D2.2: Block.IS Enablers

09/2019

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 824509.

Author(s)/Organisation(s)	R. Bröchler (INTRASOFT)
Contributor(s)	INTRASOFT and TNO
Work Package	WP2
Delivery Date (DoA)	30/09/2019
Actual Delivery Date	30/09/2019
Abstract:	Deliverable D2.2 describes the enablers, i.e. and links to the software implementation, which are provided by the Block.IS partners (INTRASOFT and TNO) to the SMEs and start-ups participating in the project open calls. The SMEs and start-ups that will participate in Block.IS open calls form the primary external target audience of the deliverable who will be able to understand the enablers being offered and consider using them when designing their proposals. In parallel to the technical documentation, links are provided to repositories including the software implementation. These repositories are accessible to interested parties and managed by the project partners. The technical documentation includes descriptions of the basic concepts, the architecture, and the interactions foreseen for each enabler. Six (6) enablers are being provided, four by INTRASOFT International and two by TNO. The deliverable is the output of Task 2.2.

Document Revision History			
Date	**Version**	**Author/Contributor/ Reviewer**	**Summary of main changes**
20/06/2019	0.1	R. Bröchler	TOC and proposal for assignments
04/07/2019	0.2	R. Bröchler, C. Ipektsidis, C. Brewster	Enabler selection and discussion on enabler documentation
09/09/2019	0.3	R. Bröchler, C. Ipektsidis, C. Brewster, J. Spek	Contribution on the enabler documentation
16/09/2019	0.4	R. Bröchler	Version submitted for internal review
23/09/2019	0.5	O. V. Deventer	Review comments
27/09/2019	0.6	R. Bröchler	Version submitted to the coordinator
30/09/2019	1.0	A. Damasceno	Submission of final version

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the EC Services)	
RE	Restricted to a group specified by the consortium (including the EC Services)	
CO	Confidential, only for members of the consortium (including the EC)	

Block.IS Consortium			
Participant Number	**Participant organisation name**	**Short name**	**Country**
1	F6S NETWORK LIMITED	F6S	UK
2	INOSENS DOO NOVI SAD	INO	RS
3	INTRASOFT INTERNATIONAL SA	INTRA	LU
4	INTERNATIONAL DEVELOPMENT IRELAND LIMITED	IDI	IE
5	POSLOVNO UDRUZENJE VOJVODJANSKI IKT KLASTER	VOICT	RS
6	FEDERACION EMPRESARIAL DE AGROALIMENTACION DE LA COMUNIDAD VALENCIANA	FEDACOVA	ES
7	R-TECH GMBH	IT-Log	DE
8	EUROPEAN DIGITAL SME ALLIANCE	DSME	BE
9	FINTECHSTAGE LTD	FTS	UK
10	UAB CIVITTA	CIVITTA	LT
11	NEDERLANDSE ORGANISATIE VOOR TOEGEPAST NATUURWETENSCHAPPELIJK ONDERZOEK TNO	TNO	NL
12	SYNELIXIS LYSEIS PLIROFORIKIS AUTOMATISMOU & TILEPIKOINONION ANONIMI ETAIRIA	SYN	EL

Funding Scheme: Innovation Action (IA) ● Theme: H2020-INNOSUP-2018-01-two-stage
Start date of project: 01 May, 2019 ● Duration: 32 months

Table of contents

List of figures

List of Abbreviations and Acronyms	
AAA	Authentication, Authorization, Accounting
API	Application Programming Interface
DAPP	Decentralised Application
DDO	DID Document
DID	Distributed Identifier
ECDSA	Elliptic Curve Digital Signature Algorithm
ETH	Ethereum
GDPR	General Data Protection Regulation
HMAC	Hashed-based Message Authentication Code
IPFS	Interplanetary File System
OSS	Open Source Software
REST	Representational State Transfer
SC	Smart Contract
SHA3	Secure Hash Algorithm 3
SHACL	Shapes Constraint Language
SOTA	State Of The Art
SSIF	Self – Sovereign Identity Framework
UDDI	Universal Description, Discovery and Integration

1 Introduction and Scope

Deliverable D2.2 entitled "Block.IS Enablers" is the output of Task 2.2 (Block.IS Blockchain Enablers and supporting materials) and includes the software of the Block.IS enablers along with the supporting technical materials. The Block.IS enablers are provided by two of the project partners that participate in the Task T2.2, namely INTRASOFT International and TNO. INTRASOFT has been the editor of the deliverable, documenting the enablers that it offers, while TNO has also contributed to the deliverable documenting the enablers it offers.

The context and the procedures related to the technical support of the usage of the Block.IS enablers are described in deliverable D2.1 "Block.IS Technical and Business support service design" which is being submitted at the same time also as output of the Task 2.2 [1].

1.1 Objectives

The main objectives of this deliverable include the following:

1. To present the selected Enablers that will be offered to the applicants of the Block.IS open calls and briefly discuss the justification for their selection
2. To provide the textual documentation of the offered Enablers including the basic concepts, the high-level architecture and the interaction (sequence) diagrams describing the usage of the enablers
3. To provide the links to the (gitlab) repository including a) the open API through which the enablers can be used for the development of applications and b) the software implementation.

