www.redhat.com/en/technologies/storage/ceph

As a secondary storage option, HDFS (the distributed file system of Apache Hadoop) is taken into consideration in order to support applying scalable Map/Reduce processing of very large datasets. However, the primary outcome of the data ingest process is to store the outcome of data set submission processes in the Ceph Storage backend. Complete data asset bundles or parts of them would be copied into HDFS if needed.

The storage backend will be used to execute the ingest of a data set submission and to store data objects and metadata files as a data asset bundle.

3.2 Cloud Technology

At its heart, cloud computing involves using the power of the internet to outsource tasks which would traditionally be executed on a personal computer – anything from handling simple storage to complex development and processing – to a vast and powerful remote network of interconnected machines. Based on the expertise of the DMA partners Catalysts and TMA , the DMA is able to provide the services and technologies to create federation of clouds (community, private or public clouds) that operate according to the preferences, choices and constraints set by its owners. The necessary resources to operate a federated cloud system will be primarily defined by WP4. At the time of the creation of this document, the DMA consortium member Catalysts is preparing a cloud-based test environment for the deployment of basic services such as data ingestion and metadata enrichment.

The DMA Platform will provide a Cloud storage solution to stash data on hardware in a remote physical location, which can be accessed from any device via the internet. Clients must be able to send files to a data server maintained by a cloud provider instead of (or as well as) storing it on their own hard drives. Cloud storage systems generally encompass hundreds of data servers linked together by a master control server, but the simplest system might involve just one.

Furthermore, possibilities for processing must be provided for the customers. Cloud computing involves clients connecting to remote computing infrastructure via a network to use shared processing power, software and other resources. This frees the DMA users from having to constantly update and maintain their software and systems, while at the same time allowing them to harness the processing power of a vast network. In recent years, cloud computing is emerging as the latest distributed computing paradigm which provides redundant, inexpensive and scalable resources on demand to system requirements. Meanwhile, cloud computing for the DMA could adopt a pay-as-you-go model where users are charged according to the usage of cloud services such as computation, storage and network services in the same manner as for conventional utilities in everyday life (e.g., water, electricity, gas and telephone). Cloud computing systems offer a new way to deploy computation and data-intensive applications. As Infrastructure as a Service (IaaS) is a very popular way to deliver computing resources in the cloud, the heterogeneity of the computing systems of one service provider can be well shielded by virtualisation technology. For the DMA it is very important that users can deploy their applications in unified resources without any infrastructure investment in the cloud, where excessive processing power and storage can be obtained from cloud service providers.

Furthermore, cloud computing systems offer a new way in which data providers and customers from all over the world can collaborate. For the DMA, all the metadata are managed in the cloud, which makes it easy to provide all users with the necessary information everywhere. Furthermore, there is also a possibility to store the whole data in the cloud, to share and sell the data among consumers.

These requirements on the cloud infrastructure for the DMA can be described with the 5 V's:

Volume, Velocity, Variety, Veracity and Value.

- Volume
  - The DMA must be able to handle enormous volume of data if needed. The cloud resources must be scalable and offer some possibility to store or temporarily store big amounts of data.
- Variety
  - Many sources and types of data both structured and unstructured will be offered on the DMA Platform. We must ensure that the storing and processing modules are able to handle the varying data.
- Velocity
  - The flow of data is nowadays massive and continuous. This real-time data can help customers and businesses to make valuable decisions that provide strategic competitive advantages. The DMA cloud resources must be able to process streaming data.
- Veracity
  - The DMA must ensure a certain standard for the metadata and data uploaded
- Value
  - A key property of the DMA is the possibility to combine data from different data providers to enrich the value of the datasets. The modified data can then be offered in a further step via the DMA to other companies and users.

3.3 Cluster/Parallelization

The **Apache Hadoop** framework[1], as already mentioned in the previous section, is a technology which is used for processing very large datasets in an infrastructure which can scale-out horizontally, i.e. the available system resources (i.e. for storage and processing) can be increased by adding new computer nodes. Using Hadoop, the number of nodes in a cluster is virtually unlimited and clusters may range from single node installations to clusters comprising thousands of computers. The general approach to parallelization of Apache Hadoop is the Map/Reduce processing paradigm which is divided into two main functions: The Map function, which splits input data into parts which are going to be processed in parallel by the processing cores being assigned in a computer cluster (fine-grained parallelization), and the Reduce function, which aggregates the outcome of the parallel processing of data parts a set of outputs generated by (possibly) multiple reducers. If the processing of very large datasets is required, e.g. to support the pre-processing of raw data, or if filtering/aggregating large data volume datasets to create data set derivatives, it is planned to use the Apache Hadoop framework. However, it is required in this case that data is transferred from the main Ceph Storage backend into the Hadoop distributed file system (HDFS) first. The time and resources needed for transferring a large data set (possibly of Petabyte size or bigger) into HDFS must be weighed against the advantage of gaining data locality, i.e. the benefit of having computation near the data.

For grid-based parallel processing, **Celery**[2] is used to spread work across multiple cores of a computer cluster (worker nodes). Celery uses execution units, called tasks, which are executed concurrently on worker servers. This kind of parallelization directly operates on data available in the Ceph storage backend or uses a copy of the data which was made available on the worker machine.

[1] http://hadoop.apache.org
[2] http://www.celeryproject.org

3.4 Streaming Data

In this section we present shortly the most popular data streaming API's currently available. The speed at which data is generated, consumed, processed, and analyzed is increasing at an unbelievably rapid pace. The DMA demands data processing and analysis in near real-time. Traditional big data-styled frameworks are not well-suited for these use cases. Multiple projects have been started in the last few years to deal with the streaming data. All were designed to process a continuous sequence of records originating from several sources. Our analysis mostly focused on Apache based frameworks since these are the most popular solutions available. All frameworks presented in the next paragraph use the Hadoop Distributed File System (HDFS) as a standard. However, HDFS is just one of the file systems that supports the data process engines. The data streaming api's can also use the ceph storage backend on which the current development is focused on. Yet, the main benefit for running the Apache frameworks in distributed mode over a Hadoop cluster is that you can take advantage of specific features like the ability to run multiple types of workloads on the same data at the same time. As already mentioned earlier in the report, HDFS is in planning for a secondary storage option which could be used as the dedicated backend to process data streams.

Table 1 gives a partial overview about the most promising candidates.

Streaming	Open Source	Licence Type	Maturity	Community Support	Guarantees	Other comments
Apache Spark http://spark.apache.org/	yes	Apache V2	**very high**; public deployment since 2010	**very high**	**Exactly-once**	Optimized for processing batch data
Apache Flink https://flink.apache.org/	yes	Apache V2	**medium**; developed since 2015	**medium**	**Exactly-once**	Optimized for processing streaming data
Apache Storm http://storm.apache.org/	yes	Apache V2	**very high**; public deployment since 2011	**high**	**At-least-once**	Pioneer in large scale stream processing

Table 1 Overview of the data streaming API's.

To fulfill our requirements, to find a highly performant and reliable system with the ability to process events, batch data and live streams at a consistently high rate with relatively low latency we compare the different candidates using benchmark evaluations done by Yahoo and other Companies. The streaming systems are compared using the maximum throughput while maintaining the best possible fault tolerance. All experiments were run on the same cluster to ensure a valid evaluation. It has to be noted that various systems approach problems fairy differently and therefore it's very hard to design not biased tests.

To define the best framework depends on the general requirements of the application. There are different factors which have an influence in the decision, such as the performance, maturity and community support. **Spark** is the most popular and mature data streaming framework. The method has some micro-batching limitations and the latency is compared to the others rather poor. However, for batch processing it is still the best option available. Spark batch processing offers highest speed advantages, trading off against high memory usage. **Flink** is conceptually a

great streaming system which fits very most streaming use cases and provides progressive functionality, like advanced windowing or time handling. If this functionality is needed then Flink is a good choice and it would outperform Spark and Storm in processing machine learning, graph algorithms and relational queries. **Storm** is the only native streaming system. If the focus is mainly on latency and near real-time processing then storm is probably the best choice and the most mature option. It can guarantee message processing and can be used with a large number of commonly used programming languages.

The best fit for the DMA will depend heavily upon the state of the data to process, how time-bound the requirements are, and what kind of results the users are interested in. There are tradeoffs between implementing an all-in-one solution and working with focused projects, and there are similar considerations when evaluating their performances. For a detailed evaluation results of the individual benchmarks we refer to the sources[1].

3.5 Blockchain

We are presently experiencing "peak Blockchain" in the sense of the well-known Gartner hype-cycle analysis. Significant venture capital (over one billion dollars) has already been invested in Blockchain-related start-ups (primarily in, but not limited to, the United States). As a result, new Blockchain projects are being published at a rate that is higher than could be feasibly evaluated by the resources available in the DMA project. Our analysis has therefore been limited to a set of the most well-known and established codebases but cannot be considered comprehensive.

Blockchain	**Open Source**	**Licence Type**	**Maturity**	**Community Support**	**Smart Contract Support**	**Other comments**
Bitcoin https://bitcoin.org/en	yes	MIT	**very high**; public deployment since 2009	**very high**	**low**; Bitcoin scripting language is not Turing-complete	Optimized for public network and virtual currency application
Enigma http://www.enigma.co/	no	unknown SaaS?	**low**; beta launched in early 2016; no information available about deployment	**low**	**high**; includes the concept of private contracts (making use of blockchain states that are not fully public)	The only solution that also supports Secure Multiparty Computation (SMC)
Ethereum https://www.e	yes	GPL 3.0	**high**; public deployment	**high**	**high**; designed for Turing-	Designed specifically for trusted code

[1] https://data-artisans.com/extending-the-yahoo-streaming-benchmark/
https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at

thereum.org/			since August 2014		complete scripting language	
Hyperledger Fabric https://github.com/hyperledger/fabric	yes	Apache V2	**low**; left incubator phase on 2017-03-03	**medium**; industry support from IBM and DAH	**high**; "Chaincode" is Turing-complete and supports multiple languages	Chaincode is executed locally and results in key-value "write set" that is applied to the distributed ledger
Hyperledger Sawtooth Lake https://github.com/hyperledger/sawtooth-core	yes	Apache V2	**very low**; still in incubator phase	**low**; very small community contributing to codebase and issue tracker	**medium**; so-called "transaction families" can be custom implemented but this requires significant know-how and resources	Uses PoET consensus protocol (very efficient) but this requires special CPU hardware support
Ripple https://github.com/ripple/rippled	yes	Ripple (MIT-like)	**medium;** exists since 2013, deployed in several commercial contexts	**medium;** support comes more from a private company rather than a community	**low;** Codius smart contract project discontinued in 2015	Strictly speaking, Ripple makes use of a "trust graph" rather than a Blockchain

Regarding matching to requirements, we assume that all of the Blockchain or distributed ledger technologies evaluated can support the decentralized resilience requirements, as this is fundamental to their design and function. Almost all Blockchains offer an open source implementation, with the exception of Enigma. We have included **Enigma** on the list in order to highlight its unique support for Secure Multiparty Computation (SMC), which is a requirement implied in the original proposal. We suggest that SMC must either be considered out of scope, or must be implemented outside of the Blockchain context, as the proprietary nature of Enigma makes it impossible to evaluate let alone integrate in the DMA technology foundation.

Given the remaining candidates, our decision will be based on three primary factors: maturity of the solution, community support, and support for Smart Contracts. In terms of maturity, **Bitcoin** (and its various derivatives) have a clear advantage, but the worst properties in terms of Smart Contracts, because the Bitcoin scripting language is by design Turing-incomplete (it does not support loops). The second candidate in terms of maturity is **Ethereum**, which has been publicly deployed since August 2014. In addition to this advantage, Ethereum was developed specifically with Smart Contracts in mind, although it should be noted that the manner in which code is executed in an Ethereum network is both complex (involving the concepts of "gas" and "ether") and limited in computational power (often compared to a "Smartphone from 1999"). **Hyperledger Fabric** has a very interesting approach to Smart Contracts (so-called "Chaincode"), but is rather immature (having only been launched in March 2017) and lacks the wide community and tool

support that exists for Ethereum. **Hyperledger Sawtooth Lake** is even less mature than Fabric and would also require substantial investment in implementation and configuration to the extent that it is no longer considered a feasible candidate. **Ripple** is a relatively mature technology with commercial support, but has suspended development of its Smart Contract component.

