NGI ONTO CHAIN

Blockchain for the Next Generation Internet

BABELFISH

D1. USE CASE SCENARIOS, DETAILED TECHNICAL SPECIFICATION, IMPLEMENTATION WORK PLAN, AND DEPLOYMENT PLAN (DRAFT)

02/12/2022

NGI ONTO CHAIN

Grant Agreement No.: 957338
Call: H2020-ICT-2020-1

Topic: ICT-54-2020
Type of action: RIA

BABELFISH

D1. Use case scenarios, detailed technical specification, implementation work plan, and deployment plan (draft)

DUE DATE	02/12/2022
SUBMISSION DATE	02/12/2022
TEAM	OwnYourData & Kybernos
VERSION	1.0
AUTHORS	Christoph Fabianek, Sebastian Haas

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

EXECUTIVE SUMMARY

Alluding to the project name, Babelfish comes from Hitchhiker's Guide to the Galaxy by Douglas Adams. Babelfish is a small yellow fish that when placed in someone's ear translates any language into their first language. This project proposes to describe services on a technical, semantic, and governance layer and will implement a component that uses such descriptions to translate interfaces (APIs), data, and data agreements from a foreign (and maybe proprietary format) to an interoperable format understood by the recipient. A registry maintains a list of all services and thus spans up an interoperable data space.

In this document we identify relevant stakeholders and map those on requirements based on the scope of a Gateway API, ONTOCHAIN's previous work, and aspects from Supply Chain Management (SCM). To complete the picture, we formulate user stories from a technical perspective as well as the SCM context. A state-of-the-art analysis includes relevant standards and already existing software components that will be used in the course of the development: (i) Semantic Container (SemCon) for technical interoperability - enabling to connect APIs and handling authentication as well as processing aspects; (ii) Semantic Overlay Architecture (SOyA) for data harmonisation - using RDF and JSON-LD for data model management and aligning & transforming data models on the fly; (iii) Data Agreements - covering data governance, tracking provenance metadata and act as usage policies between services describing rules for the use of data.

A chapter on software design and component specification provides the current (preliminary) implementation plan for developing the Gateway API and encompasses (i) a Service Description that will be used by participants of the ONTOCHAIN environment to form a service catalogue, (ii) the Gateway API component called Babelfish to process Service Descriptions, and (iii) a concrete use case in SCM to demonstrate the planned features. A work plan and outlook of the next steps concludes this deliverable.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

LIST OF FIGURES

LIST OF TABLES

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ABBREVIATIONS

API	Application Programming Interface
DID	Decentralised Identifier
DRI	Decentralised Resource Identifier
FMCG	Fast Moving Consumer Goods
IP	Internet Protocol
SCM	Supply Chain Management
SDG	Sustainable Development Goals
TCP	Transmission Control Protocol
VC	Verifiable Credential
VP	Verifiable Presentation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

1 INTRODUCTION

With the Gateway API component in the ONTOCHAIN environment we will address the challenge of interoperability in a heterogeneous environment. Interoperability by itself provides overall system benefits at different, distinct dimensions and a common approach[1] is to distinguish between the following aspects: technical (connectivity), semantic (informational), and organisational (governance, business models etc.)

- On the technical level connectivity, syntactics, and protocols for data exchange (e.g., APIs) and data storage underpin basic integration;
- the semantic level requires harmonised information with shared data models and mutually agreed content; and
- the organisational level (usually only addressed in more mature ecosystems) encompasses shared objectives and policies between organisations.

In this document the first version of our planned approach for the Gateway API is described. We identified relevant requirements to conclusively answer the question of WHAT to build and also addressed our planned approach to validate developments in the supply chain management domain. Existing components used as a basis for developing the software artefacts are described and provide the foundation for reaching the challenging goals of this project.

An overview architecture diagram and work plan conclude the document and will be complemented in February 2023 by deliverable D2 Design.

[1] 'Architecture constraints for Interoperability and composability in a smart grid', Power and Energy Society General Meeting, 2010 IEEE. https://www.researchgate.net/publication/224178883_Architecture_constraints_for_Interoperability_and_composability_in_a_smart_grid

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2 Motivation and Planned Functionalities

This section presents the performed work in the project regarding current status and requirements elicitation. In the course of the project the following steps were already performed respectively are planned in the next months:

- research and familiarise with NGI ONTOCHAIN ecosystem
- identify relevant list of stakeholders and describe their needs as well as their environment where they operate and make decisions
- describe requirements (this document) and deduce the Design Specification (D2, due in February 2023)
- setup a dedicated test system for deploying and verifying available components and iteratively feedback any learnings
- deploy solution with partners and other ONTOCHAIN participants to collect further feedback

2.1 Stakeholder

Initially, a list of relevant stakeholders was compiled:

- Consumers access information (tagged as publicly available) collected along the value chain
  (tag: ind)
- Entities generating events (i.e., data from organisations) and authorised parties accessing data to facilitate interoperability
  (tag: event)
- Regional stakeholders, regulators, authorities, and institutions provide the framework for business
  (tag: reg)
- Technical infrastructure that enables interoperability / builds a common data space for all actors
  (tag: infra)

All requirements were mapped to at least one of those stakeholders to document source and motivation. During the course of the project multiple data flows will be implemented that demonstrate the interaction of above stakeholders.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2.1.1 Supply Chain Management Stakeholder Goals

For the Supply Chain Management use-case the list of stakeholders above can be described in more detail:

- "direct, events-producing" supply chain parties (i.e. institutions, organisations, enterprises): any authorised stakeholder to the trade transaction should be able to access the same data in order to facilitate multi-modal transport and interoperability in the exchange of data across varying modes of transport platforms
    - production parties (farmers, producers, processors / refiners)
    - transport parties (shippers / forwarders / carriers)

Figure 2.1: Event Producers

- "indirect, systems' minting" regional stakeholders & regulators: regulator-driven business choreographies tend to be quite similar even though terminology for business steps, documents, objects and entities may be different across the modes of transport and also sometimes across sectors dealing with specific types of goods/trade items; the fact that the choreographies are similar is a key enabler for connecting the choreographies that currently exist only in specific modes or industry sectors
    - local and regional authorities ("Gebietskörperschaften")
    - national authorities

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

- EU institutions
- sector associations
- UN work programme for 2023:
  - Market transparency – define a common trading language, with minimum quality requirements for agricultural produce that facilitate fair and sustainable trade, prevent technical barriers to trade and contribute to economic growth (SDG 8)
  - Food security – promote the sustainable production and consumption of quality agricultural produce, including the prevention of food loss (SDG 2, SDG 12);