Given that the current deliverable is prepared in the first phase of the project, i.e. at M5, we expect to have significant interaction with the potential applicants in the open calls in the context of using the enablers. In this view, we expect that the resources and documentation related to the enablers will be evolving / enhanced during the lifetime of the project. This will be associated with the continuous evolution of the technical support material but will be also reflected on the enhancement of the online sources related to the enablers (the gitlab groups/projects that are described below).

1.2 Scope and Relation with other WPs

D2.2 is a key deliverable in the Block.IS ecosystem as it describes the enablers that are offered in the open calls of the project. The work on the provision and support of the Enablers is taking place mainly in the context of WP2, so the dependencies on other Tasks / WPs are, at this phase, mainly related to the timing of the open calls.

D2.2 is strongly related with the deliverable D2.1 "Block.IS technical and business support service design" and especially with reference to the technical support aspects (which also refer to D2.2). While the project advances, with the procedures related to the open calls, the experiences and the lessons learnt will be gathered and consolidated by WP2 partners. D2.3 "Block.IS technical and business support design - v2", as the output of Tasks 2.1, 2.2 and 2.3 will include the updates and enhancements related to the Blockchain enablers based on the feedback collected through the 1st acceleration program. D2.3 is due for submission at month M18.

1.3 Selection of Enablers

The Block.IS Blockchain enablers have been at the core of the project since its design. They are considered to offer frequently used and meaningful functionality, generic enough in order to be applicable at an extensive set of applications. The enablers leverage the Blockchain functionality and facilitate access to their added value for the application designers.

The content of the enablers has already been indicatively described during the design of the project (as discussed in the DoA). This description has been updated in the first phase of the project through dedicated discussions and teleconferences in the context of WP2, considering the state of the art in the frameworks to be used as well as the added value of the software modules. The functional areas of ID management, business process management, AAA (Authentication, Authorization, and Accounting), media storage, trusted negotiations, and semantic ledgering have been considered as crucial for application designers, forming functional blocks that are repeatedly met in SOTA applications.

The Block.IS enablers in principle enhance and enrich functionality and operations with the security-related (non-functional) characteristics of the Blockchain infrastructures. These characteristics can include (depending on the characteristics and context of the enabler):

1. Integrity of data stored and exchanged
2. Authenticity of data stored and exchanged
3. Tracing and tracking through timestamping
4. Non-repudiation of activities performed.

Our work has resulted in the provision of 6 Blockchain enablers (enhancing the initially indicated, in DoA, number of 5 enablers).

The selected enablers offered by the Block.IS consortium are the following:

1. ID management with AAA support based on uPort enabler
2. Storage enabler
3. Service registry enabler
4. Trusted negotiations enabler
5. ID and business process management based on SSIF enabler
6. Semantic ledger enabler

The enablers are self-contained modules (with their functionality fully defined through their APIs) and the interested SMEs can use one or more of them, depending on the needs of their applications. Our estimation is that the enablers, due to their generic nature, are equally applicable in the three domains pursued by the project (agrifood, logistics and finance).

The full technical material is consolidated during the provision of the technological support services and content (in the context of Block.IS WP2). Furthermore and as already mentioned, the responsible partners will be closely interacting with the potential applicants, leveraging the received feedback in order to continuously improve their offerings.

Given the technological nature of the deliverable, the reader is assumed to have some basic knowledge on a) the concept of modular application design and development, b) the technologies and tools employed (such the sequence diagrams) for the description of the enablers and c) the key technologies employed (including Blockchains). Sources informing about specific frameworks and tools used (such as uPort and SSIF) are provided for the interested reader and are further extended in the technical support services (website and forum).

1.4 Document Structure

To support readability and coherence each Section follows a uniform structure, consisting of the following sub-sections:

- Basic concepts
- High level architecture and expected interactions including sequence diagrams
- Links to the instructions related to installation and deployment and to the Open API

The Sections of the document are the following Section 2 describes the *ID management with AAA support based on uPort* enabler, Section 3 describes the *Storage* enabler, Section 4 describes the *Service registry* enabler, Section 5 describes the *Trusted negotiations* enabler, Section 6 describes the *ID and business process management based on SSIF* enabler and Section 7 describes the *Semantic ledger* enabler. The document is concluded with Section 8 that reviews the work and discusses the expected next steps.

The deliverable is closely connected with the technical material included in the gitlab projects / repositories and also with the technical support services provided to the potential applicants.

2 ID management with AAA support based on uPort Enabler

2.1 Basic Concepts

The focal area of the *Identity Management with AAA (Authentication, Authorization and Accounting) support based on uPort* enabler is the provisioning of services related to the administration of the user identity, supporting authentication and authorization. The enabler implements AAA and trust-management based on the uPort framework [2].

The uPort framework provides an *Open Identity System for the Decentralized Web*. This system allows users to register their identity on Ethereum, send and request credentials, sign transactions, and securely manage keys & data.

The *Identity Management with AAA support based on uPort* enabler protects interactions with Smart Contracts (DAPPs) under a multi-layered perspective in order to support:

1. Check signer existence
2. Check signer access groups
3. Check signer ability to interact with a Smart Contract

Based on the above, the enabler allows for a) user authentication and authorization, b) message integrity verification (validating the signatures of messages) and c) storage of data into the blockchain which can support accounting. The uPort allows for interaction with Decentralized Application (DAPPs) without the need for cryptocurrency at client side. This is expected to facilitate adoption on behalf of the application providers.