4 Components

In this section the components which are developed in WP5 will be described in detail. The components will be used to build the DMA ingest solution and will be integrated as REST services. An exception is the "Semantic Enrichment component" which is developed in WP6. The integration of this component is planned, but it will only be referenced in this section. Details about this component have to be consulted in the corresponding deliverable D6.1.

The main service components (SC) which are going to be integrated into the dataset ingest environment are listed in Table 2.

Component	Service name	Description
Conduit Dataset Ingest & Preservation	Conduit	Data ingest pipeline framework which is used to perform a sequence of tasks needed for data ingest.
Persistent Identifier	PID-Service	This component offers a service which allows assigning a unique identifier to a data asset, data catalogue, data set, or any other data object contained.
Data Management GUI	DM-GUI	The Data Management GUI is a web user interface which permits uploading, modifying or removing submitted data assets. It provides also user interface for accessing administrative functions depending on the user role (DMA platform user, DMA organisation, DMA Administrator, etc.).
Blockchain Security & Provenance	BC-Service	The Blockchain Security & Provenance component. It allows registering required entities for data provenance capturing and smart contracts in the blockchain distributed database.
Data Quality	DQ-Service	This component allows improving the quality of dataset submission by identifying issues in CSV files like missing irregularity and encoding issues, and tools for automatic normalisation of data entities like mapping to common date or time representations and numeric formats.
Semantic Enrichment	SE-Service	Semantic enrichment of data is developed by WP6 (task 6.3). The service will be integrated

		into the Conduit data set ingest pipeline.

Table 2 WP5 Components

In the following, these components will be described in detail.

4.1 Conduit Dataset Ingest & Preservation

The different components developed in WP5 and partly in WP6 will be integrated into a data ingest pipeline processing framework named *Conduit*.[1] This framework is a backend for asynchronous task execution to perform data validation, transformation, quality enhancement and semantic enrichment operations. These operations are implemented as individual tasks which can be combined in data processing pipelines. It allows parallel processing of tasks on a cluster and is available as a multi-container application (Docker containers). Figure 1 gives an overview about the main components of Conduit. Starting with the box on the left side of the diagram, the user interface, represented by the top layer, is a Python/Django-based web application which allows managing the creation and transformation of data assets. The task execution backend, represented by the bottom layer, is based on Celery, an asynchronous task queue. Tasks can be assigned to Celery workers (green boxes with a "C") according to predefined queues which can access the Ceph storage backend in order to perform data manipulation operations in the working directory. The RabbitMQ[2] messaging system is used by Celery as a messaging agent to handle task execution requests, and the Redis[3] key-value store is used as a result backend for task execution metadata. The middle layer represents an optional layer based on Airflow, an Apache Incubator project for workflow orchestration which integrates seamlessly with Celery.[4]

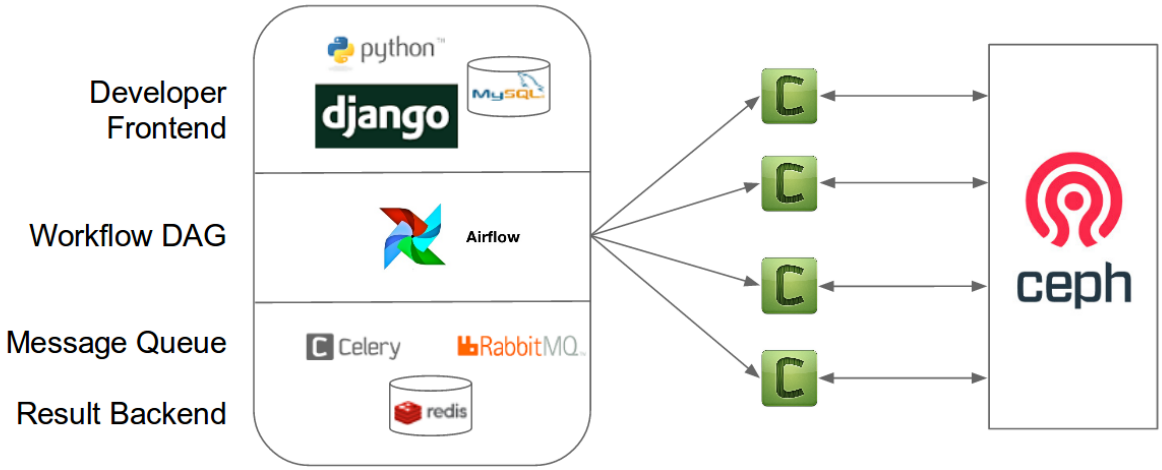

Figure 1 Overview about the main Conduit components used to create a data set ingest processing pipeline

Figure 2 shows a basic version of a data ingest pipeline which is deliberately kept simple for illustration purposes. It combines the tasks “Validation” to verify compliance regarding structural requirements and metadata of a DMA data set submission, “Registration” for assigning a unique identifier to a data set submission, “Data Quality improvement” for operations to to perform

[1] https://gitlab.com/datamarket/conduit
[2] http://www.rabbitmq.com
[3] http://redis.io
[4] https://airflow.incubator.apache.org

migrations in order to make sure the datasets are available in appropriate formats and possibly to improve the data itself, “Semantic enrichment” improve existing or create additional metadata, “Packaging & Storage” to create a transferable entity of the datasets included in a data asset and to transfer the data asset bundle to the storage backend.

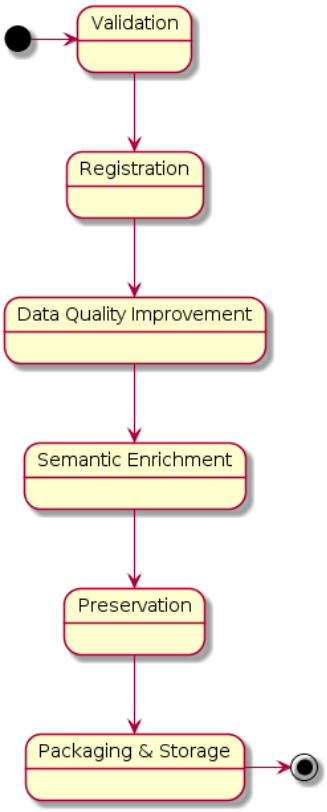

Figure 2 Example of a data set ingest processing pipeline.

As already pointed out, depending on the type of data and metadata available in a submission, this basic workflow is going to be extended by additional steps, such as “metadata extraction”, “data set encryption” and other data set type specific operations.

The backend can also be controlled via remote command execution without using the web frontend. Outcomes of operations which are performed by a task are persisted in the working directory of the dataset submission process.

The data set ingest is implemented as a set of backend Celery tasks. These tasks can be executed using the web application frontend, by invoking the tasks in headless mode. The tasks can also be combined as pre-defined workflows for batch processing which executes a chain of tasks for automatic data set ingest.

The data ingest processing pipeline is an extensible, data-oriented workflow which can be adapted to specific needs by inserting new tasks at any point of the workflow. According to the Unix maxim to “make each program do one thing well”[1], the ingest process is divided into a set of atomic tasks which perform a specific validation, transformation, or enrichment step. However, each processing step does not necessarily execute one single action, but can initiate a series of tasks or a complete workflow as well.

[1] *The Bell System Technical Journal*. Bell Laboratories. M. D. McIlroy, E. N. Pinson, and B. A. Tague. “Unix Time-Sharing System Forward”. 1978. 57 (6, part 2). p. 1902.

4.1.1 Web user interface

The user interface is a Python/Django-based web application. It consists of the Conduit admin user interface which can be used to test individual tasks of the dataset submission pipeline for the creation, validation, transformation, and enrichment of data assets. Part of this pipeline creation is also the integration of service invocations, such as the metadata enrichment workflow, for example (see deliverable D6.1, section 5.3, Figure 2).

Furthermore, the web application includes a wizard-assisted dataset upload user interface where at least the obligatory metadata must be provided. Apart from the dataset upload, there are other means to prepare a dataset submission for a DMA node. For example, data can be provided by harvesting from external sources or created by custom batch creation.

4.1.2 Validation

The DMA project develops an agreed-upon set of metadata which allows a collaboration with international data portals, for example the Industrial Data Space (IDS) of the European Commission. In cooperation with WP 6 and 7 a metadata core for the DMA is being defined which covers the minimum requirements needed to publish and process data using the DMA platform. It is inspired by the W3C Data Catalog Vocabulary (DCAT)[1] metadata standard for describing datasets in order to lay the ground for enabling interoperability. The DMA metadata core establishes classes and properties for describing datasets and services that are accessible on the DMA. However, it does not provide a formal, complete definition of all necessary dimensions for describing datasets. Instead, it provides the means by which dataset and service descriptions can be made discoverable on the DMA. That way, it is possible for users of DMA to find and retrieve datasets and services, as well as to judge their suitability for a particular purpose.

In alignment with the terminology used in this DCAT standard, WP5 uses the term "*catalogue*" to denote the encapsulating entity of a collection of datasets. A *data set submission* denotes the bundle consisting of the catalogue with associated metadata. The catalogue consists of a series of datasets and each data set has data items, documentation, and data set specific metadata.

As an example, let us assume a data set submission consists of a catalogue with two datasets. One data set with the data file `example1.csv` and the other one with the data file `example2.csv`. Additionally, each data set has documentation (`documentation1.odt` and `documentation2.odt` respectively) and data set specific metadata files (`specific1.xml` and `specific2.xml` respectively). Finally, the catalogue metadata is a file named `dcat.xml`. Figure 3 shows the structure of this dataset submission.

[1] https://www.w3.org/TR/vocab-dcat/

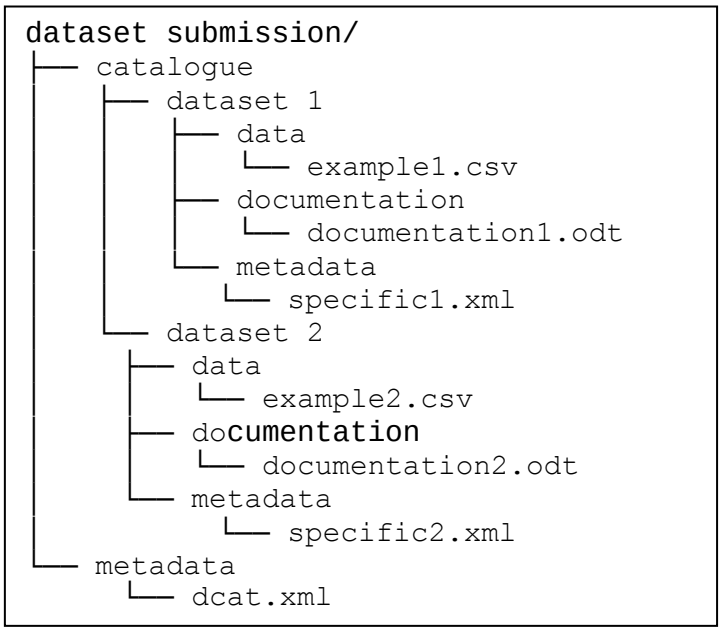

Figure 3 Dataset submission consisting of the Catalogue including the datasets and with associated metadata.

A *dataset ingest process* starts with a *dataset submission*. All operations of a dataset submission are executed in a working directory. It does not matter if the directory is populated using the data set upload forms of the user interface, or if they have been provided by transferring it as a packaged entity (created separately e.g. by batch creation). Data can also be provided by indicating external source locations from where the datasets can be *harvested*. The only requirement is that structural and metadata requirements must be fulfilled, i.e. the dataset submission must pass the validation procedure successfully. This means that any transformations which are required to create a valid dataset submission (e.g. metadata mappings) must be made in the "pre-ingest" phase, i.e. before the dataset ingest process starts.