- Freight Data Space Interoperability Infrastructure
- Consumers

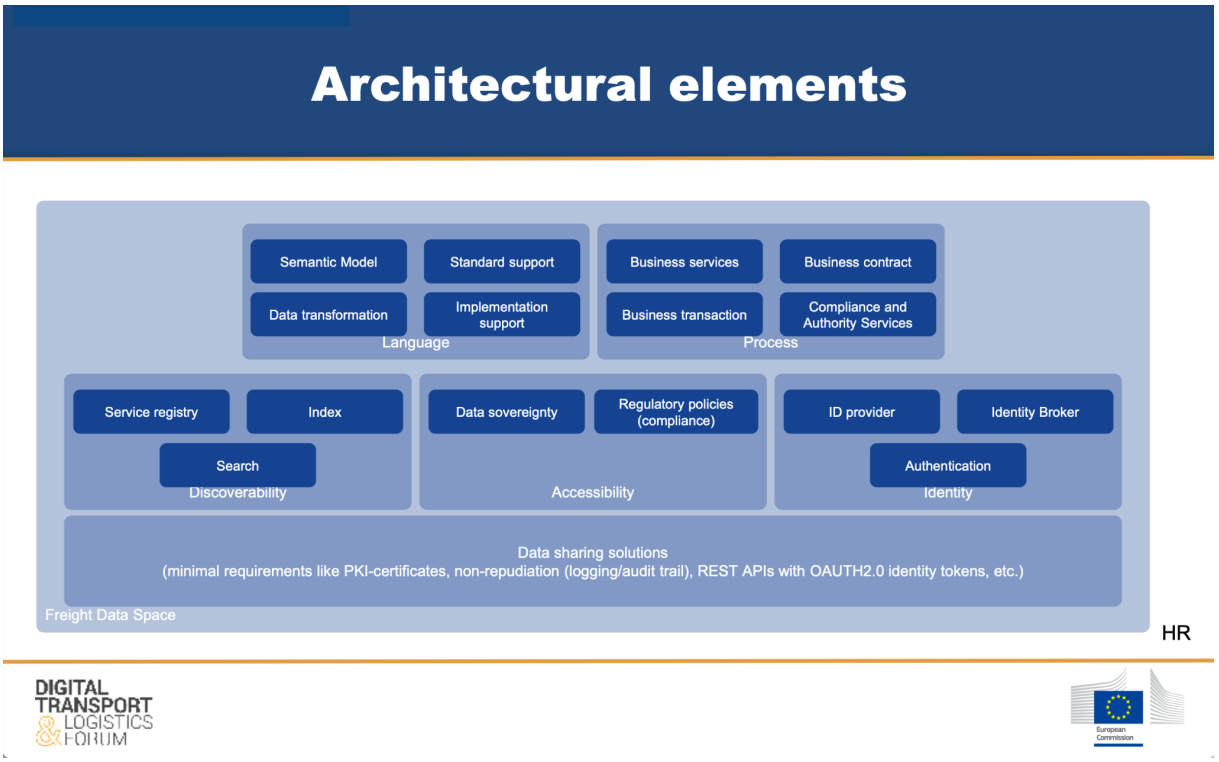

Figure 2.2: Architectural Elements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2.2 Non-functional Requirements

2.2.1 Performance and Scalability

Requirements that describe throughput under a given workload for a specific time frame in each setting.

ID	Tags	Description
perf_1	infra	The GatewayAPI shall handle at least 1.000 registered services.
perf_2	infra	A single instance of the Gateway API shall handle at least 100 requests per minute.

Table 2.1: Performance and Scalability Requirements

2.2.2 Portability and Compatibility

Requirements to make sure that the system can be operated now and in the foreseeable future on the available platform infrastructure and also works together with adjacent systems.

ID	Tags	Description
port_1	infra, reg	Available standards and best practices for the respective areas should be identified and adhered to.
port_2	infra	Data exchange between building blocks shall use JSON.
port_3	infra	Interfaces of the different components shall be clearly defined and documented.

Table 2.2: Portability and Compatibility Requirements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

2.2.3 Reliability, Availability, and Maintainability

Requirements describing the accessibility of the system to the users at a given point in time and how to quickly recover from any failures.

ID	Tags	Description
rel_1	infra	Components shall be easy to deploy and configure. (In this project Docker containers are the preferred way to make the developed software artefacts available.)
rel_2	infra	All software components shall be documented.
rel_3	infra	The system shall perform input validation.

Table 2.3: Reliability, Availability, and Maintainability Requirements

2.2.4 Privacy and Security

Requirements about privacy (safeguarding data) and security (authorization and protection) needs from different stakeholders.

ID	Tags	Description
priv_1	ind, event	All external data transfer shall be encrypted through TLS for communication over a network.
priv_2	event	Consent information for data exchange (Usage Policies and Data Agreements) shall be inseparably linked to the payload.

Table 2.4: Privacy and Security Requirements

2.3 Service Discovery

As defined in ONTOCHAIN's D3.5 Framework Specification (section 3.2 Service Discovery) the following API endpoints are required.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

This module will provide a programmable way of discovering the services provided through the ONTOCHAIN ecosystem. Access to the functions will be authenticated and regular users will only have access to read operations. Write operations will be accessible only to administrators, unless noted in the function's description.

ID	Tags	Description
sd_1	event	`GET <base-uri>/list/count/page?filter=JSON` Obtain a list of all the available services as a JSON string. Arguments count (max. 100) and page are used to control the number of results. An optional argument can be provided to filter the types of desired services; the argument is a url-encoded JSON document that can contain every field of the resource description schema. Returns a JSON document with the desired result count (or less) and HTTP code 200 in case of success, HTTP 400 if the provided filter cannot be decoded, and code 401 if the user is not authenticated.
sd_2	event	`GET <base-uri>/service/search?query=QUERY` Obtain a list of services which name or description match the provided search terms. The QUERY argument can contain one or several url-encoded terms. Returns a JSON document with matching project descriptions and HTTP code 200 in case of success, HTTP 400 if QUERY is absent, and code 401 if the user is not authenticated.
sd_3	event	`GET <base-uri>/service/SERVICE_ID` Obtain the detailed description of a service as a JSON string. The argument is the service identifier that can be obtained with the /list function (requirement sd_1). Returns a JSON document and HTTP code 200 in case of success, HTTP 404 if the requested service cannot be found, and code 401 if the user is not authenticated.
sd_4	event	`POST <base-uri>/service/` Add a new service to the catalogue. The request body must contain a JSON document containing the complete service description. Returns HTTP code 200 in case of success, HTTP 400 in case the description has an incorrect format, and HTTP 401 if the user is not authenticated or does not have the required privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
sd_5	event	`PUT <base-uri>/service/SERVICE_ID` Update the details of a service already stored in the catalogue. The function takes the identifier of the service that must be updated as an argument and the request body must contain a JSON document containing the new service description. Returns HTTP code 200 in case of success, 400 in case the description has an incorrect format, 404 if the requested service does not exist, and code 401 if the user is not authenticated or does not have the required privileges.
sd_6	event	`DELETE <base-uri>/service/SERVICE_ID` Removes the service pointed by argument SERVICE_ID from the catalogue. The service is not actually removed, it will only be marked as deleted and it will no longer be part of the results of the /list and /search functions (requirements sd_1 and sd_2).