The enabler is accompanied by two cross-platform client libraries, implemented a) in Python and b) in Java. Both clients are provided with the enabler.

2.2 Architecture and Interactions

In this Section we describe the high-level architecture and the main interactions (using sequence diagrams).

2.2.1 High Level Architecture

To guarantee on-chain AAA processing, data integrity and lock-in free operation, at least at the level of identity provisioning, the enabler bases its operation on the relevant set uPort Identity SCs (Smart Contracts). The enabler is based on the automated execution of specific Smart Contacts in the Blockchain and it acts as a security gateway between the user and the protected DAPPs (Decentralized Applications) or with the Blockchain infrastructure (based on Ethereum).

The architectural concepts of MetaIdentityManager SC, the Proxy SC and the (Tx) Relay SC are employed as depicted in Figure 1 and Figure 2 (which are in principle aligned with the uPort reference architecture [2]).

The Relay checks the integrity of the received message and forwards it to the MetaIdentity Manager. This way the user (user equipment) is not necessary to host mining software and pay for transaction fees. The MetaIdentity Manager checks the validity of the identity of the sender.

The Blockchain infrastructure is based upon the Ethereum open source, public, blockchain-based distributed computing platform featuring smart contract (scripting) functionality [3]. The deployment of the Ethereum infrastructure are based upon the standard procedures (described by the 3rd party tools), while further details on the usage by the enabler are included in the enabler deployment instructions (in the respective gitlab repository).

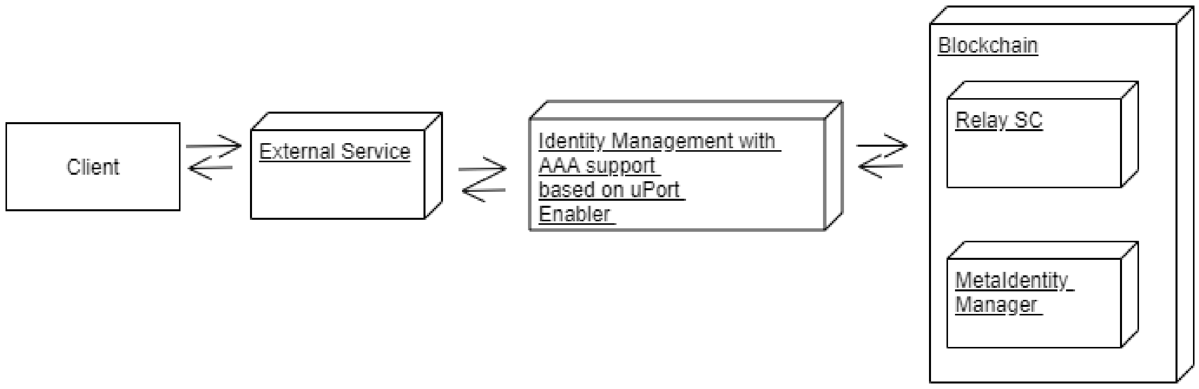


Figure 1: Identity Management with AAA support based on uPort Enabler high level architecture (non interactive scenario, client interacting with an external service).

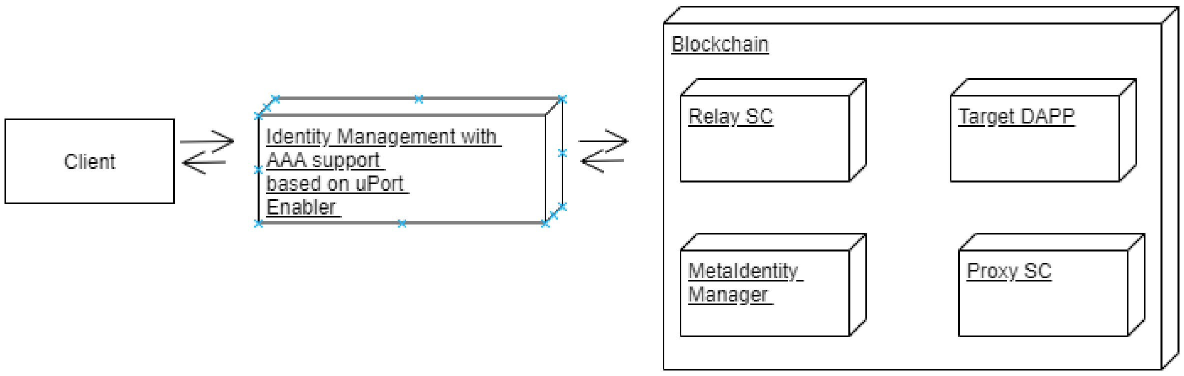


Figure 2: Identity Management with AAA support based on uPort Enabler high level architecture (interactive scenario, client interacting with a DAPP).

In principle we foresee two cases: a) the *non-interactive* where the client interacts with an external service which uses the enabler for authentication and b) the *interactive* where the client interacts with a Smart Contract (Target DAPP), through the enabler. In the case of (b) the Target DAPP is also depicted as well as the Proxy Smart Contract which forwards the message to the target DAPP.