The "catalogue" entity may contain any number of data set entities. For example, they could contain CSV files available in different languages or a set of geographic data files in different formats which are described by DCAT metadata and optionally other descriptive metadata available in the metadata directory.

The DCAT metadata is validated to verify formal validity and completeness. First of all, it will be checked if the obligatory metadata elements are provided. Furthermore, it will be checked for completeness, i.e. if each dataset which is part of the catalogue is referenced by the metadata and vice versa, if each file which is referenced is also available in the catalogue.

4.1.3 Semi-automatic metadata extraction and generation

See deliverable 6.1 task 6.3 "Semantic Enrichment and Linking of Data" for details about the functioning of the Semi-automatic metadata extraction and generation component.

4.1.4 Persistent storage

Each dataset ingest process has a *process identifier* (UUID) and a corresponding working directory which is used as long as the ingest process is not finalized.

Data validation, transformation, and manipulation operations are executed in this working directory which contains the catalogue including the datasets and the metadata directory. When the ingest process is finished, the catalogue and metadata will be aggregated as a *data asset* and it must have a *persistent identifier* (PID) which is different from the *process identifier* (UUID).

Once the dataset ingest process is finished, a PID for the digital asset is created according to the rules explained in section 4.2.2, for example:

```
ait:168bc315a2ee09042d83d7c5811b533620531f67
```

The PID is used to derive the folder name of the data asset which would then have the following name[1]:

```
ait=168bc315a2ee09042d83d7c5811b533620531f67
```

In a final step the datasets contained in the data asset are packaged (in tar or tar.gz format) and the data asset bundle is transferred to the storage area.

Figure 4 illustrates these concepts.

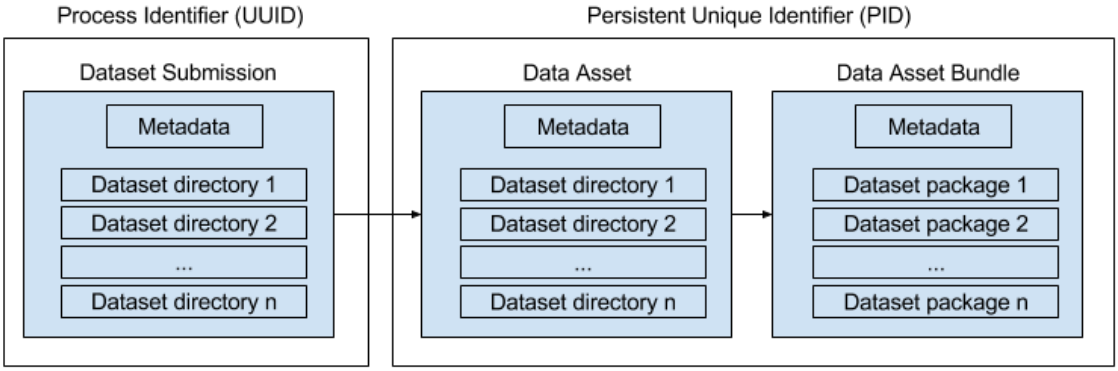

Figure 4 Dataset ingest process starting with a dataset submission to create a data asset and data asset bundle.

Data assets have a life-cycle and can be changed and there is a distinction between two cases:

As illustrated in Figure 5, an *edit* operation does not change the content in a way that it is necessary to assign a new persistent unique identifier. A *new version of the data asset* is created in this case. This can be a metadata correction or a format conversion which is supposed not to have any impact on the actual content. There will be a difference due to edit or conversion operations, however, the data asset can still be regarded as being “the same data asset” compared to the previous version.

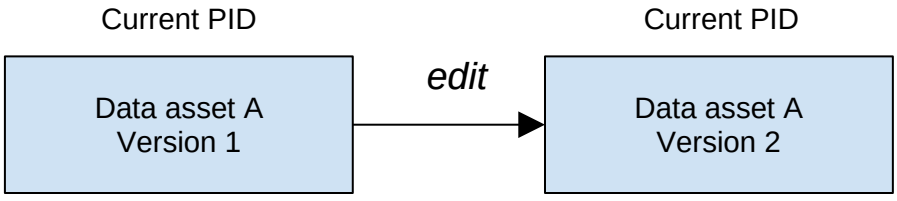

Figure 5 An edit operation creates a new version of a data asset. The persistent unique identifier (PID) remains the same.

[1] The identifier string is adapted according to According to the Pairtree specification, i.e. the slash ‘/‘ is replaced by the ‘=’ sign and ‘. This is explained in the following.

As illustrated in Figure 6, a fork operation changes the content of the data asset and thereby creates a *new data asset derivative*. The difference to the source data asset(s) is different to a significant extent so that the new data asset cannot be regarded as being "the same data asset" compared to the previous version.

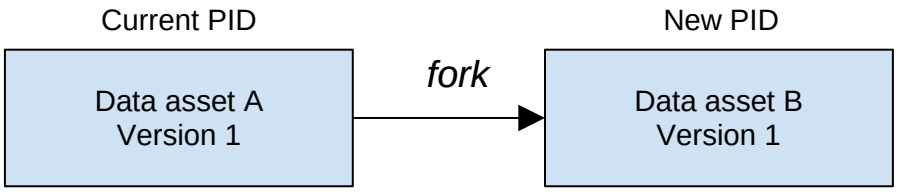

Figure 6 A fork operation creates a new data asset which gets a new persistent unique identifier (PID)

The data asset derivative keeps the provenance relation to the original datasets it was derived from. Fork operations can also reference several source datasets as shown in Figure 7.

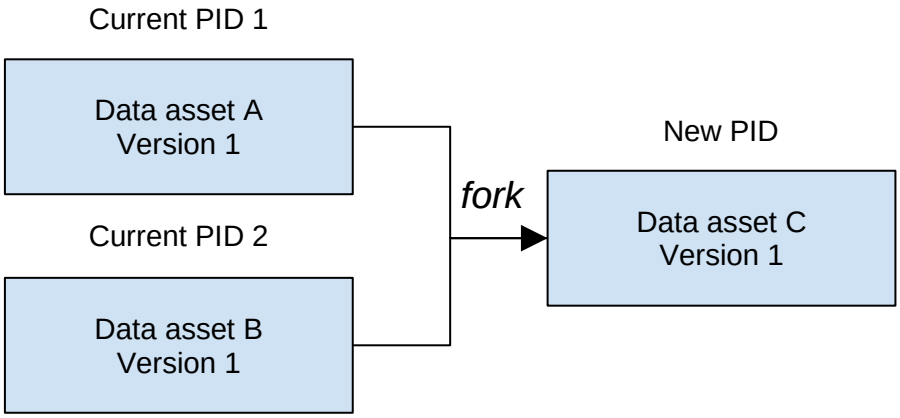

Figure 7 A fork operation can reference multiple source data assets.

Data asset bundles will be stored using the Python-based *Pairtree File System* implementation[1] of the *Pairtrees for Collection Storage* specification[2]. It is used to store the physical representation of a data asset bundle in a file system. The *Pairtree* is a filesystem hierarchy for storing dataasset bundles where the unique identifier string of the object is mapped to a unique directory path so that the file system location of the object can be derived from the identifier string. Basically, the identifier string is split each two characters at a time and the object folder has by definition more than two characters. Furthermore, the specification defines a mapping of special characters to a set of alternative characters in order to ensure file system level interoperability.

As an example, let us assume the following path to be the root folder of the *Pairtree* storage:

```
/var/data/dma/storage/pairtree_root
```

Furthermore, we assume that a data asset bundle is associated with the following identifier:

```
ait:168bc315a2ee09042d83d7c5811b533620531f67
```

Following the method defined by the *Pairtree* specification, this identifier is mapped to the following path:

```
ai/t=/16/8b/c3/15/a2/ee/09/04/2d/83/d7/c5/81/1b/53/36/20/53/1f/67
```

[1] https://pypi.python.org/pypi/Pairtree

[2] https://wiki.ucop.edu/display/Curation/PairTree?preview=/14254128/16973838/PairtreeSpec.pdf

and the actual data asset bundle is stored in a “data” folder which represents the leaf node of the *Pairtree* file system hierarchy.

The leaf node contains one or possibly more sub-directories (5-digits fixed length zero-filled number) for the versions of the data asset.

The full path to the data asset bundle is then as follows (to be read as a single line string):

```
/var/data/dma/storage/pairtree_root/ai/t=/16/8b/c3/15/a2/ee/09/04/2d/
83/d7/c5/81/1b/53/36/20/53/1f/67/data/00001/
```

Figure 8 shows the data asset bundle consisting of a set of packaged datasets and the catalogue metadata.

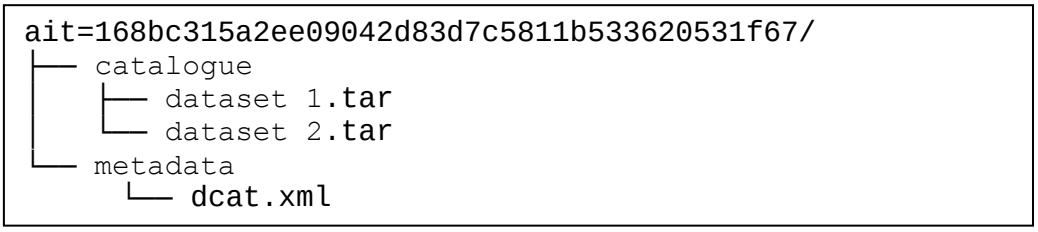

Figure 8 Data asset bundle consisting of two packaged datasets with metadata

For example, the full path to access the metadata file of this data asset bundle would be (to be read as a single line string):

```
/var/data/dma/storage/pairtree_root/ai/t=/16/8b/c3/15/a2/ee/09/04/2d/
83/d7/c5/81/1b/53/36/20/53/1f/67/data/00001/ait=168bc315a2ee09042d83d
7c5811b533620531f67/metadata/dcat.xml
```

The purpose of the *Pairtree* storage backend is to allow a large number of data asset bundles to be stored in a file system and permit fast access to individual datasets contained in a data asset bundle.

4.2 Long-term preservation and data citation components

4.2.1 Long-term preservation

The minimum requirement regarding long-term preservation is to ensure "bitstream preservation". Bitstream preservation denotes the ability to deal with data loss due to hardware or storage media failure and provides the means to restore the exact copy of a data file in case of data loss or file corruption. This requirement is addressed by using a storage backend which stores copies of data redundantly and allows detecting bit corruption by continuously comparing the signatures of replicated data files.

To ensure access and interpretability of stored datasets, it might be necessary to take advanced measures. The most prominent strategies to ensure render-ability of outdated file formats are "emulation" and "migration". The "emulation" approach essentially simulates an environment in which the file format was originally used. The “migration” approach creates an up-to-date derivative. In the DMA project, only the latter is supported.

In a broader sense, "migration" means to renew the technological environment to support the usage of data files in an up-to-date hard-/software environment. In a narrower sense and as it will be used in the context of DMA, "migration" means that data is converted from one data format to another in case the current data format is not considered adequate for future usage scenarios.

As a basis to support data format migrations, the Open Source software FIDO[1] is used to determine the Pronom Unique Identifier (PUID)[2], a file format identifier based on the PRONOM technical registry [3], for each individual data file which is part of a data set submission.

Conduit includes tasks for performing policy-based data format migrations based on PUIDs identified in a dataset submission. A set of tasks allows executing migrations according to predefined migration rules. A specific task queue can be defined for file format migrations in order to avoid that the main ingest pipeline is blocked by large numbers of data file migrations which need to executed in the background. Progress of migrations can be checked by invoking a REST service.

4.2.2 Persistent unique identifier (PID)

The Persistent Unique Identification (PID) component allows requesting identifiers and assigning it to data assets, services, and other DMA entities. In WP5 the default case is that a PID resolves to the URL of a DCAT metadata file of a data asset.