Table 2.5: Service Discovery Requirements

2.4 Organisation and User Accounts

As defined in ONTOCHAIN's D3.5 Framework Specification (section 3.3 Organizations and User Accounts) the following API endpoints are required.

> This module provides interfaces for creating users and organisations, and basic interactions with wallets. All API calls must be authenticated and write operations are accessible only to administrators, unless specifically noted.

ID	Tags	Description
ou_1	event	`POST <base-uri>/organization/` Create a new organisation with no users. The request body must contain the details of the organisation as a JSON document (e.g. its name, short description, contact address). The call returns the identifier of the new organisation and HTTP code 200 in case of success. The call can also return HTTP code 400 if the organisation description is not a valid

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
		JSON document or if required fields are missing, and code 401 if the user is not authenticated.
ou_2	event	`PUT <base-uri>/organization/ORGANIZATION_ID` Updates the information related to an existing organisation. The identifier of the organisation must be provided in the query string, and the request body must contain the new details of the organisation as a JSON document. The call returns HTTP code 200 in case of success, code 400 if the organisation description is not a valid JSON document or if required fields are missing, 404 if the organisation does not exist, and code 401 if the user is not authenticated.
ou_3	event	`GET <base-uri>/organization/ORGANIZATION_ID` Obtain the details related to an existing organisation. The call takes the identifier of the organisation as an argument and returns a JSON document containing the organisation's information. The call returns HTTP error code 200 in case of success, 404 if the organisation does not exist, and 401 if the user is not authenticated.
ou_4	event	`GET <base-uri>/organization/ORGANIZATION_ID/list` Obtain the list of users that are members of an organisation. The identifier of the organisation is taken as the sole argument to the call. The call returns a JSON array of objects describing the users. In case of success, the call returns HTTP code 200; the call can also return code 404 if the organisation cannot be found, and 401 if the user is not authenticated.
ou_5	event	`DELETE <base-uri>/organization/ORGANIZATION_ID` Request the deletion of an organisation. The organisation will be marked as deleted but no data will actually be removed, in order to preserve the history of the platform. If the organisation contains users, the users will not be removed but will appear as independent. The only argument to the call is the identifier of the organisation. In case of success, the call returns HTTP code 200. The call can also return code 404 if the organisation does not exist and code 401 if the user is not authenticated or does not have the required privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
ou_6	event	`POST <base-uri>/user/` Create a new user. The request body must contain the details of the user as a JSON document (e.g. its name, organisation, email address, phone number). The call returns the identifier of the new user and HTTP code 200 in case of success. The call can also return HTTP code 400 if the user description is not a valid JSON document or if required fields are missing, and code 401 if the requesting user is not authenticated.
ou_7	event	`PUT <base-uri>/user/USER_ID` Updates the information related to an existing user. The identifier of the user must be provided in the query string, and the request body must contain the new details of the user as a JSON document. The call returns HTTP code 200 in case of success, code 400 if the user description is not a valid JSON document or if required fields are missing, 404 if the user does not exist, and code 401 if the requesting user is not authenticated.
ou_8	event	`GET <base-uri>/user/USER_ID` Obtain the details related to an existing user. The call takes the identifier of the user as an argument and returns a JSON document containing the user's information. The call returns HTTP code 200 in case of success, 404 if the user does not exist, and 401 if the requesting user is not authenticated.
ou_9	event	`DELETE <base-uri>/user/USER_ID` Request the deletion of a user. The user will be marked as deleted and all of associated information (except its identifier, in order to preserve the history of the platform) will be removed. The only argument to the call is the identifier of the user. In case of success, the call returns HTTP code 200. The call can also return code 404 if the user does not exist and code 401 if the requesting user is not authenticated or does not have the required privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
ou_10	event	`GET <base-uri>/user/USER_ID/wallet` Obtain the details (chain, address, balance) of the user's wallets in the ONTOCHAIN ecosystem. The call takes the identifier of the user as an argument and returns the wallet information as a JSON document along with HTTP code 200 in case of success. The call can also return code 404 if the user does not exist or has been deleted, and code 401 if the requesting user is not authenticated or does not have the required privileges.

Table 2.6: Organisation and User Accounts Requirements

2.5 Storage Service Results/Output

As defined in ONTOCHAIN's D3.5 Framework Specification (section 3.4 Storage Service Results/Output) the following API endpoints are required.

> The module provides users and applications with high-level interfaces that are common to all of the supported backend storage services, both members of the ONTOCHAIN ecosystem, and external services. Storage providers can be searched in the Service catalogue like any other service.

ID	Tags	Description
ssr_1	event	`POST <base-uri>/collection/` Create a new collection of objects, similar to a directory in a file system. The call takes a description of the collection (including the storage provider) in the JSON format as an argument. The call only creates a record for the collection, it does not store any data object. In case of success, the call returns the identifier of the new collection and HTTP code 200. In case of failure, the call returns code 400 (if the JSON document cannot be interpreted) and 401 if the user is not authenticated or does not have the required privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
ssr_2	event	`PUT <base-uri>/collection/COLLECTION-ID` Updates the information related to an existing collection of objects. The identifier of the collection must be provided in the query string, and the request body must contain the new details of the collection as a JSON document. The call returns HTTP code 200 in case of success, code 400 if the collection description is not a valid JSON document or if required fields are missing, 404 if the collection of object does not exist, and code 401 for unauthorised access.
ssr_3	event	`GET <base-uri>/collection/COLLECTION-ID` Obtain the details related to an existing collection of objects. The call takes the identifier of the collection of objects as an argument and returns a JSON document containing the information of the corresponding collection of objects. The call returns HTTP code 200 in case of success, 404 if the collection of objects does not exist, and 401 for the unauthenticated access request.
ssr_4	event	`GET <base-uri>/collection/list` Obtain the list of collection of objects. The call returns a JSON array of objects describing the collection of objects. In case of success, the call returns HTTP code 200; the call can also return code 404 if the collection of objects cannot be found, and 401 for unauthenticated access requests.
ssr_5	event	`DELETE <base-uri>/collection/COLLECTION-ID` Request the deletion of a collection of objects. The collection of objects will be marked as deleted but no data will actually be removed, in order to preserve the history of the platform. If the collection of objects contains users, the users' information will not be removed but will appear as independent. The only argument to the call is the identifier of the collection. In case of success, the call returns HTTP code 200. The call can also return code 404 if the collection ID does not exist and code 401 if the requester (user) is not authenticated or does not have the required privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
ssr_6	event	`POST <base-uri>/object/` Create a new object. The call takes a description of the collection (including the storage provider) in the JSON format as an argument. The call only creates a record for the object. In case of success, the call returns the identifier of the new object and HTTP code 200. In case of failure, the call returns code 400 (if the JSON document cannot be interpreted) and 401 if the user is not authenticated or does not have the required privileges.
ssr_7	event	`PUT <base-uri>/object/OBJECT_ID` Updates the information related to an existing object. The identifier of the object must be provided in the query string, and the request body must contain the new details of the object as a JSON document. The call returns HTTP code 200 in case or success, code 400 if the object description is not a valid JSON document or if required fields are missing, 404 if the object does not exist, and code 401 for unauthorised access.
ssr_8	event	`GET <base-uri>/object/OBJECT_ID` Obtain the details related to an existing object. The call takes the identifier of the object as an argument and returns a JSON document containing the information of the corresponding object. The call returns HTTP code 200 in case of success, 404 if the object does not exist, and 401 for the unauthenticated access request.
ssr_9	event	`DELETE <base-uri>/object/OBJECT_ID` Request the deletion of an object. The object will be marked as deleted but no data will actually be removed, in order to preserve the history of the platform. If the object contains users, the users' information will not be removed but will appear as independent. The only argument to the call is the identifier of the object. In case of success, the call returns HTTP code 200. The call can also return code 404 if the object ID does not exist and code 401 if the requester (user) is not authenticated or does not have the required privileges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
ssr_10	event, ind	`GET <base-uri>/object/OBJECT_ID/read` Obtain the details related to an existing object. The call takes the identifier of the object as an argument and returns information of the corresponding object. The call returns HTTP code 200 in case of success, 404 if the object does not exist, and 401 for an unauthenticated access request.
ssr_11	event	`PUT <base-uri>/object/OBJECT_ID/write` Updates the information related to an existing object. The identifier of the object must be provided in the query string, and the request body must contain the new details of the object as a document. The call returns HTTP code 200 in case of success, code 400 if the object description is not valid or if required fields are missing, 404 if the collection of object does not exist, and code 401 for unauthorised access.
ssr_12	event	`POST <base-uri>/object/OBJECT_ID/USER_ID` Checks the access control for a particular user to access a particular object. If the user has the access grant for accessing the object it returns HTTP code 200. Returns code 400 if the object id or user id are not valid or missing, 404 if the object does not exist, and code 401 for unauthorised access.