We foresee two operation modes: 1) BC-based AAA services, interacting with the ETH BC for explicitly authenticating and authorizing every request towards the set of protected modules and 2) Cached AAA services, necessitating the invocation of BC-based AAA services at least once and after that periodically invoking the BC-based services to update the AAA-cache. The 2nd mode of operation is oriented towards real time requirements. Specifically, due to the execution rate of the Smart Contracts residing in the BC, the SCs are automatically executed whenever a new block is generated. The inherent nature of the ETH BC (by design) introduces a certain delay (indicatively 10-20 seconds) between the generation of subsequent blocks. The AAA results are provided by the BC in the form of batches (after related delay). Considering that such a delay may not be acceptable by certain types of applications we provide the cached mode of operation.

The selection of the operation mode is performed during the deployment and configuration of the enabler.

2.2.2 Interactions

The clients of the *Identity Management with AAA support based on uPort* enabler are able to interact with a) external services in a non-interactive scenario and b) with the protected DAPPs in the interactive scenario by issuing sets of digitally signed requests that generate interactions with the blockchain. In the following we discuss the interactions, through sequence diagrams, for both scenarios.

Non - Interactive scenario

The typical interactions are depicted in Figure 3:

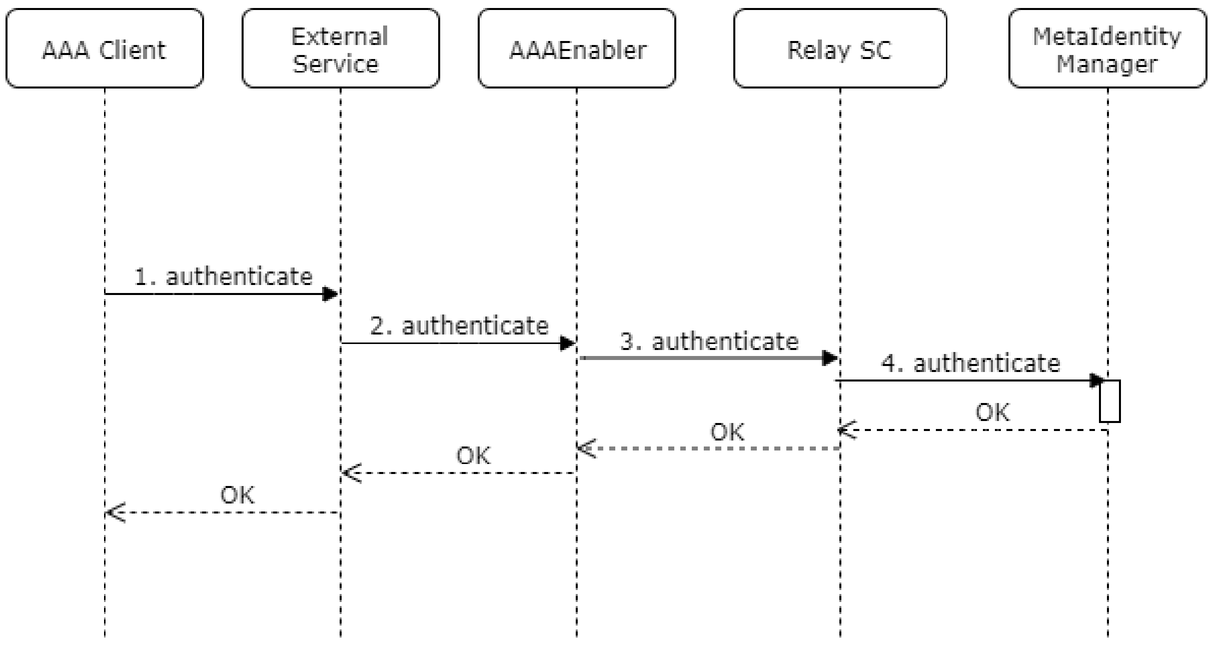


Figure 3: Sequence diagram for non-interactive authentication

The basic steps included in the sequence are the following:

1. The client contacts an external service (i.e. not belonging to the Blockchain infrastructure) and needs to authenticate himself in order to use it
2. The external service invokes the *Identity Management with AAA support based on uPort* enabler in order to verify the identity of the sender. The enabler should verify that the public address of the user that signed the request (using the Hashed-Based Message Authentication Code – HMAC) is known to the blockchain infrastructure.
3. The enabler contacts the Relay Smart Contract
4. The MetaIdentity Manager is then contacted and performs the verification.

The responses follow the reverse route as depicted in the figure. In this case of non-interactive authentication, no transaction hash is generated as the interaction with the blockchain is passive (only calling the Relay SC and the MetaIdentity manager).

Interactive Scenario

In the case of interactive scenario, the client is interacting with a Smart Contract (e.g. one of the DAPPs) and the Enabler should authenticate and authorize that request. In this case a transaction hash characterizing the interaction of the client with the blockchain is generated.

The typical sequence consists of the following steps and it is depicted in Figure 4.