Originally, WP5 considered using the Invenio-PIDStore[4] as a core component for persistent identification. However, the fact that blockchain technology is used to capture the provenance, the alternative to develop a service which integrates with the blockchain layer will be taken into consideration. This would mean that data assets and services receive a blockchain address which is the basis to create the persistent unique identifier. By this way it is ensured that a tamper-proof provenance track of operations related to registered objects can be established.

Specific operations related to a data asset will be relevant regarding the persistent identification of the data asset.

The data asset which is identified by a persistent unique identifier (PID) may be subject to changes. However, not every change leads to the assignment of a PID as this would conflict with the requirement of using a stable PID URI for citation. A spell error correction in the metadata, for example, will not lead to a PID change because the underlying data asset is supposed to be “the same” as before.

However, if metadata of the data asset is changed which are of significance regarding the data asset provenance (e.g. data asset creator, publisher, etc.), then this change will be recorded and a new version of the data asset will be created. Even though this operation leads to the assignment of a new blockchain address in the provenance track, the PID always relates to the first version of the data asset.

If the content (i.e. one of the datasets included in an existing data asset) is changed, two types of operations are distinguished. An “edit” operation does not represent a significant change regarding the content of the data asset. It will lead to a new version and the PID will remain the same. If the content is going to be changed significantly, then a “fork” operation is required which leads to the assignment of a new PID while the relationship to the original data asset is recorded. Usually, changing the content of a data asset leads to a new PID assignment. However, maintenance activities or preservation measures may be necessary which are “content edit” operations that do not lead to the assignment of a new PID.

[1] https://github.com/openpreserve/fido

[2] http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm

[3] http://www.nationalarchives.gov.uk/PRONOM

[4] https://invenio-pidstore.readthedocs.io/en/latest/

The persistent identifier (PID) is composed of a namespace string corresponding to the member organization and the digital object identifier hash of the following form

```
{member-organization-namespace-string}:{data-asset-id}
```

For each PID there is a corresponding PID URI which is assigned to the metadata record of the data asset. This PID URI is used for dissemination purposes and in the context of semantic web and has the following form:

```
info:dma/{member-organization-namespace-string}:{data-asset-id}
```

Let us assume the organization AIT Austrian Institute of Technology GmbH got assigned the namespace string "`ait`" and it has generated the identifier hash "`168bc315a2ee09042d83d7c5811b533620531f67`" for a data asset, then this would entail the following URI:

```
info:dma/ait:168bc315a2ee09042d83d7c5811b533620531f67
```

4.3 Data Management GUI

The data management GUI will support the upload of datasets and the registration of data access endpoints including all CRUD operations (Create, Read, Update, Delete).

The interface will be build using Django as Python Web framework and AngularJS.

The following mockups give an insight into the data management GUI planning. A wizard-type data upload form guides users step-by-step through the upload process.

Figure 8 shows the first step of the dataset upload wizard-type data upload form. The purpose of this step is to collect basic catalogue metadata.

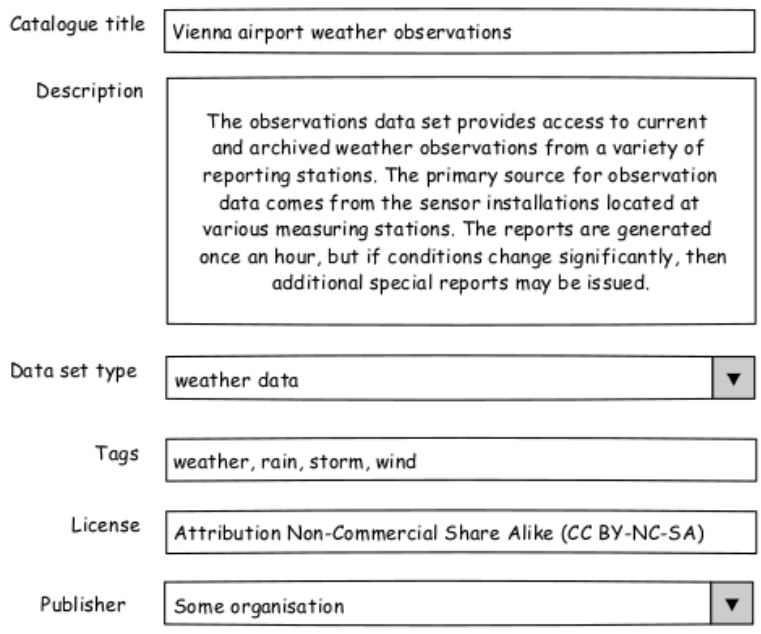

Figure 9 First step of the dataset upload wizard-type data upload form: Basic catalogue metadata

Figure 9 shows the second step where the data provisioning method is selected. Data items can be uploaded as individual data file items (“Upload resource”), as a resource bundle (“Upload resource bundle”), by indicating a harvesting URL (“Indicate URL where the resource bundle can be downloaded”), by referring to an externally hosted data set (“Add reference to external dataset”), or by adding streaming data endpoints (“Add reference to streaming data endpoint”).

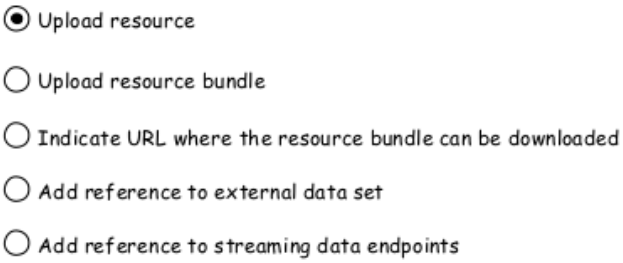

Figure 10 Second step of the dataset upload wizard-type data upload form: Select data provisioning method

Figure 10 shows the thirds step which corresponds to the first option shown in Figure 9 “Upload resource”. It allows adding a dataset resource with associated metadata. Note that regarding the first three options the result is that the actual data to start the dataset submission process is available. For the other two options there is no need for storing actual data. Only metadata about externally available datasets or as streaming data endpoints are provided in that case.

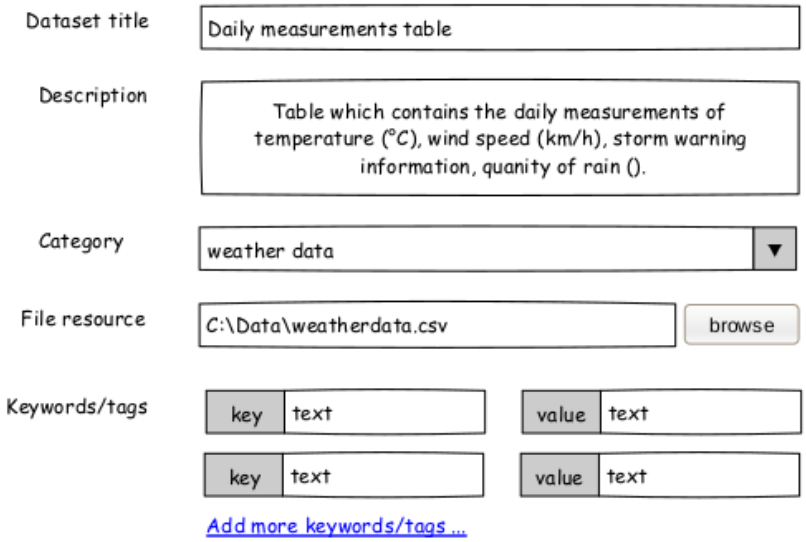

Figure 11 Third step of the dataset upload wizard-type data upload form: Add dataset item with metadata

4.4 Blockchain components for security and provenance

Membership

Based on our requirements, we intend to deploy a private (permissioned) Blockchain. The collection of organizations participating in this private Blockchain will be known as the Blockchain Network. We intend to use the Blockchain itself to regulate the participation in the Blockchain Network. This innovation counters a drawback of typical permissioned systems, in which the entity

that controls membership to the network enjoys a powerful and privileged central role, which in turn negates the presumed advantages of a decentralized system.
In order to implement a decentralized membership approval model, it will be necessary to introduce the "Organization" entity and this entity must have various types:

- **Candidate Organization**: a serious (trusted) organization that has been suggested for membership or has requested membership in the DMA network.
- **Member Organization**: An Organization that was either a Founding Organization or Candidate Organization that were voted in by existing Member Organizations.
- **Founding Organization**: the original members of the DMA project consortium. We must define this role in order to bootstrap the network membership and instantiation of the Blockchain nodes.

We will then introduce a voting protocol (in the form of a Smart Contract) that will admit (or deny) Candidate Organisations through transparent and irrevocable votes of the existing Members recorded in the Blockchain. A successful vote will result in a Candidate automatically becoming a Member.

Once an Organization has be established as a Member Organization, it can create Agents (individual user accounts) that can act on behalf of the Organization as for example Dataset curators or Service developers.

4.4.1 Deployment

The private DMA Blockchain is instantiated by a collection of DMS Nodes that all operate on the identical Genesis Block. DMS Nodes are Member Organizations, each of which must deploy an instance of the DMA Technology Foundation Stack (DTFS) under their control (as discussed in more detail in sections below). Every instance of the DTFS includes the DMA Blockchain component.

As subset of DMS Nodes will also be Mining Nodes. Mining Nodes are those nodes that also run Blockchain "miners", that is, those processes that validate transactions blocks in the system. As these systems require more computational power, we assume that the initial Mining Nodes will be instantiated at the three cloud installations foreseen by the project plan (the EODC, Catalysts, and T-Systems installations).

An additional special Blockchain node, the Bootstrap Node, must also be deployed (presumably at the central DMA instance that also hosts the portal application) in order to coordinate access to the peer-to-peer network.

4.4.2 Decentralized registry

The DMA Blockchain will be used as a decentralized registry for the primary entities in the DMA ecosystem. All entities will be assigned permanent identifiers when instantiated, and these identifiers will be stored in the DMA Blockchain; or, alternatively, Smart Contracts representing these entities will be stored in the Blockchain. Based on our requirements analysis, we conclude that the initial necessary fundamental entities are:

- **Organizations**: these are the legal entities that publish datasets and services and employ Agents.
- **Agents**: these are human beings that act on behalf of an organization. Agents for example curate datasets, develop services, or make use of datasets and services.
- **Datasets**: these are collections of digital information or objects that are published by Organizations and Curated by Agents.

- **Services**: these are software applications that are developed by Agents, published by Organizations, and operate on Datasets.

4.4.3 Provenance storage

Provenance documents the inputs, actors, systems, and processes that influence entities of interest, in effect providing a historical record of the entity and its origins. The permanent and irrevocable nature of Blockchain transactions makes them a good candidate for provenance tracking. However, this requirement implies that, in addition to storing entities themselves, the DMA Blockchain must also store transactions related to entities. Examples of such transactions would be:

- An Organization defines an Agent to be the curator of a Dataset
- An Agent forks an existing Dataset to create a new Dataset
- A Service accesses a Dataset
- A Service changes a Dataset (through semantic enrichment for example)

This in turn implies that these transactions must be modelled in the Blockchain, either through the core transaction model or through Smart Contracts. This aspect will be addressed in detail in DMA Deliverable D5.2.

4.4.4 Data set authenticity

Data authenticity means that a digital object is indeed what it claims to be or what it is claimed to be. Digital data can be assumed to be authentic if it is provable that it has not been corrupted after its creation. It is thereby clear that the provenance of DMA Datasets is a necessary but not sufficient condition for demonstrating authenticity. In order to complete the authenticity requirement, we need a mechanism for determining that a Dataset has not been altered, and linking this determination with the provenance mechanism.

One candidate technology for this would be hash encoding or hashing. Hashing is the transformation of a byte array into a shorter fixed-length value that represents the original digital object. A good hashing algorithm is (a) fast, (b) irreversible (it is not possible to deduce the original content from the hash value), and (c) unique (no two digital objects will produce the same hash value, or at least it is very unlikely to happen). Hashing is a key technology in Blockchains already and could also be applied to some Datasets.