Table 2.7: Storage Service Results/Outputs Requirements

2.6 Interoperability

The Integration Helper in the Babelfish component provides mechanisms to support interoperability between services and applications.

ID	Tags	Description
io_1	event, infra	A System Description shall document all required information for covering technical, semantic, and governance interoperability aspects.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

ID	Tags	Description
io_2	event, infra	The system shall allow to connect API endpoints from different services specified in their System Description to allow for technical interoperability.
io_3	event, infra	The system shall be able to transform between data models from different services specified in their System Description to allow for semantic interoperability.
io_4	event, infra	The system shall compare Usage Policies from different services in their System Description to evaluate compliance.
io_5	event, infra	The system shall generate a Data Agreement between compliant services based on their Service Description to cover the governance interoperability aspect.

Table 2.8: Interoperability Requirements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

3 User Stories and Use Case Analysis

This chapter provides user stories based on the requirements from the previous chapter. It is split between the technical view for implementing the Gateway API and the perspective of supply chain actors as a sample use case.

3.1 User Stories for Gateway API

Service Discovery

As a service developer I want to be able to register a service in a service catalogue so that others can easily discover it.

As a service developer I want to be able to update and delete my existing entries in a service catalogue so that I can keep everything up-to-date.

As a developer I want to be able to query the service catalogue so that I'm able to discover available services.

Organisation and User Accounts

As an organisation I want to be able to create an entry in a registry so that I can access services with this identity.

As an organisation I want to be able to update and delete my organisation entry in the registry so that I can keep it up-to-date.

As an organisation I want to be able to retrieve my stored data so that I can verify the data.

As a user I want to be able to create an entry in a registry and assign it to an organisation so that I can access services with this identity.

As a user I want to be able to update and delete my user entry in the registry so that I can keep it up-to-date.

As a user I want to be able to retrieve my stored data (incl. configured chain, address, balance) so that I can verify the data and have the latest version.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Storage Service Results/Output

As a developer I want to be able to create a collection so that I can store objects in the collection.

As a developer I want to be able to update and delete collection entries so that I can keep them up-to-date.

As a developer I want to be able to retrieve all information of a collection and a list of all accessible collections so that I can verify the latest state.

As a developer I want to be able to create objects in a collection so that I can have a flexible storage mechanism.

As a developer I want to be able to update and delete objects in a collection so that I can keep them up-to-date.

As a developer I want to be able to retrieve the details about an object so that I can work with the data and have the latest version.

Interoperability

As a service/application I want to be able to document my API endpoints, used data model, and applicable usage policies in a Service Description so that this information can be used for seamless integration with other services/applications.

As a service/application I want to be able to connect to other services/applications independent of the respective API endpoints so that I can exchange data without technical integration limits.

As a service/application I want to be able to connect to other services/applications independent of the respective data models so that I can exchange data without semantic integration limits.

As a service/application I want to be able to have usage policies automatically validated upon data exchange so that I can exchange data with full governance conformance.

As services/applications that exchange data we want to be able to have this data exchange documented in a data agreement so that full governance conformance is unambiguously documented.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

3.2 User Stories and Context for Supply Chain Management Use Case

Product Owner

As a supply chain party I want to be able to share and receive relevant data so that this data can be used by other supply chain parties for planning, optimization and documentation purposes.

Transport Service Provider

As a transport service provider I want to be able to share and receive relevant data so that this data can be used by other supply chain parties for planning, optimization and documentation purposes.

Supply Chain Party

As a supply chain party I want to be able to share and receive relevant data so that this data can be used by other supply chain parties for planning, optimization and documentation purposes.

Market Maker

As a market maker I want to be able to provide structured food product information so that this data can be used by customers for market and product differentiation.

Regulator

As a regulator I want to be able to design, co-create and mint purpose-driven data circles within the supply chain data space so that the shared data can be (re-)used for data-driven circular economy policy schemes like the Digital Product Passport and supply chain management regulations in general.

Region

As a region I want to be able to document and shape public procurement practices to be able to reach net zero targets for the food and transport sector.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Data Intermediation Service Provider DISP for Digital Product Passport

As a data intermediation service provider I want to be able to structure and govern a product data space around the regulator-driven design artefact Digital Product Passport so that this structured and incentivised data exchange can be used by consumers, regulators, investors and insurance companies for their purposes.