1. The client contacts the Identity Management with AAA support based on uPort Enabler in order to interact with the target DAPP
2. The enabler gives the control of the AAA process to the Relay Smart Contract. The input to the Relay Server that gets relayed should be a signed message containing:
    a. Data to be relayed including a) the address target ETH distributed application (DAPP) that should be activated if all AAA checks are validated, b) the target function of the DAPP to be executed, c) the data parameters to be passed into the target DAPP.
    b. Signature attributes of the message in compliance with the Elliptic Curve Digital Signature Algorithm (ECDSA) implementation of ETH.
    c. Owner address (public key) in the ETH BC.
    d. Destination address in the ETH BC (the address of the MetaIdentityManager Smart Contract).
3. The Relay SC validates that the provided signature elements are compatible with the hashed data included in the message, and forwards the message to the MetaIdentityManager SC.
4. In turn, the MetaIdentityManager SC checks whether the owner address is valid in the context of this MetaIdentityManager by means of calling the relevant Proxy SC, essentially performing authentication. Then the authorization part takes place. If the request is both authenticated and authorized, the request gets be relayed to the appropriate Proxy SC.
5. The Proxy SC forwards the message to the target DAPP, as initially specified by the AAA Enabler client initiating the whole transactions forwarding chain. Depending on the target DAPP, *accounting* may be achieved if relevant accounting functionalities are included in this DAPP (depending on the pursued application scenario).

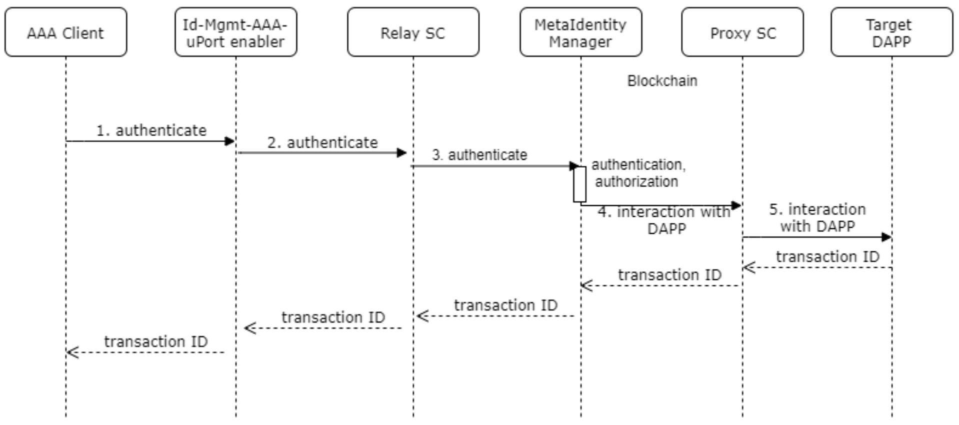


Figure 4: Sequence diagram for interactive usage of AAA (authentication)

The Relay SC, the MetaIdentityManager SC and the Proxy SC are based on the OSS uPort set of contracts. The MetaIdentityManager SC is able to register new identities and owners, migrate Proxy SCs to other MetaIdentityManager SCs and handle the identity recovery operations.

At all stages, the involved SCs can emit properly defined (Ethereum-related) events so that errors particularly related to unauthenticated or unauthorized access are caught by an event server (configured by the application provider) and get published to relevant communication channels to be consumed by devices.

As discussed, in the context of the Relay Server, successful authentication and authorization requests may be cached at a local context for a limited timeframe so that authentication and authorization for time-critical applications can also be supported. In such cases, the Relay server will simply skip the AAA process, directly invoking other DAPPS.

2.3 Installation and Administration

The software and instructions for its installation / deployment and administration for the *ID Management with AAA support based on uPort* enabler are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/id-management-aaa-uport-enabler

In parallel with the documentation related to the deployment of the enabler, instructions are provided for the accompanying Ethereum platform in order to form a pilot environment.

The group is administered by INTRASOFT International.

2.4 Open API

The technical documentation and the software of the Identity Management with AAA support based on uPort enabler are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/id-management-aaa-uport-enabler

The group is administered by INTRASOFT International.

3 Storage Enabler

This Section describes the *Storage* enabler.

3.1 Basic Concepts

The *Storage* enabler allows the end user (i.e. the user of the proposed application) to encrypt and store files (consisting of e.g. text, photos and videos) in a distributed file system (and specifically in the context of the Storage enabler in the *Interplanetary File System, IPFS*) which collaborates with the Blockchain infrastructure, registering the relevant activities [4]. The information exchanges, during the file exchanges, are permanent, immutable and traceable, while the files are kept encrypted in the file system. As the files are stored in the file system, their size is not limited (by the typical limitations of the Blockchain infrastructure).

The Storage enabler is combined with a smart phone application (*Storage App*), which is used by the end user and collects the messages containing text, images and video (collaborating with the Android OS of the mobile phone). The Storage App digitally signs each message before transmission in order to allow for identity checking and integrity validation. The enabler is typically configured so that these messages are deleted on the user mobile device after being sent them to the enabler. *The storage App is also provided with the enabler.*

For the usage of the Storage enabler, it is assumed that user registration has already taken place and the user is already authenticated (using for example the *Identity Management with AAA (Authentication, Authorization and Accounting) support based on uPort* enabler).

3.2 Architecture and Interactions

In this Section we describe the high-level architecture and the main interactions (using sequence diagrams) of the Storage enabler.