We suggest the following approach: when a Dataset is created (that is, registered and potentially uploaded to the DMA network), a unique identifier is minted (as discussed in the Persistent Storage section above). At the same time, a hash value would be computed for the Dataset. This hash value would then be stored in the Blockchain as an attribute associated with the identifier. If a Dataset is altered, a new hash value is calculated, and this value is registered in the blockchain as a new version of the dataset (still associated with the original identifier). If a Dataset is forked, a new identifier is minted, and the hash value is stored with the new identifier, along with a reference to the identifier of the parent Dataset.

The drawback of this approach is that it is not idea for very large datasets (as it would take too much computational power to calculate the hash value) and it is inapplicable to dynamic datasets (as these would yield a different hash value each time the dataset changes) or data streams (for the same reason as dynamic datasets, but in this case the dataset could change even during the hash computation, making the hash value meaningless).

4.5 Data quality enhancement

As data quality was identified as a key aspect regarding requirements in a community-driven data-services ecosystem (see WP2, D2.2.), the following section provides an initial set of dimensions and metrics to assess issues in the existing metadata, as well as suggestions regarding the automated correction of issues within datasets. The referred metadata fields are taken out of the initial core set of the DMA metadata specification. As a defined and finalized list of data to be treated is not yet available, the suggestion for automated correction of datasets is demonstrated on the example of comma-separated value (CSV) files. Furthermore, additional data quality improvement methods will be provided via semantic enrichment of the given datasets (WP6, T6.3).

4.5.1 Data quality assessment

The following section describes the initial list of data quality assessment metrics, based on available metadata fields out of the core metadata set of DMA. Each metric is described along its dimension, how to measure it, and what specific metadata fields are required for the assessment (see Table 3)

ID	Dimension Definition	Metric Definition	Involved Metadata field(s)	Type of metadata
#DQ1	Header Completeness	Level of completion of metadata fields	Dataset::all mandatory	Mandatory
#DQ2	Understandability	Check complexity of dataset description	Dataset::Description	Mandatory
#DQ3	Contactability	Provision of required contact details	Dataset::Contact Point	Recommended
#DQ4	Temporal cohesion	Check whether stated temporal coverage is in line with release date	Dataset::Release date Dataset::temporal coverage	Recommended
#DQ5	Openness	Level of *open definition* compliance	Dataset::License	Optional
#DQ6	Language match	Does the language of the dataset matches the language field of the metadata	Dataset::Language	Optional
#DQ7	Currentness	Time between	Dataset::Frequency	Optional

		planned update and current version	Dataset::Last update/mod date	
#DQ8	Format compliance	Check whether the dataset is in the stated format	Dataset::Format	Optional

Table 3 Data quality dimensions and metrics

#DQ1 – Header Completeness: This metric measures the completeness of all mandatory metadata fields of a given dataset. The completeness is stated in %, based on the total amount of the existing mandatory metadata fields:

$$\boldsymbol{Header\ Completeness} = \frac{\boldsymbol{\#\ of\ filled\ metadata\ fields}}{\boldsymbol{\#\ of\ total\ metadata\ fields}} * \mathbf{100}$$

#DQ2 - Understandability: This metric measures the complexity of a given dataset description in terms of readability and therefore its understandability. As actual measurement, the Laesbarhedsindex (LIX)[1] is suggest, as it is particularly suitable for Western European languages:

$$LIX = (words + sentences) + 100 * \frac{words > 6\ characters}{words}$$

The results can be ranked according to the schema shown in Table 4:

Score	Assessment
0-24	Very easy
25-34	Easy
35-44	Standard
45-54	Difficult
55+	Very difficult

Table 4 LIX complexity score ranges

[1] Reck, R. P., & Reck, R. A. (2007). Generating and rendering readability scores for Project Gutenberg texts. In Proceedings of the Corpus Linguistics Conference. pp. 1-18

#DQ3 – Contactability: This metric assesses the availability of contact information regarding the person responsible for the given dataset:

$$Contactability = \{0, no\ information\ available \mid 1, information\ available\}$$

#DQ4 – Temporal cohesion: This metric evaluates for the description of the temporal coverage of the dataset (i.e., which time frame the data within the dataset cover) and the actual release date of the dataset being logical consistent (i.e., the dataset cannot be released earlier than the latest datum within the dataset).

$$Temporal\ cohesion = \{0, logical\ consistency\ not\ given \mid 1, logical\ consistency\ given\}$$

#DQ5 - Openness: This metric assesses the openness of a given dataset according to the attached license. As a quantitative representation of openness is challenging, a star-based rating system is proposed, which is reduced according to the level of restriction applied to the use of the dataset by the license.

License	Star-based rating
CC-Zero	5 (star)
CC-BY	4 (star)
CC-BY-SA	3 (star)
CC-BY-NC	2 (star)
CC-BY-NC-ND	1 (star)

#DQ6 – Language match: This metric compares the language which is stated in the metadata of a given dataset with the actual language being present within the dataset.

$$Language\ match = \{0, language\ does\ not\ match\ metadata \mid 1, language\ matches\ metadata\}$$

#DQ7 - Currentness: This metric describes the currentness of a given dataset decreasing over time. Along the definition of Atz (2014), the currentness can be formulated for a given dataset as follows:

$$Currentness\ \tau = \frac{update\ frequency\ \lambda + \delta}{today - last\ update}$$

Where denotes a fixed period of time, common for all dataset, regarding the required processing to be made available, while denotes the update frequency (i.e., as provided in the Dublin Core description)

Update frequency	Days	λ
Daily	1	1
Weekly	7	1,1
Monthly	30	1,15

Table 5 Update frequencies according to Dublin Core

#DQ8 – Format compliance: This metric compares the actual format of a given dataset and compares it with the format stated in the respective metadata field of the dataset.

$$Format\ compliance\ = \{0, format\ does\ not\ match\ metadata | 1, format\ matches\ metadata\}$$

4.5.2 Data quality improvement

4.5.2.1 CSV validation & correction

One of the most common dataset formats on the web are comma-separated value (CSV) files. A suitable candidate for the validation of CSV files is, e.g., CSV Lint[1] by the Open Data Institute (ODI). It checks a given CSV file along the specifications stated in RFC 4180[2] . It provides feedback in three different dimensions (i.e., errors, warnings, and additional messages) (see Figure 11), with a listing of all issues along these dimensions, what the problem is, how to possibly solve it, and where it is located in the given CSV file (see Figure 12).

	0 Errors	2 Warnings	1 Messages
Structure	0	0	1
Schema	0	0	0
Context	0	1	0

Figure 12 Reporting dimensions of CSV Lint (example taken from CSV Lint website)

[1] https://csvlint.io

[2] https://tools.ietf.org/html/rfc4180

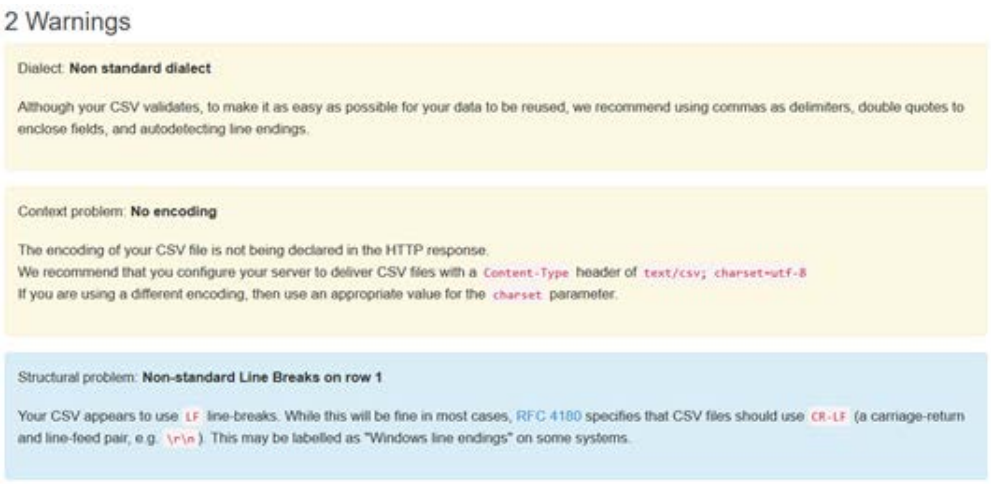

Figure 13 Detailed description and suggestions for resolving found identified issue

Regarding the automated correction of CSV files, there arise several challenges, which only some can be tackled without the danger to compromise the data within the file. These are mostly related towards fixing format-related issues, along with encoding issues present in the file. Such fixes can be, e.g., removal of optional quote characters, changing the field delimiter to a RFC-compliant format, fixing quote characters to standardized double quotes, as well as setting the encoding to the proper UTF-8 format. A candidate for performing these automated correction comes in form of, e.g., csvkit[1].

5 Services

In this section an overview about the ingest services provided on top of the ingest components is given. The first section explains the context of the ingest services in the DMA data market ecosystem. The second section explains the dataset ingest pipeline as the core technology developed by WP5. The third section is a first draft of the Service Interface Specifications (REST) provided by WP5.

5.1 Context

The deployment of services depends on the type of *DMA Node* which is a deployment instance participating in the DMA Austrian Data-Services Ecosystem, i.e. a stack of services deployed on IT infrastructure which is either available in house or being made available by a DMA compliant cloud infrastructure provider.

As illustrated in Figure 13, the stack of services deployed on a node depends on the type of node. On the right side of the diagram, there is a “DMA Data Provider node” (DP) which includes the obligatory object and metadata store and blockchain service layer as well as a set of additional services required to fulfil the requirements for data provisioning. And on the left side there is the “Central DMA Node” which, additionally to the DMA core services, provides central services for “Billing”, “Security & User Management”, and “Search & Recommendation”.

[1] https://csvkit.readthedocs.io/en/1.0.2/index.html

Figure 13 also shows that the data objects can be available in specific nodes only. In this example, the data asset bundle “A” is available in the object store of the central DMA node only and data asset bundles “B” and “C” are available on the “DMA data provider (DP) node”.

However, metadata about the availability of data packages available in the different nodes is stored and synchronized across all participating nodes of the federated DMA. This means that every node “knows” about data assets available in the DMA environment. The same applies for the distributed blockchain database which captures the provenance and logs exchange and usage of data assets.

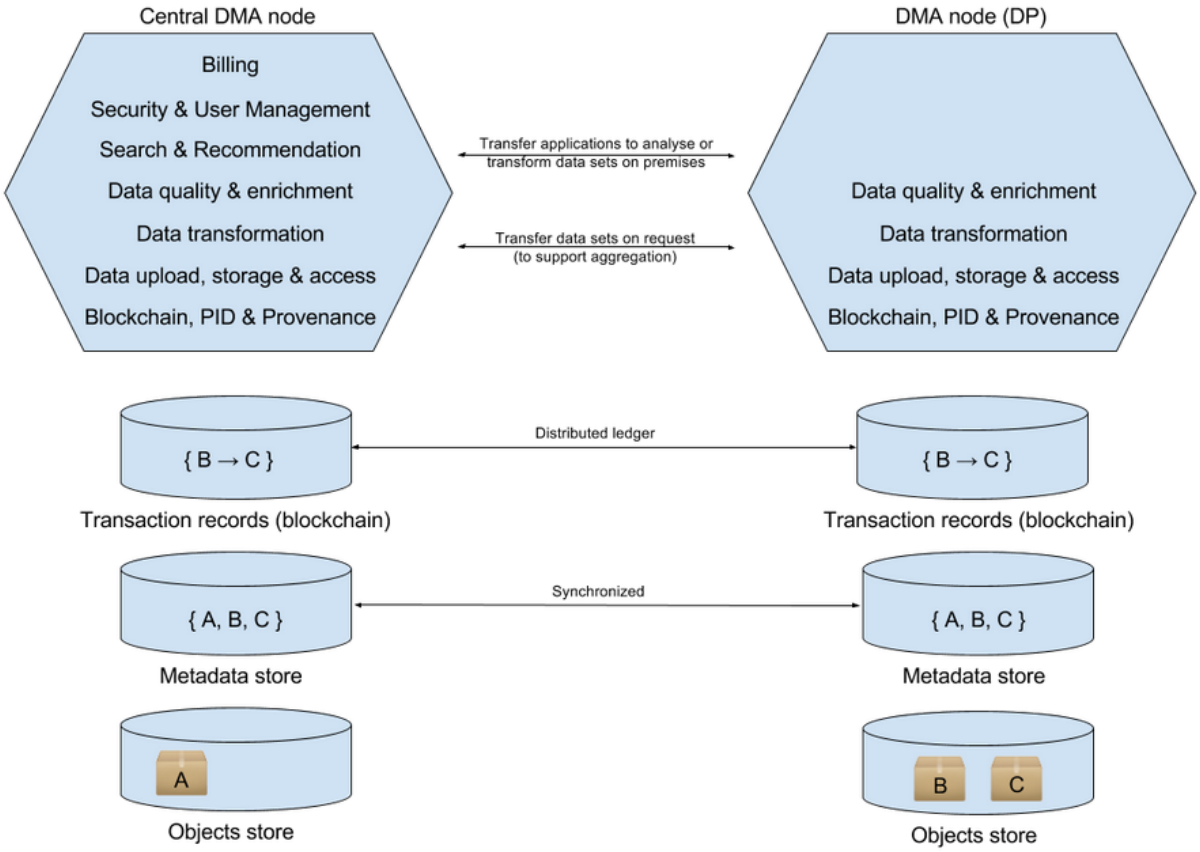

Figure 14 Central node and DMA node to illustrate the fact that data asset bundles can be available in specific object nodes only, whereas the metadata and transaction records metadata is synchronized between all nodes.