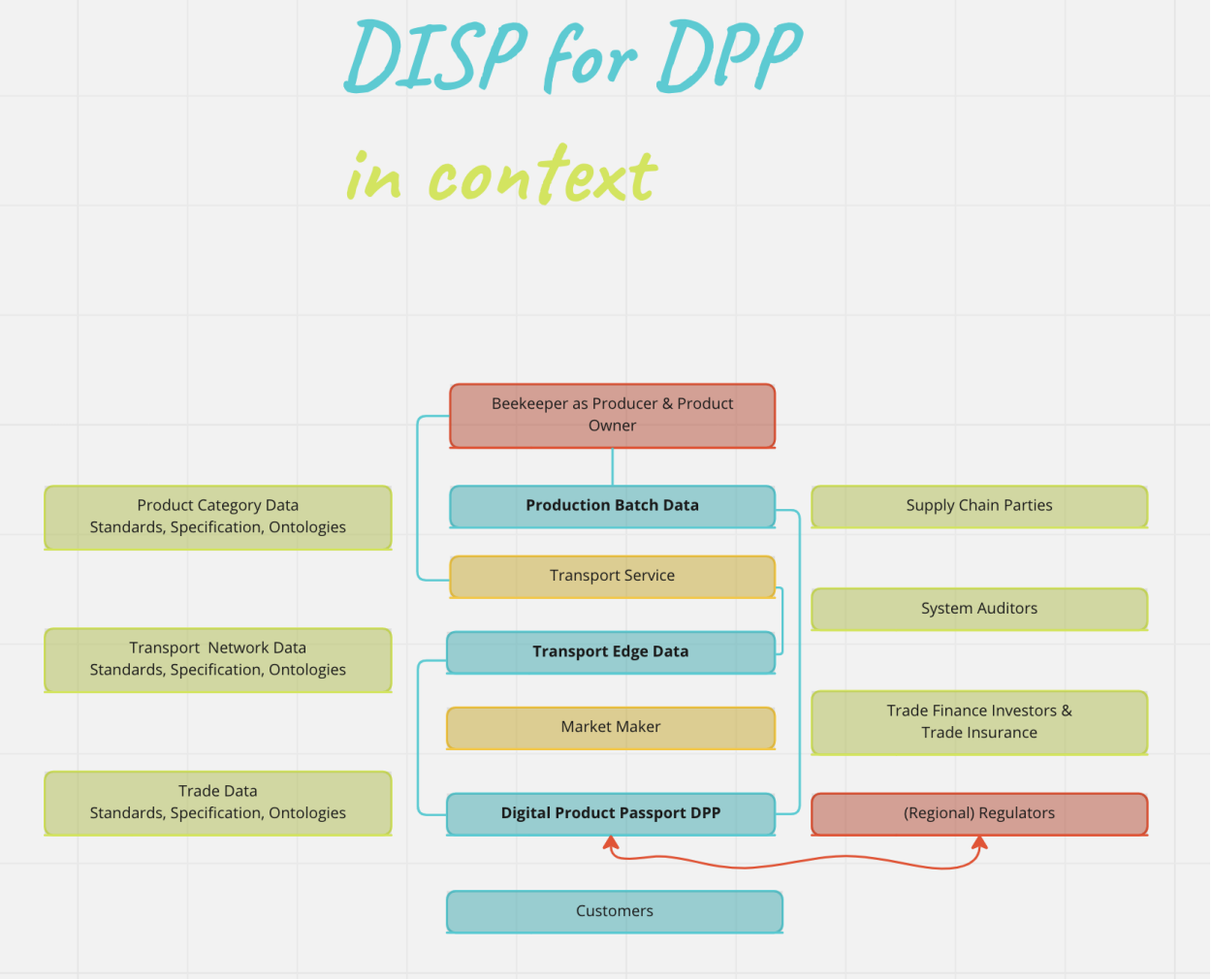

Figure 3.1: Supply Chain Management Context

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

4 State of the Art Analysis, Background, and Innovation

This section is dedicated to the analysis of existing solutions, including developments, and demonstrating the innovation potential of the solution compared to what already exists.

4.1 State of the Art Analysis

In the course of the project the following standards were identified and are adhered to in developing the system:

- **JSON-LD** v1.1 a light-weight Linked Data format
  described in detail here: https://www.w3.org/TR/json-ld/
- **Decentralised Identifier** (DIDs) for managing identities of entities;
  described in detail here: https://www.w3.org/TR/did-core/
- **Verifiable Credentials** (VCs) expressing credentials on the Web in a way that is cryptographically secure, privacy respecting, and machine-verifiable
  described in detail here: https://www.w3.org/TR/vc-data-model/
- **Open API Specification** v3 is a programming language-agnostic interface description for REST APIs
  described in detail here: https://spec.openapis.org/oas/v3.1.0
- **EPCIS** (Electronic Product Code Information Services) enables disparate applications to create and share visibility event data, both within and across institutions, organisations, enterprises, and supply chain parties
  described in detail here: https://ref.gs1.org/standards/epcis/
- **Buy-Ship-Pay BSP Reference Data Model** is the global dictionary for terms in trade. The UN/CEFACT vocabulary is essential for bringing the rich BSP semantic model for trade to the modern web environment
  more information here: https://tfig.unece.org/contents/buy-ship-pay-model.htm
- **GoodRelations** is a powerful vocabulary for publishing details of products and services in a way friendly to search engines, mobile applications, and browser extensions
  more information here: https://www.w3.org/wiki/GoodRelations

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

In addition, the following specifications were used or will be further developed:

- **Data Agreements** record conditions for an organisation to process data in accordance with a privacy regulation (e.g. GDPR)
  described in detail here:
  https://github.com/decentralised-dataexchange/automated-data-agreements/blob/main/docs/data-agreement-specification.md
- **UNCEFACT** and Traceability APIs (API definitions, JSON-LD vocabulary, and testable mocks for a UN standard supply chain traceability service based on GS1 EPCIS)
  described in detail here: https://github.com/uncefact/project-traceability
- **did:oyd Method (OYDID)** a self-sustained environment to manage decentralised identifiers
  described in detail here: https://ownyourdata.github.io/oydid/
- **Semantic Overlay Architecture (SOyA)** data model authoring and publishing platform
  described in detail here: https://ownyourdata.github.io/soya/

4.2 Description of Background

4.2.1 Semantic Container

Semantic Containers provide a standardised infrastructure for data provisioning and allow data providers to efficiently distribute data without giving up control over its usage and monetization while providing data consumers with efficient and well-managed mechanisms to obtain and integrate data in a trustworthy and reproducible manner. By packaging data and processing capabilities into reusable containers, describing the semantics of the content and permissible usage, and providing uniform interfaces, a data set becomes a commodity with well-defined content, properties, quality, and usage policy, as well as clear ownership rights and a price tag.

The Semantic Container approach leverages existing container technologies such as Docker, which already provide scalable mechanisms for deploying complex software assemblies and use them as a foundation for an infrastructure for data discovery, provisioning, and integration. To create a suitable environment for the emergence of a commodity market around data, a set of rules for permissible usage of the data is captured in semantic descriptions, provides cryptographic methods to prove ownership rights, and applies blockchain technology to guarantee immutability.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Complete audit trails of data sources and processing steps provide gapless provenance and facilitate reproducibility.

4.2.2 did:oyd Method (OYDID)

The aim of the did:oyd method is to provide a decentralised identifier (DID) that is not based on a distributed ledger. Many DID methods are based on blockchain technology and provide a trust anchor based on the respective governance of the used ledger for handling sensitive data. For certain aspects however, it could be interesting to have DIDs that don't require the full stack of a decentralised system. OYDID provides such a self-sustained environment for managing digital identifiers. The did:oyd method links the identifier cryptographically to the DID Document and through also cryptographically linked provenance information in a public log it ensures resolving to the latest valid version of the DID Document.