3.2.1 High level Architecture

The *Storage* enabler allows the end user (i.e. the user of the proposed application) to encrypt and store files (consisting of e.g. text, photos and videos) in a distributed file system. In parallel the enabler collaborates with the Blockchain infrastructure for registering the file storage transactions and the IPFS distributed file system for storing files (text, photographs and videos). The enabler offers a secure Application Programming Interface (API) to facilitate interactions at application level (as depicted in Figure 5).

The Storage App (application) is responsible for getting the file (through the mobile phone) and providing it to the Storage enabler, which then communicates with the distributed file system (IPFS) and the Blockchain.

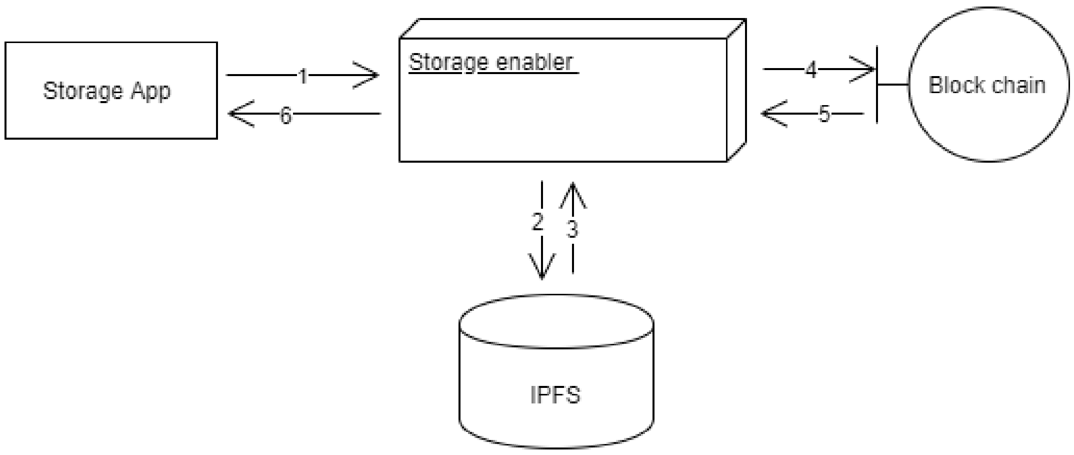
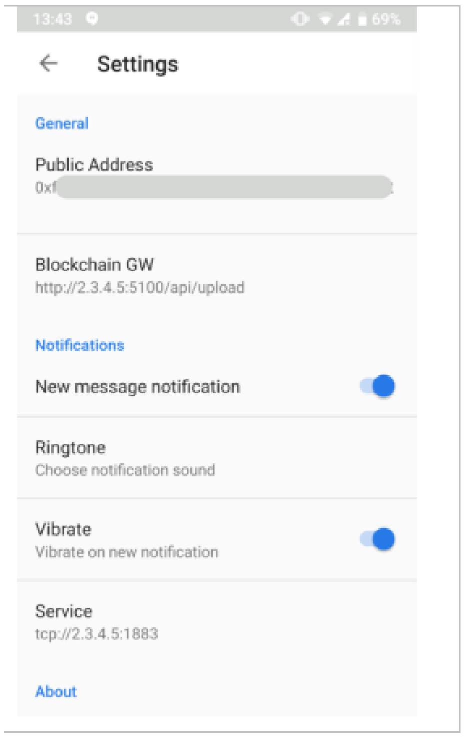


Figure 5: Storage enabler high level architecture

The Blockchain infrastructure is based upon the Ethereum open source, public, blockchain-based distributed computing platform featuring smart contract (scripting) functionality. Its deployment is based upon the standard procedures (described by the 3rd party tools), while further details on the usage by the enabler are included in the enabler deployment instructions.

The App features a similar layout to popular messaging applications to facilitate adoption and usage on behalf of the users. A view of the Android App is presented in Figure 6.



13:43 69%

Settings

General

Public Address
0xf[redacted]

Blockchain GW
http://2.3.4.5:5100/api/upload

Notifications

New message notification ●

Ringtone
Choose notification sound

Vibrate ●
Vibrate on new notification

Service
tcp://2.3.4.5:1883

About

Figure 6: Storage enabler Application view (settings)

The Storage Enabler App is expected to be allowed access by the user to specific smart phone features and specifically a) the camera, to allow the application shoot photographs and videos, b) the microphone if the user wishes to include sound in the recording, c) the mobile phone storage for temporarily storing the file prior to sending them and d) the location info for event localization.

3.2.2 Interactions

The entities involved include the (human) user, the Application, the Storage Enabler, the IPFS, the ID management enabler (depicted as AAA enabler) and the Blockchain infrastructure. The *ID management with AAA support based on uPort* enabler can be used for example.

When the user installs the Storage Enabler App, the App automatically generates a key pair (public and private one) that is used for signing the requests toward the Storage Enabler API Interface. To protect this information from unauthorized use, upon first application opening, the user is asked to enter a password; this password is used by the application for locking the user blockchain wallet. Whenever a user wants to send a file (text, photograph or video) the application prompts the user for typing their password so that their private key store gets unlocked and the content is signed before sending.