5.2 Data Set Ingest Pipeline

Services for data ingest can be connected into a dataset ingest pipeline using Conduit. Figure 14 shows an example of such a processing chain, including service invocations for “Validation”, “Registration”, “Semantic Enrichment”, “Preservation”, and “Packaging & Storage”. The chain of services is provided as a set of modular tasks which can be extended or organized in a different order. Service endpoints for one or different data ingest pipelines are available.

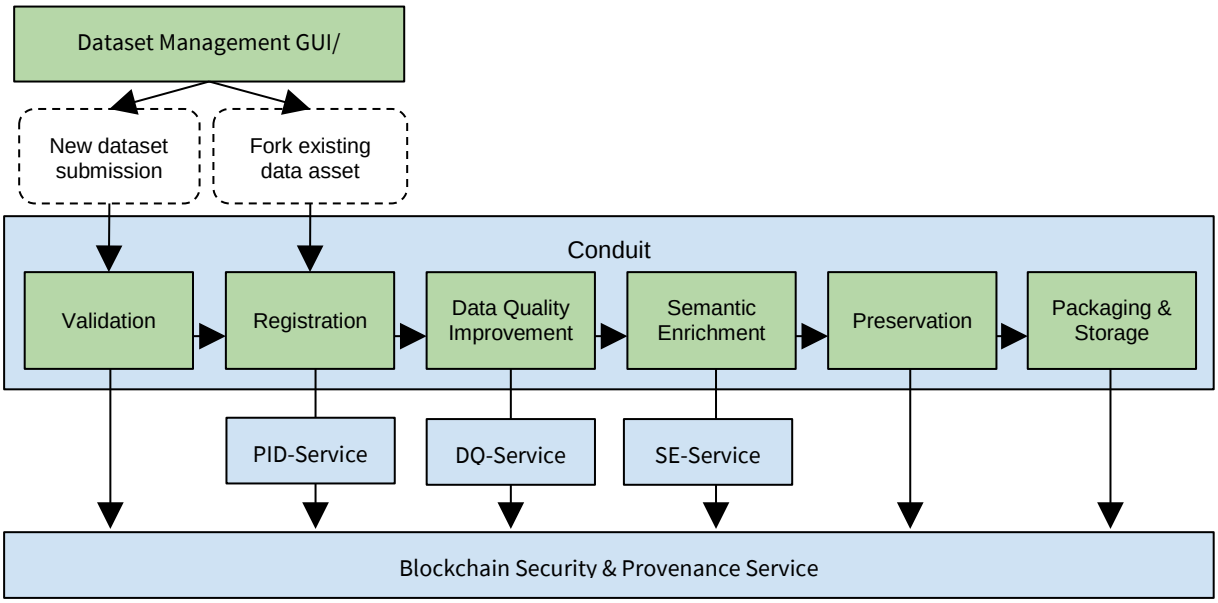

Figure 15 Example of a dataset ingest pipeline using Conduit

Figure 14 also illustrates that some of the tasks, such as the "Validation", "Preservation", and "Packaging & Store" tasks, are implemented directly in Conduit, while other tasks, such as "Registration", "Data Quality Improvement", and "Semantic Enrichment" tasks, invoke other services. Service invocations are always asynchronous. However, the pipeline processing only continues only if preconditions are fulfilled. For example, the execution of predecessor task must have been successful, otherwise the dataset ingest process is in a fault state. Service functions for polling the task processing status provide the required information to decide if the next step in the processing pipeline can be started.

The PID-Service and DQ-Service modules are developed in WP5, and the SE-Service is developed in WP6 (task 6.3). Conduit integrates with the Blockchain Security & Provenance to allow provenance capturing throughout all the processing steps being executed as part of the dataset ingest process.

The dataset ingest process ends if the data asset bundle is stored in the object store of the DMA node. The final status of the data asset is a catalogue metadata file together with a set of packaged datasets (in tar or tar.gz file format).

Creating the data asset bundle including metadata and packaged datasets is done because of the following reasons:

- It allows efficiently accessing and downloading selected datasets (because individual datasets contained in a data asset might be very large).
- It supports the integration with the blockchain layer because checksums for packaged datasets can be generated and stored in the blockchain.
- It makes metadata management more convenient because the metadata can be kept separate from the datasets (and metadata edits do not entail blockchain operations).
- The data asset bundle can be shared or synchronized more easily between DMA nodes because the transfer of a packaged dataset is more efficient compared to transferring the contained files individually.

5.3 Service Interface Specifications (REST)

The purpose of the following REST API service specification is to provide the basic definitions for an implementation using the OpenAPI Specification[1], formerly known as the Swagger API framework[2]. It provides a standard, language-agnostic interface to define REST APIs. Code can be generated for a variety of programming languages. The Django REST framework[3] will be used as the toolkit for building the Web API.

Note that most of the REST service operations specified in this section are supposed to be executed in an asynchronous manner as illustrated in Figure 15.

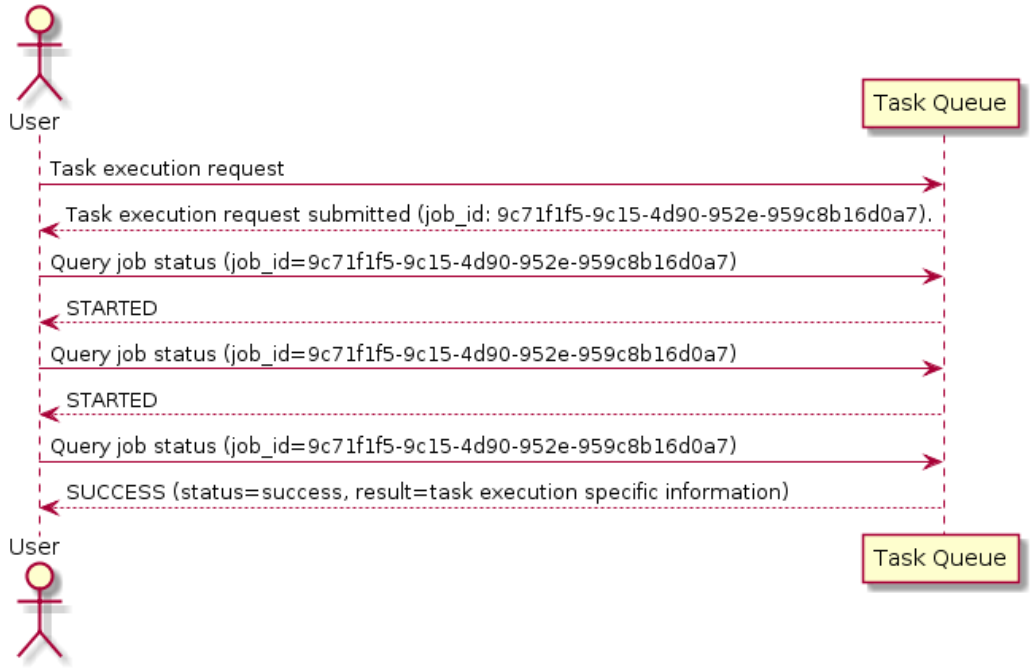

Figure 16 Asynchronous task execution

The request to execute a task is submitted and in case of success the response returns a job ID (job_id) which allows querying the job status. Querying the job status can be repeated until the job status is “SUCCESS”, in this case the response can return additional task specific results. This example represents the successful job execution. Failures can occur when submitting the job request or when polling the job status.

See Appendix A “HTTP Status Codes” to see the full list of HTTP Status Codes used in this specification.

5.3.1 Data set ingest services

5.3.1.1 Default ingest pipeline

The default ingest pipeline allows executing a predefined ingest workflow that operates on a working directory where datasets including the required metadata is available. The workflow ends with the data asset available in the local data store with a PID assigned.

[1] https://github.com/OAI/OpenAPI-Specification
[2] http://swagger.io
[3] http://www.django-rest-framework.org

5.3.1.1.1 Initialize new dataset submission process

Starting a new dataset submission process means assigning a process ID (process_id parameter which is a UUID). The process ID corresponds to a working directory on the DMA node which might get initialized by a scaffolding (folder structure) for data and metadata and initial metadata files. The process ID is only used to identify a dataset submission process in the context of a DMA node. Once the data asset is created and identified by a persistent unique identifier, the process ID is no longer of importance.

<table>
<tr><th colspan="2">Request</th></tr>
<tr><th>Method</th><th>URL</th></tr>
<tr><td>POST</td><td>api/v1.0/datasetsubmissions/</td></tr>
<tr><th colspan="2">Parameters/Request Body</th></tr>
<tr><th>Type</th><th>Params</th></tr>
<tr><td>HEAD</td><td>auth_key</td></tr>
<tr><td colspan="2">auth_key
The <code>auth_key</code> that was given in response to <code>/api/login</code></td></tr>
<tr><th colspan="2">Response</th></tr>
<tr><th>Status</th><th>Response</th></tr>
<tr><td>202</td><td>An example response is:
<pre>{
    "message": "Dataset submission process initialization
               request submitted successfully.",
    "job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7"
}</pre>Result can be obtained by job status request (using job_id):
<pre>{
    "success": true,
    "message": "Dataset submission process initialized
               successfully.",
    "process_id": "5f71d1d3-1a16-2d44-362e-959c8b16d0b8"
}</pre></td></tr>
<tr><td>400</td><td>{"error":"Process ID does not exist."}</td></tr>
<tr><td>401</td><td>{"error":"Unauthorized. Auth key is missing"}</td></tr>
<tr><td>500</td><td>{"error":"An error occurred."}</td></tr>
</table>

5.3.1.1.2 Perform dataset ingest

Request	
Method	**URL**
POST	api/v1.0/datasetsubmissions/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD	auth_key
GET	process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"message": "Ingest job start request submitted successfully.",
	"job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7"
	}
	Result can be obtained by job status request (using job_id):
	{
	"success": true,
	"message": "Ingest job finished successfully.",
	"pid": "info:ark/ait/168bc315a2ee09042d83d7c5811b533620531f67"
	}

400	{"error":"Process ID does not exist."}
401	{"error":"Unauthorized. Auth key is missing"}
500	{"error":"An error occurred."}

5.3.2 Persistent Identification and Long-term preservation services

The purpose of persistent identification services is to create digital assets by assigning a persistent unique identifier to the dataset submission. In order to change existing data assets, it is required to

either fork it, in case the intention is to create a new derivative (added value/content change), or to check it out if the data asset should be submitted as a new version under the same persistent identifier.

Change operation must therefore always operate on a dataset submission process (using the corresponding process_id).