A W3C conform DID Method Specification is available here:
https://ownyourdata.github.io/oydid/

4.2.3 Semantic Overlay Architecture (SOyA)

SOyA allows data structures to be described in simple terminology. This description includes groups of data records with the same attributes, references between data records, and meta-attributes of these data structures.

Datasets are referred to as "bases", while meta-attributes are summarised in so-called "overlays". Overlays can contain information about attributes (e.g. detailed descriptions, permissible values, or formatting), but also transformations into other data structures.

For the exchange of these definitions of data structures, those structures can be stored in online repositories. When saving in such repositories, 2 versions are created:

- the original variant with the given names of the individual artefacts (i.e. of Structure, Base, and Overlay)
- a "frozen" variant where each name is replaced by a DRI; the DRI is a content-based address, which is also the unchangeable fingerprint of the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

content. This ensures that the previous version remains available if changes are made later.

With the described definitions of data structures, concrete data can now be recorded. SOyA offers the following functions:

- Acquire: If the attributes are named accordingly in a simple JSON ("flat JSON"), a conversion into JSON-LD can take place automatically
- Validate: Existing data records can be checked for conformity using a Validate overlay
- Transform: you can switch between data structures with a transformation overlay; It is thus possible to retain existing data formats (legacy formats), while automatic mapping to new standards is guaranteed
- Capture: with automatically generated HTML forms based on the structure information, data can be conveniently visualised, recorded and processed

A W3C conform Community Group Specification is available here: https://ownyourdata.github.io/soya/

4.3 Innovation Compared to the State of the Art

This section describes how Babelfish advances the state of the art identified in the two previous sections.

The Gateway API will be implemented using the framework of Semantic Containers as a basis and leverages existing functionality to interpret and semantically annotate data as gateway to an actual storage provider. New functionality will include authentication of organisations and users, as well as parsing a service description with technical, semantic, and governance annotation to aid applications with integration.

This service description can be provided by other services and applications in the ONTOCHAIN ecosystem and will form a service catalogue that describes the dataspace formed by all participating actors. The technical aspects of a service are described using the Open API Specification v3.0 (Swagger), semantic aspects build on the Semantic Overlay Architecture (SOyA) and allow to describe data models together with information about alignments with other ontologies as well as transformation between used schemas, and governance aspects are described with usage policies following the structure proposed by the Data Privacy Vocabulary.

The integration helper is able to parse the information in the service description provided by each service and automatically map APIs, translate data, and ensure any

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

restrictions defined in Usage Policies. The actual data exchange is then documented in the provenance trail and as a Data (Sharing) Agreement also available as metadata.

Overall will the process of data exchange via the Gateway API be documented using tutorials in the Supply Chain Management domain. Based on inputs from our partners at Clever Clover (https://cleverclover.vc/, Europe's #1 FMCG Accelerator) and established ontologies from the mobility domain, examples for Service Description will provide real world use cases.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

5 Software Design and Analysis, Component Specification (preliminary)

This section provides the technical specification of the solution and the overall architecture diagram.

5.1 Software Modules

The main component to be developed in this project is the Gateway API. Based on Semantic Container (described in Section 4.2.1) it provides API endpoints as specified in the sections 2.3, 2.4, and 2.5 to provide a Service Catalogue, manage Organizations & Users, and handle access to Storage Providers. Moreover, it will provide a flexible Integration Helper to support entities (other services or applications) in exchanging data. The next two subsections describe the necessary input (Service Descriptions) and the expected process for this Integration Helper.

5.1.1 Service Description

For the Service Catalogue to be maintained with the API endpoints as described in section 2.3 the following structure is currently planned:

- describe the interface - specifically API endpoints - and general aspects of the entity using the Open API Specification v3 (also known as Swagger documentation)
- document the expected data model using the Semantic Overlay Architecture, i.e., describing classes, attributes, relations and associated meta-information (incl. validation and transformation)
- specify any attached governance policies using as basis the structure from the Data Privacy Vocabulary and extending it with concepts/vocabulary from Supply Chain Management ontologies.

The Service Description itself will use a flat-JSON format for easy adoption by developers.

5.1.2 Babelfish Sequence Diagram

Figure 5.1 depicts the sequence upon a data exchange between two entities.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

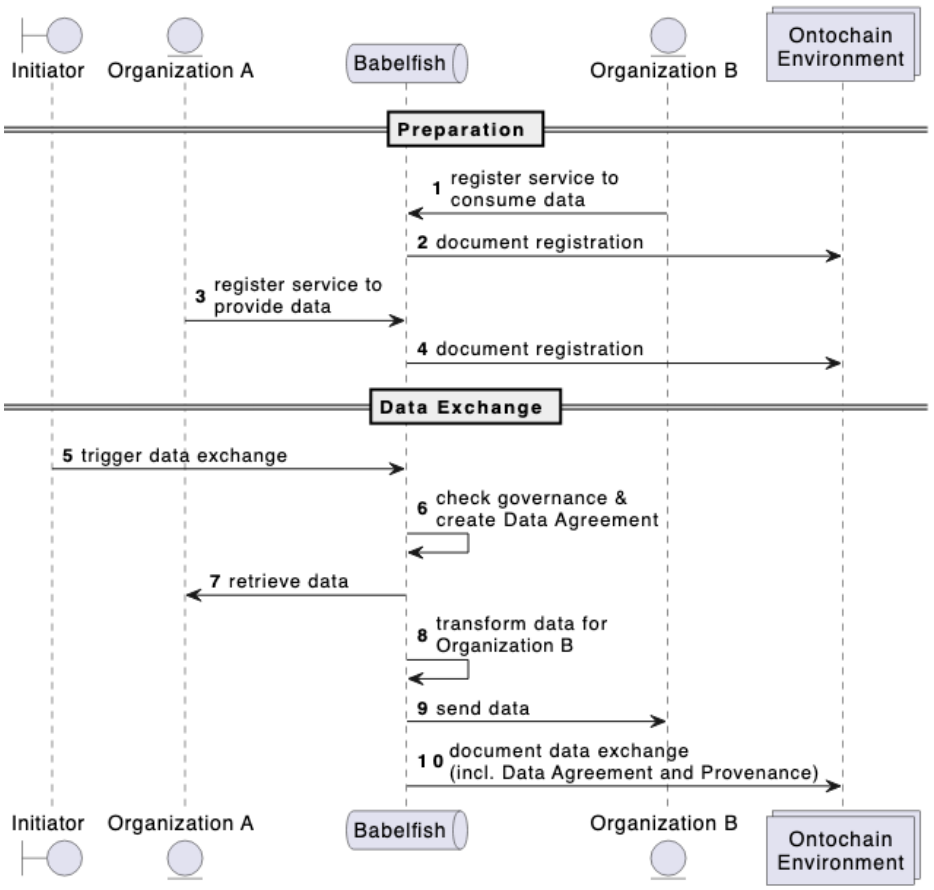

Figure 5.1: Babelfish Sequence Diagram

Step 1 & 3: an entity registers its service description for the Service Catalogue and information is stored (step 2&4) in the ONTOCHAIN environment (the configured storage provider for the Service Catalogue)