In the typical operational scenario of a user uploading a file with the Blockchain-based mechanisms for providing the authenticity and integrity of the file, we foresee that the following interactions take place, as depicted in Figure 7.

1. The user creates the content to be uploaded using the smart phone application (App). The Application interacts with the smartphone and the file becomes available (e.g. a video or photo is taken)
2. The user indicates to the smart phone application (App) to upload the file. The application generates metadata including the BC address of the sender, the timestamp and the location (upon confirmation from the user).
3. The user is asked to authenticate using the AAA enabler (which uses the Blockchain infrastructure).
4. The Application calculates the MD5 hash of the file to be uploaded and then the SHA3 256 hex string of the MD5 hash. It unlocks the Ethereum wallet (key file) of the user, asking for his password and signs the SHA3 (Secure Hash Algorithm 3) hash with the private key of the unlocked user wallet. It then locks the Ethereum wallet of the user and uploads the file to the Storage enabler.
5. The Storage enabler uploads the file to the IPFS file system. The information is stored on the distributed file system (IPFS) in encrypted chunks. The IPFS hash is returned to the enabler. The IPFS hash can be used to retrieve the file from the IPFS.
6. The hash is sent by the Storage enabler in order to be registered in the Blockchain infrastructure (and specifically the relevant Ethereum Smart Contract), and a transaction id is generated. The transaction is merged into a new block. The transaction id can validate the address of the sender (origin of the file) and its integrity. The file is deleted from the smartphone of the user.

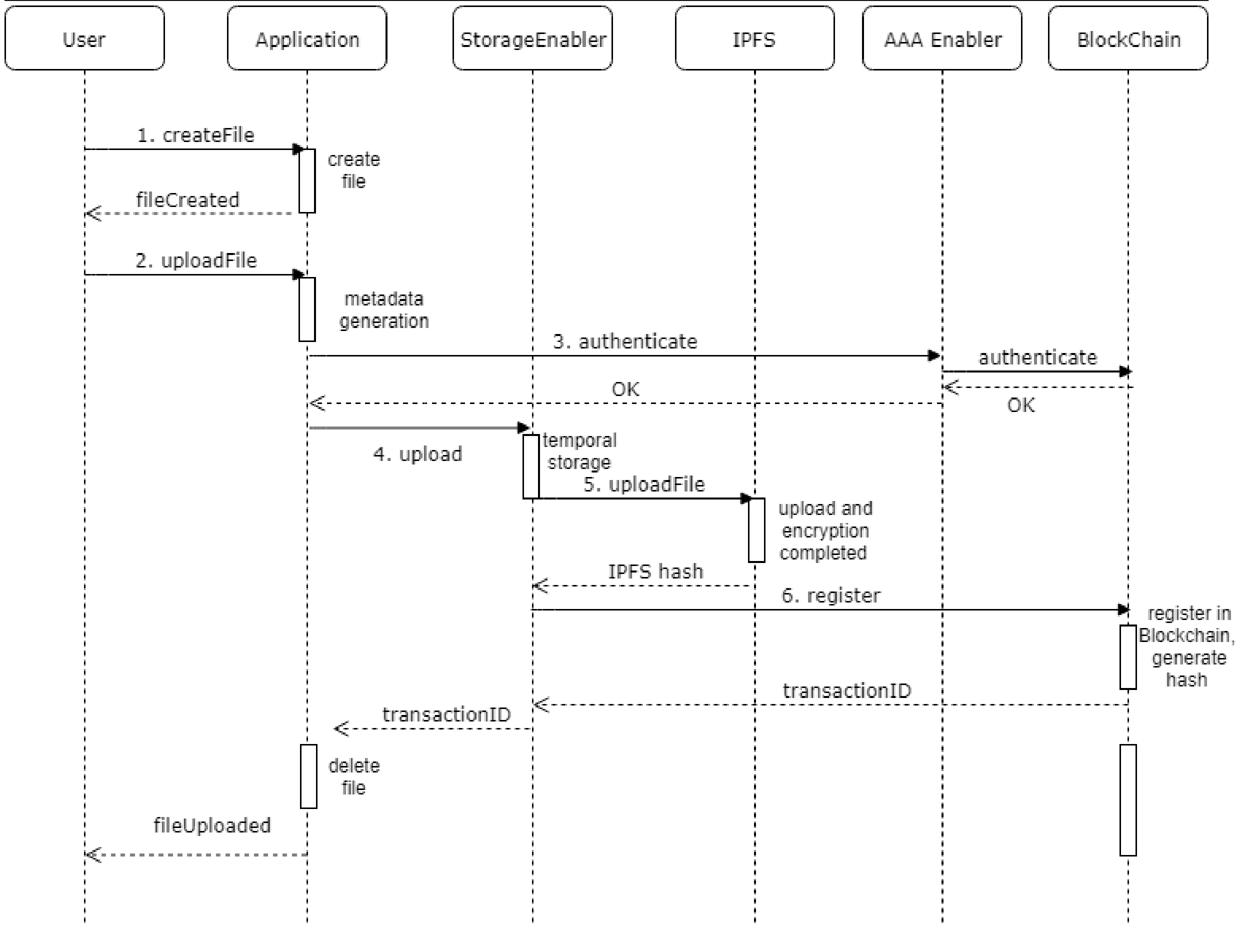


Figure 7: Storage enabler interactions

The integrity and authenticity of the file can be verified at any time therough the IPFS hash, by the user (who uploaded the file or other interested parties, e.g. the users of the application using the *Storage* enabler), using the functionality of the Ethereum blockchain and the IPFS infrastructure.