5.3.2.1 Register dataset submission

A dataset submission is registered by invoking the PID-Service which assigns a persistent unique identifier (PID). After registration of the dataset submission, it is a data asset. The PID resolves to a URL where the metadata view of the data asset can be retrieved.

5.3.2.1.1 Registration request

Request	
Method	**URL**
POST	api/v1.0/dataassets/
Parameters/Request Body	
Type	**Params**
HEAD	auth_key
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	

Request body
{
"process_id": "4c65f1f5-9c10-4d90-952e-959c8b16d0a7"
}	

Response	
Status	**Response**

202	**An example response is:**
	{
	"message": "Registration request sent successfully.",
	"job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7"
	}
	Result can be obtained by job status request (using job_id):
	After checkout the new PID (pid) can be obtained.
	{
	"success": true,
	"message": "Data asset status assigned successfully.",

	"pid": "info:ark ait/168bc315a2ee09042d83d7c5811b533620531f67" }
400	{"error":"Process ID (process_id) is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
403	{"error":"Forbidden. The current user does not have access to perform the requested action"}
500	{"error":"An error occurred during registration."}

5.3.2.1.2 Fork a data asset

Forking a registered dataset means that a copy of the existing data asset will be created to start a new dataset submission process. The provenance relation to the original data asset will be recorded. A new identifier can be assigned to create a derivative data asset.

Request	
Method	**URL**
POST	api/v1.0/dataassets/<pid>/fork/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key pid
auth_key The `auth_key` that was given in response to `/api/login` **pid** The `pid` of the forked data asset.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"message": "Fork request sent successfully.",
	"job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7"
	}
	Result can be obtained by job status request (using job_id):
	After checkout the new process ID (process_id) can be obtained.
	{
	"success": true,
	"message": "Fork finished successfully.",

	<pre>"process_id": "8f71d1d5-1a16-1d00-352e-959c8b16d0a2"
	}</pre>

400	<pre>{"error":"Process ID (process_id) is missing."}</pre>
401	<pre>{"error":"Unauthorized. Auth key is missing"}</pre>

403	<pre>{"error":"Forbidden. The current user does not have access
	to perform the requested action"}</pre>

500	<pre>{"error":"An error occurred during registration."}</pre>

5.3.2.1.3 Edit a data asset

Editing a registered dataset means that a copy of the existing data asset will be created to start operating on a copy of the data asset. A new version of the data asset will be created. However, the difference compared to forking the dataset, no new identifier will be assigned. The modified dataset has the same external identifier which resolves to a new version.

Request	
Method	**URL**
POST	api/v1.0/dataassets/<pid>/edit/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key pid
auth_key The `auth_key` that was given in response to `/api/login` **pid** The `pid` of the data asset to be edited.	
Response	
Status	**Response**

202	**An example response is:**
	<pre>{
	"message": "Checkout request sent successfully.",
	"job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7"
	}</pre>**Result can be obtained by job status request (using job_id):**
	After checkout the new process ID (process_id) can be obtained.
	<pre>{</pre>

	<pre>{
	    "success": true,
	    "message": "Checkout finished successfully.",
	    "process_id": "8f71d1d5-1a16-1d00-352e-959c8b16d0a2"
	}</pre>

400	{"error":"Process ID (process_id) is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
403	{"error":"Forbidden. The current user does not have access to perform the requested action"}
500	{"error":"An error occurred during registration."}

5.3.2.1.4 Revoke data asset

If a data asset is obsolete and should be removed, this service allows revoking it from the DMA shared space. Note that this will not delete the data asset, but it will "mark it as deleted" in the federated blockchain database and authorization and access services will block use of the data asset.

Request	
Method	**URL**
DELETE	api/v1.0/dataassets/<pid>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key pid
auth_key The `auth_key` that was given in response to `/api/login` **pid** The `pid` of the data asset which is to be revoked.	
Response	
Status	**Response**

202	**An example response is:**
	<pre>{
	    "message": "Revocation request sent successfully.",
	    "job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7"
	}</pre>**Result can be obtained by job status request (using job_id):**

	{
	"success": true,
	"message": "Data asset revoked."
	}

400	{"error":"Process ID (process_id) is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
403	{"error":"Forbidden. The current user does not have access to perform the requested action"}
500	{"error":"An error occurred during registration."}

5.3.2.2 Preservation

The default ingest pipeline allows executing a predefined ingest workflow that operates on a working directory where datasets including the required metadata is available.

5.3.2.2.1 Apply migration policy

Application of a migration policy. A policy is of a set of policy definitions which consists of the format identifiers which can trigger a migration action. The action must be implemented and registered with the given name. Optionally, one or several conditions can be implemented as part of the migration action and referenced in the policy.

Request	
Method	**URL**
PATCH	api/v1.0/dataassets/<process_id>/migrate
Parameters/Request Body	
Type	**Params**
HEAD	auth_key
GET	process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
{ "policy": [{	

```
            "name": "pdfa",
            "formats": ['fmt/14', 'fmt/15', 'fmt/16'],
            "action": "ghostscriptpdfamigrate",
            "conditions": [
                "none"
            ]
        },
        {
            "name": "giftotiff",
            "formats": ['fmt/3', 'fmt/4'],
            "action": "imagemagickgifmigrate",
            "conditions": [
                "none"
            ]
        }
    ]
}
```

Response	
Status	**Response**

202	**An example response is:**
	`{ "message": "Migration policy application request submitted successfully.", "job_id": "9c71f1f5-9c15-4d90-952e-959c8b16d0a7" }`
	Result can be obtained by job status request (using job_id):
	`{ "success": true, "message": "Migration policy applied successfully." }`

400	`{"error":"Process ID does not exist."}`
401	`{"error":"Unauthorized. Auth key is missing"}`
404	`{"error":"No asset with the given PID (pid) found."}`
415	`{"error":"Unsupported Media Type (JSON request body`

	required)."}
500	{"error":"An error occurred."}

5.3.3 Blockchain services for security and provenance

5.3.3.1 Create an Entity

Create a Blockchain entity which can either by type "organization" or type "agent".

Request	
Method	**URL**
POST	api/v1.0/blockchain_entity/
Parameters/Request Body	
Type	**Params**
HEAD	auth_key

Request body
{
"entity_type": "agent"
}	

auth_key The auth_key that was given in response to /api/login **entity_type** The type of entity to create. Valid types are "organization" or "agent"	
Response	
Status	**Response**

202	**An example response is:**
	{
	"entity_id": "0x168bc315a2ee09042d83d7c5811b533620531f67"
	}

400	{"error":"Missing or invalid request parameters."}
415	{"error":"Unsupported Media Type (JSON request body required)."}
401	{"error":"Unauthorized. Auth key is missing"}
500	{"error":"An error occurred."}

5.3.3.2 Create a Contract

Create a Blockchain Smart Contract associated with either a "dataset", or "service".

Request	
Method	**URL**
POST	api/v1.0/blockchain_contract/
Parameters/Request Body	
Type	**Params**
HEAD	auth_key

Request body

```
{
  "entity_type": "dataset",
  "tou_contract_id": "0x225178b4829bbe7c9f8a6d2e3d9d87b66ed57d4f"
  "entity_hash":
   "0x60635f91977076abd6ed80e38908c962c028616b7d36ce4598e4f78a719af677"
}
```

auth_key
The `auth_key` that was given in response to `/api/login`
entity_type
The type of entity to create. Valid types are "dataset" or "service"
tou_contract_id
The Blockchain ID of the Smart Contract that defines the Terms of Use for the entity
entity_hash
{optional} This is a unique hash value associated with the entity that can be registered in the Blockchain (recommended: SHA256)

Response	
Status	**Response**
202	**An example response is:** `{ "contract_id": "0x168bc315a2ee09042d83d7c5811b533620531f67" }`
400	`{"error":"Missing or invalid request parameters."}`
415	`{"error":"Unsupported Media Type (JSON request body required)."}`
401	`{"error":"Unauthorized. Auth key is missing"}`
500	`{"error":"An error occurred."}`

5.3.3.3 Execute a Transaction

Execute a transaction by defining the transaction type, source and recipient of the transaction.

Request	
Method	**URL**
POST	api/v1.0/blockchain_transaction/
Parameters/Request Body	
Type	**Params**
HEAD	auth_key

Request body

```
{
  "contract_id": "0x168bc315a2ee09042d83d7c5811b533620531f67",
  "transaction_function": "transfer_ownership",
  "transaction_parameters": {
      “param_1” : “value_1”,
      ...
  }
}
```

auth_key
The `auth_key` that was given in response to `/api/login`
contract_id
The Blockchain address of the contract in question
transaction_function
The type of transaction to execute (that is, the function to call).
transaction_paramters
A list of key-value pairs that are parameters of the desired function call.

Response	
Status	**Response**
202	**An example response is:** `{ "message": "Message from underlying transaction." }`
400	{"error":"Missing or invalid request parameters."}
415	{"error":"Unsupported Media Type (JSON request body required)."}

401	{"error":"Unauthorized. Auth key is missing"}
402	{"error":"Undefined function specified"}
403	{"error":"Parameter mismatch with specified function"}
500	{"error":"An error occurred."}

5.3.4 Data quality enhancement services

5.3.4.1 Header Completeness

Via the unique external identifier, a submitted dataset is assessed regarding its completeness of all existing metadata fields.

Request	
Method	**URL**
POST	api/v1.0/quality/dq1/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"metadata completeness": 80,
	"description": "completeness score in %"
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.2 Understandability

Via the unique external identifier, a submitted dataset is assessed regarding its complexity based on the provided description field in its metadata.

Request	
Method	**URL**
POST	api/v1.0/quality/dq2/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"understandability score": 26,
	"description": "Easy"
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.3 Contactability

Via the unique external identifier, a submitted dataset is assessed regarding its completeness in terms of provided contact information.

Request	
Method	**URL**
POST	api/v1.0/quality/dq3/<process_id>/

Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	

Response	
Status	**Response**

202	**An example response is:**
	{
	"contactability": 0,
	"description": "contact information not available"
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.4 Temporal cohesion

Via the unique external identifier, a submitted dataset is assessed regarding its logical soundness in terms of the relationship between its creation data and the time period the datasets covers.

Request	
Method	**URL**
POST	api/v1.0/quality/dq4/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login`	

process_id The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"temporal cohesion": 0,
	"description": "creation date of dataset before last datum
	of dataset"
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.5 Openness

Via the unique external identifier, a submitted dataset is assessed regarding its openness of the attached license.

Request	
Method	**URL**
POST	api/v1.0/quality/dq5/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**
202	**An example response is:**

	{
	"openness score": 5,
	"description": "5-star rating due to CC-Zero license"
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.6 Language match

Via the unique external identifier, a submitted dataset is assessed regarding the language stated in its metadata and the actual language within the dataset.

Request	
Method	**URL**
POST	api/v1.0/quality/dq6/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"language match": 1,
	"description": "metadata field matches language of dataset
	}

400	{"error":"Process ID does not exist / is missing."}

401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.7 Currentness

Via the unique external identifier, a submitted dataset is assessed regarding its currentness (freshness) based on the relation of the current data, its release data, as well as the planned update cycle.

Request	
Method	**URL**
POST	api/v1.0/quality/dq7/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"currentness": 1.15,
	"update frequency lambda": 1.15,
	"processing time delta": 1.0
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.4.8 Format compliance

Via the unique external identifier, a submitted dataset is assessed regarding its actual format and the format stated in its metadata.

Request	
Method	**URL**
POST	api/v1.0/quality/dq8/<process_id>/
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key process_id
auth_key The `auth_key` that was given in response to `/api/login` **process_id** The `process_id` Process ID of the dataset submission process.	
Response	
Status	**Response**

202	**An example response is:**
	{
	"format compliance": 0,
	"description": "metadata field does not match dataset format"
	}

400	{"error":"Process ID does not exist / is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No asset with the given PID (pid) found."}
500	{"error":"An error occurred during registration."}

5.3.5 Generic services

5.3.5.1 Job status

The function for polling the job status is applicable for asynchronous requests which accept a job and queue it for execution. The asynchronous requests function returns a job_id parameter which can be used to query the status of the job execution.