Step 5: the actual data exchange can be either triggered by one of the participating entities (Organization A or B) or be an external trigger - depicted in the sequence diagram as “Initiator”

Step 6: upon starting a data exchange the Usage Policies (if present) for the participating entities are evaluated and checked for compliance; if successful, a Data Agreement is compiled based on the Usage Policies documenting the governance aspects of this data exchange

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 957338

Step 7: data is retrieved from the data providing entity (alternatively, it could also be that a data exchange is triggered by a POST request and then the data is already available to Babelfish and doesn't need to be retrieved in this step)

Step 8: in case the Service Descriptions provide information about transforming data between the two entities, this steps performs this transformation

Step 9: the payload is forwarded to the data consuming service

Step 10: the complete data exchange is documented in the ONTOCHAIN environment (using the configured Storage Provider)

Another aspect of a data exchange are the various types of data generated in the course of the process:

- Data: the actual payload sent from data provider to data consumer
- Metadata: additional information that either specifies specific aspects of the payload or is generated in the course of the data exchange; examples for metadata include:
    - Usage Policies: can be specified in the Service Description for any data provided by an entity or can be specifically provided together with the payload)
    - Provenance: if provided together with the data it is automatically extended using Prov-O (W3C Provenance Ontology) by default
    - specific identifiers to address the data, e.g., hash for content-based addressing
- Trust: to proof detailed aspects of the data, Verifiable Credentials (VC) can be created to attest certain qualities; VCs are usually stored in the wallet of the holder and require a Verifiable Presentation (VP) when requested by a verifier
- Access: data, metadata and trust artefacts are stored in different locations and service endpoints document access to those artefacts; by default a DID Document provides those service endpoints

5.1.3 Use Case Sequence Diagram

To demonstrate the use of Service Descriptions and the process of a data exchange between two entities, a short supply chain management example will be used as depicted in Figure 5.2.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

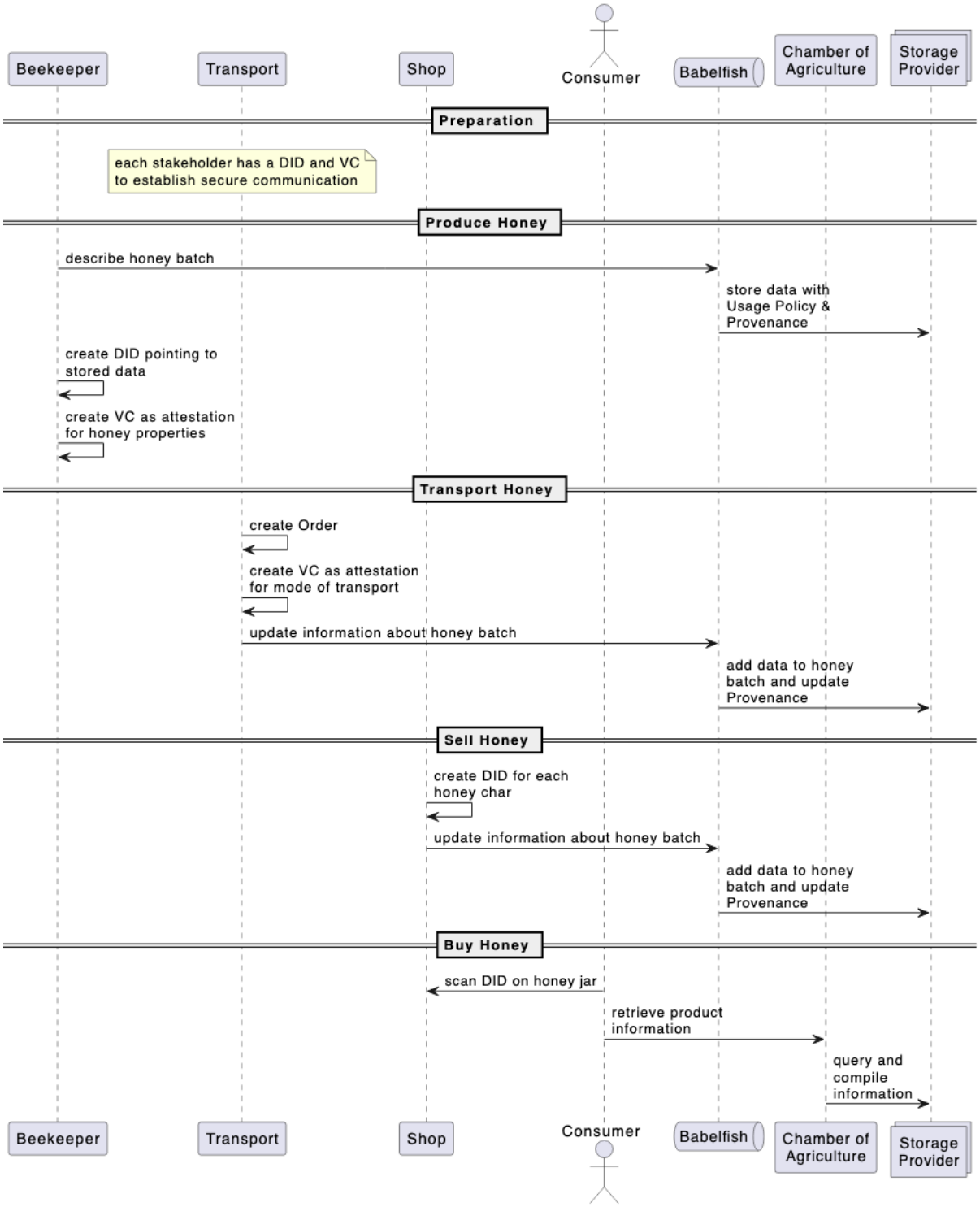

Figure 5.2: Honey Use Case Sequence Diagram

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

In this example the following four data exchanges will be facilitated by the Babelfish component:

- Produce Honey: a beekeeper produced a batch of honey and documents the initial information (e.g., date, location of beehive, quantity)
- Transport Honey: in the most simple case the honey is sold locally and brought by a bicycle courier to the shop where it is sold; source, destination, and mode of transported are documented in this step
- Sell Honey: at a shop the batch of honey (e.g., 40 jars) is then assigned with unique identifiers per jar (derived from the batch identifier) and offered to customers
- Buy Honey: a customer can then use the identifier (e.g., provided through a QR code on the honey jar) to retrieve all associated information from the product; providing such information is compiled by a separate entity (Chamber of Agriculture in the example above) that uses data stored in the ONTOCHAIN environment (i.e., the configured Storage Provider)