Request	
Method	**URL**
GET	api/v1.0/jobs/<job_id>/status
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key job_id
auth_key The `auth_key` that was given in response to `/api/login` **job_id** The `job_id` that was received in response to a job submission	
Response	
Status	**Response**

202	{
	"status": "finished"
	"message": "Job finished successfully."
	}

400	{"error":"Job ID (job_id) parameter is missing."}
401	{"error":"Unauthorized. Auth key is missing"}
404	{"error":"No job with the given job ID (job_id) found."}
500	{"error":"An error occurred."}

5.3.5.2 Cancel job

Cancel a job that was accepted for execution and which is in status “running”. If the job is finished already, it will return status “finished”. Only if a job that was queued for execution was removed from the queue or if a running job was stopped, then the status “cancelled” is returned.

Request	
Method	**URL**
DELETE	api/v1.0/jobs/<job_id>
Parameters/Request Body	
Type	**Params**

HEAD GET	auth_key job_id

auth_key
The `auth_key` that was given in response to `/api/login`
job_id
The `job_id` that was received in response to a job submission

Response	
Status	**Response**

202	**Job cancelled:**
	`{ "status": "cancelled", "message": "Job cancelled." }`
	Job finished already:
	`{ "status": "finished", "message": "Job finished." }`

400	`{"error":"Job ID (job_id) parameter is missing."}`
401	`{"error":"Unauthorized. Auth key is missing"}`
404	`{"error":"No job with the given job ID (job_id) found."}`
500	`{"error":"An error occurred."}`

5.3.5.3 Access

WP5 is planning to build on the ResourceSync Framework Specification[1] for exposing metadata of available datasets of a node. In a similar way as OAI-PMH[2] this protocol allows exposing metadata to external parties.

The advantage of ResourceSync Framework compared to OAI-PMH is that it the enables sharing of both metadata and content with aggregators and search engines, i.e. it provides the means for synchronizing binary resources if needed. A corresponding python client and library implementation of ResourceSync is available on Github.[3]

[1] http://www.openarchives.org/rs/1.1/resourcesync
[2] https://www.openarchives.org/OAI/openarchivesprotocol.html
[3] https://github.com/resync/resync

5.3.5.3.1 List available resources of a DMA node

Returns a Resource List[1] as a snapshot of the DMA node's resources at a particular point in time. The invocation of "`api/v1.0/dataassets`" shown in the request below corresponds to the preferred method of exposing available resources which is described in section "6.3.4 robots.txt"[2]: to publish a `sitemap.xml` of available resources.

Request	
Method	**URL**
GET	api/v1.0/dataassets
Parameters/Request Body	
Type	**Params**
HEAD	auth_key
auth_key The `auth_key` that was given in response to `/api/login`	
Response	
Status	**Response**

200	**Resource List**
	
	`<?xml version="1.0" encoding="UTF-8"?>`
	`<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"`
	`xmlns:rs="http://www.openarchives.org/rs/terms/">`
	`<rs:ln rel="up"`
	`href="http://example.com/dataset1/capabilitylist.xml"/>`
	`<rs:md capability="resourcelist"`
	`at="2013-01-03T09:00:00Z"`
	`completed="2013-01-03T09:10:00Z"/>`
	`<sitemap>`
	`<loc>http://example.com/resourcelist1.xml</loc>`
	`<rs:md at="2013-01-03T09:00:00Z"/>`
	`</sitemap>`
	`<sitemap>`
	`<loc>http://example.com/resourcelist2.xml</loc>`
	`<rs:md at="2013-01-03T09:03:00Z"/>`
	`</sitemap>`
	`<sitemap>`
	`<loc>http://example.com/resourcelist3.xml</loc>`
	`<rs:md at="2013-01-03T09:07:00Z"/>`
	`</sitemap>`
	`</sitemapindex>`

[1] http://www.openarchives.org/rs/1.1/resourcesync#ResourceList
[2] http://www.openarchives.org/rs/1.1/resourcesync#robots

	Content of the Resource List identified by the URI http://example.com/resourcelist1.xml <?xml version="1.0" encoding="UTF-8"?> <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9" xmlns:rs="http://www.openarchives.org/rs/terms/"> <rs:ln rel="up" href="http://example.com/dataset1/capabilitylist.xml"/> <rs:ln rel="index" href="http://example.com/dataset1/resourcelist-index.xml"/> <rs:md capability="resourcelist" at="2013-01-03T09:00:00Z"/> <url> <loc>http://example.com/api/v1.0/dataassets/pid1</loc> <rs:md hash="md5:1584abdf8ebdc9802ac0c6a7402c8753" length="4385" type="text/turtle"/> </url> <url> <loc>http://example.com/api/v1.0/dataassets/pid2</loc> <rs:md hash="md5:4556abdf8ebdc9802ac0c6a7402c9881" length="883" type="text/turtle"/> </url> </urlset>
401	{"error":"Unauthorized. Auth key is missing"}
403	{"error":"Forbidden. The current user does not have access to perform the requested action"}
500	{"error":"An error occurred."}

5.3.5.3.2 Get metadata record of a data asset

Returns the Metadata record of a registered data asset.

Request	
Method	**URL**
GET	api/v1.0/dataassets/<pid>
Parameters/Request Body	
Type	**Params**
HEAD GET	auth_key pid
auth_key The `auth_key` that was given in response to `/api/login` **pid** The `pid` of the data asset.	

Response	
Status	**Response**
200	:catalog a dcat:Catalog ; dct:title "Imaginary Catalog" ; rdfs:label "Imaginary Catalog" ; foaf:homepage <http://example.org/catalog> ; dct:publisher :transparency-office ; dct:language <http://id.loc.gov/vocabulary/iso639-1/en> ; dcat:dataset :dataset-001 , :dataset-002 , \ :dataset-003 ; . :dataset-001 a dcat:Dataset ; dct:title "Imaginary dataset" ; dcat:keyword "accountability","transparency", \ "payments" ; dct:issued "2011-12-05"^^xsd:date ; dct:modified "2011-12-05"^^xsd:date ; dcat:contactPoint <http://example.org/transparency-office/contact> ; dct:temporal <http://reference.data.gov.uk/id/quarter/2006-Q1> ; dct:spatial <http://www.geonames.org/6695072> ; dct:publisher :finance-ministry ; dct:language <http://id.loc.gov/vocabulary/iso639-1/en> ; dct:accrualPeriodicity <http://purl.org/linked-data/sdmx/2009/code#freq-W> ; dcat:distribution :dataset-001-csv ; . [...] :dataset-001-csv a dcat:Distribution ; dcat:downloadURL <http://www.example.org/files/001.csv> ; dct:title "CSV distr. of imaginary dataset 001" ; dcat:mediaType "text/csv" ; dcat:byteSize "5120"^^xsd:decimal ; . [...]
400	{"error":"Data asset not found."}
401	{"error":"Unauthorized. Auth key is missing."}
403	{"error":"Forbidden. The current user does not have access to perform the requested action."}
404	{"error":"No asset found for the given PID (pid)."}
500	{"error":"An error occurred."}

6 Development and Deployment Roadmap

The development of WP5 components is divided into two main phases “Phase 1: Prototype Release” and “Phase 2: Final Release” according to the prototype and final release deliverables D5.3 “Initial Release of Foundational Data Technology Prototypes [M18]” and D5.4 “D5.4: Final Release of Foundational Data Technology Prototypes [M30]”.

6.1 Phase 1: Prototype Release

Regarding the development progress of the “Prototype Release”, there is a set of tasks and milestones which all of the components have in common (excluding the “Semantic enrichment” component which is developed in WP6) :

- “Prototype code development” task: The prototype code is published in the DMA code repository (Gitlab) and ready to be tested by other developers. There is documentation (minimum: README file) which allows installing the prototype and doing tests. Appropriate tests for the prototype is provided.
- “Sandbox demo deployment” task: The prototype code is deployed in the Sandbox demo environment. This task covers also dependencies on other components.
- “REST Service available” milestone: The REST web service is functioning according to the REST interface specification given in this document. However, this specification has draft status and deviations are therefore probable. Updated specifications will be part of the component documentation.
- “Finalizing demo” task: This task includes deployment/integration/data preparation activities which are required to prepare a demonstration in the sandbox environment.

Table 6 shows the development and deployment roadmap for WP5 components. Cells with blue background highlight component development, integration, or deployment tasks with their corresponding runtime in months, and the cells with red background denote milestones which must be completed at the beginning of the corresponding month.

Asset	2017									2018		
	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar
Prototype release												
- Conduit Data Ingest												
- Prototype code development												
- Sandbox demo deployment												
- REST Service available												
- PID Service & Preservation												
- Prototype code development												
- Sandbox demo deployment												
- REST Service available												
- Conduit integration												
- Data Management GUI												
- Prototype code development												
- Sandbox demo deployment												
- REST Service available												
- Conduit integration												
- Finalizing demo												
- Blockchain Sec. & Prov.												

- Prototype code development												
- Sandbox demo deployment												
- Conduit integration												
- REST Service available												
- Finalizing demo												
- Data Quality Services												
- Prototype code development												
- Sandbox demo deployment												
- Conduit integration												
- REST Service available												
- Finalizing demo												
- Semantic Enrichment												
- Conduit integration												
- REST Service available												
- Finalizing demo												
- Demo of prototype ready												

Table 6 Development and deployment roadmap for the prototype release WP5 components

6.2 Phase 2: Final Release

The goal of the final release is to create software components which are adequate for "production deployment" (deployment in an environment which is close to the targeted DMA marketplace production environment). Regarding the development progress of the "Final Release", there are tasks and milestones which all of the components have in common (excluding the "Semantic enrichment" component which is developed in WP6) :

- "Final release code improvement" task: The prototype code is improved in order to reach a maturity level for production deployment. Instructions how to deploy the component in the production environment are provided.
- "Initial production deployment" task: The first deployment an environment which is close to the targeted DMA marketplace production environment. This task covers also dependencies on other components.
- "Conduit integration" milestone: Component and service adaptations which are required to integrate components with the Conduit data set ingest processing pipeline.
- "Production deployment" task: The final deployment an environment which is close to the targeted DMA marketplace production environment. This task covers also dependencies on other components.

Table 6 shows the development and deployment roadmap for WP5 components

Asset	2018									2019		
	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar
Final release												
- Conduit Data Ingest												
- Final release code improvem.												
- Initial production deployment												
- PID Service & Preservation												
- Final release code improvem.												
- Initial production deployment												
- Conduit integration												

- Data Management GUI												
- Final release code improvem.												
- Initial production deployment												
- Conduit integration												
- Production deployment												
- Blockchain Sec. & Prov.												
- Final release code improvem.												
- Initial production deployment												
- Conduit integration												
- Production deployment												
- Data Quality Services												
- Final release code improvem.												
- Initial production deployment												
- Conduit integration												
- Production deployment												
- Semantic Enrichment												
- Conduit integration												
- Production deployment												
- Enhanced blockchain security												
- Production deployment ready												
- Final release												

Table 7 Development and deployment roadmap for the final release WP5 components

7 Conclusion

This document presented the DMA Data Technology Specification and Development Roadmap regarding the ongoing work in WP5 of the DMA project. The purpose of this document was to show that basic technologies and frameworks have been selected and that the necessary technical specifications are in place to support the development, integration, and deployment process.

It must be noted that the technical specification is in draft status. The specifications will be continuously adapted according to new requirements coming up, especially during the integration and deployment phases.

8 Appendices

8.1 Appendix A: HTTP Status Codes

All status codes are standard HTTP status codes. The below ones are used in this API.

2XX - Success
4XX - Error occurred in client's part
5XX - Error occurred in server's part

Status Code	Description

200	OK
201	Created
202	Accepted (Request accepted, and queued for execution)
400	Bad request/Not found
401	Authentication failure
403	Forbidden
404	Resource not found
405	Method Not Allowed
409	Conflict
412	Precondition Failed
413	Request Entity Too Large
500	Internal Server Error
501	Not Implemented
503	Service Unavailable