5.2 Architecture Diagram

The Gateway API will be a single component in the ONTOCHAIN environment exposing available resources to other services and applications as well as external applications.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

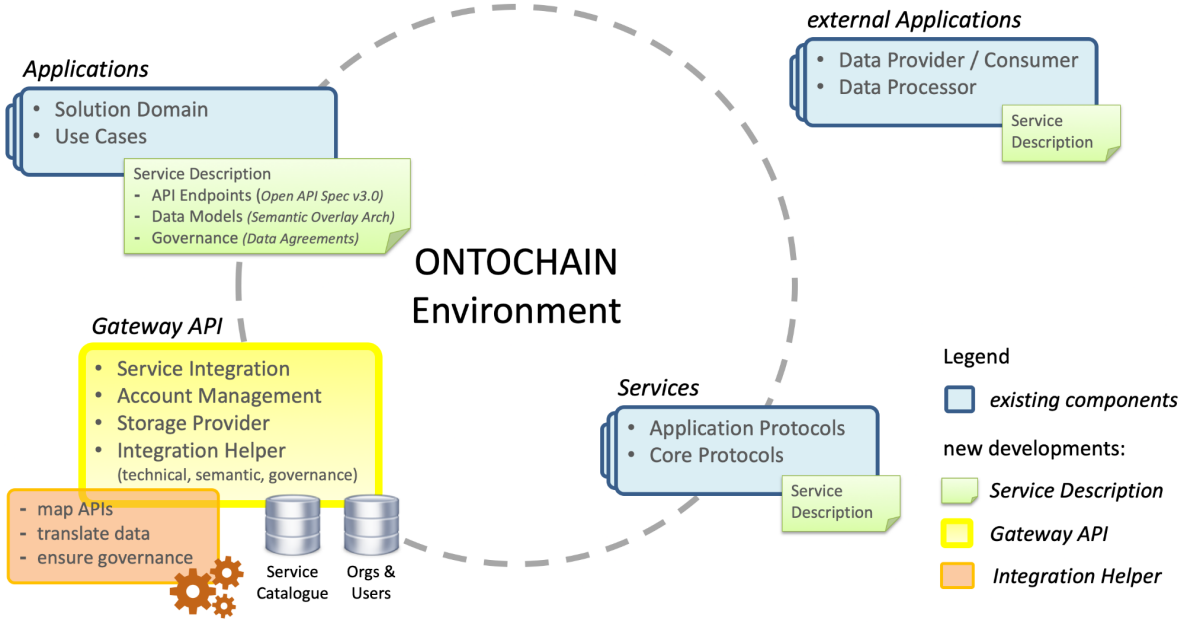

Figure 5.3: Architecture Overview

A Service Catalogue provides based on Service Descriptions a list of available services. Such descriptions are a combination of technical, semantic, and governance information for a service / an application and facilitate integration into the ONTOCHAIN environment. Based on this information an Integration Helper can be used to map APIs, translate data models between entities and ensure governance through Data Agreements. Backend storage services (either from within the ONTOCHAIN ecosystem as well as external services) are part of the Service Catalogue. Account Management is provided on an organisation- and user-level to provide authentication mechanisms.

From a technology viewpoint the Gateway API is a mediator within the ONTOCHAIN environment publishing available functions and ensuring governance. To allow for a stand-alone testing of the Gateway API core functions like storage and identities are provided through the internal use of Semantic Containers and the did:oyd method respectively. Nevertheless, other standard-conform storage and identities mechanisms can be used as well.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

6 Detailed Work Plan for Implementation and Deployment (preliminary)

The work plan for implementing Babelfish during the 10-month funded project duration takes a stepwise approach and is depicted in Figure 6.1.

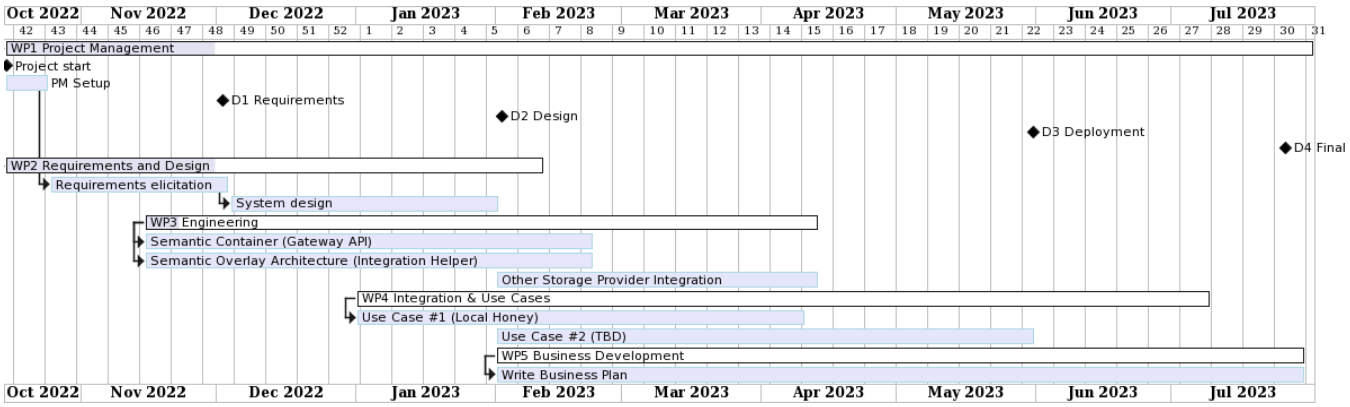

Figure 6.1: GANTT Chart

Work Package #1 Project Management (Oct 2022 - Jul 2023)

This WP spans the whole project duration and covers all administrative aspects of internal communication, organising meetings, and monitoring progress.

Work Package #2 Requirements and Design (Oct 2022 - Feb 2023)

Deliverables D1 & D2 for Requirements and Design are discussed, written, and reviewed in this WP. These documents describe the detailed features to be implemented and tested in the course of the project.

Work Package #3 Engineering (Nov 2022 - Apr 2023)

Features specified in WP2 are implemented in WP3. This WP also includes packaging, documenting, and publishing the source code to make it easier for others to find and integrate the developed software artefacts.

Work Package #4 Integration and Use Cases (Jan - Jun 2023)

Software artefacts developed in WP3 will be verified and demonstrated in various use cases. (Currently, 2-3 full-fledged use cases are planned.) This includes refining the deployment process, writing tutorials for aiding in adoption, and recording short videos for demonstration purposes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

Work Package #5 Business Development (Feb - Jul 2023)

WP5 describes the business aspects of this project to use Babelfish and relevant technologies beyond the project. The focus is here on Supply Chain Management and our collaboration with Clever Clover - Europe's #1 FMCG Accelerator.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338

7 Conclusions

This document outlined the requirements and use cases identified in the initial design phase of the Babelfish project. Based on a stakeholder analysis and the goals defined in the project proposal the main objectives were identified and non-functional as well as functional requirements were documented. Based on these requirements the design will be described in deliverable D2 Design.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957338