3.3 Installation and Administration

The software and instructions for its installation / deployment and administration are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/storage-enabler

The package consists of the enabler and the smartphone App (application).

The group is administered by INTRASOFT International.

3.4 Open API

The technical documentation and the software of the Storage enabler are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/storage-enabler.

The group is administered by INTRASOFT International.

4 Service Registry Enabler

This Section describes the Service Registry enabler.

4.1 Basic Concepts

The Service Registry enabler implements an entity, service and smart contract registry directory where the entities participating in the Block.IS ecosystem can register (and advertise) themselves as well as the services and smart contracts they offer based on keywords (tags).

This way finding services and smart contracts in Block.IS is facilitated and streamlined. The services registered in the Service Registry enabler are not necessarily using Blockchains.

The Service Registry Enabler manages information and metadata related to:

1. Block.IS participating entities
2. Services and smart contracts offered by the participating entities and used in the Block.IS ecosystem

For the registered services, the enabler includes the service endpoint URLs, while for the smart contracts, the Decentralized Applications (DAPPs) addresses are included.

For compatibility purposes and to facilitate adoption and usage by potential users, the concepts and approach of UDDI (Universal Description, Discovery and Integration) and similar approaches, such as Apache ZooKeeper as a centralized service for maintaining configuration information and naming are considered and adapted [5, 6].

4.2 Architecture and Interactions

In this Section we describe the high-level architecture and the main interactions involved in the usage of the enabler.

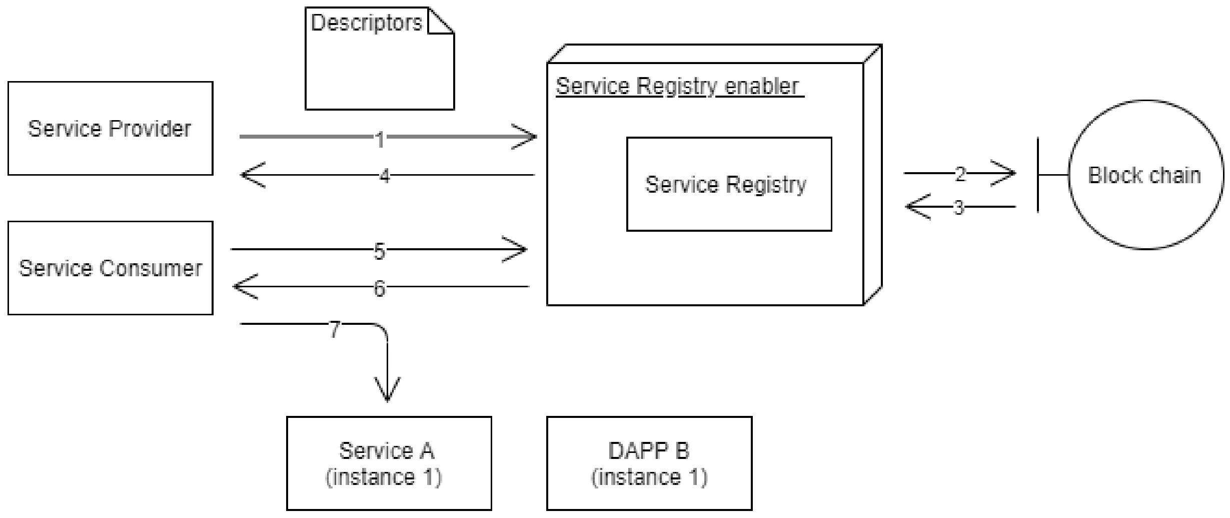
4.2.1 High Level Architecture

The Service Registry enabler hosts published information on the entities participating in the Block.IS ecosystem and the services or smart contracts (definitions as well as their deployed implementations / available instances) they provide within the Block.IS ecosystem. In this view, the Service Registry enabler can act as an internal Block.IS project entity, service and smart contract registry.

We foresee two key roles i) the Service Provider, who is the provider of the web services (or DAPPs) and ii) the Service Consumer, who is searching for services (or DAPPs) to consume. Both must be registered in the Registry. In terms of the mode of registration (self-registration or registration by 3rd party), the participating entities can register themselves and their services or smart contracts (self - registration).

The entities, the services and the bindings can be registered in (and deleted from) the Service Registry as well as searched for (through the Registry). For reasons of compatibility with similar (well established solutions) and in order to increase potential adoption of the enabler, the interface and the data model definitions are based upon that of UDDI.

The main enhancement provided by the Service Registry enabler, in comparison with existing registry solutions, is related to the Blockchain support, I.e. the registration in the (Ethereum-based) Blockchain infrastructure of the full set of actions that are performed through the enabler in order to



support authenticity, integrity and immutabilty.

Figure 8: Service registry enabler high level architecture

As depicted in Figure 8, the Service Registry enabler interacts with (1-4) the Block.IS entities (service and smart contract providers and consumers) in order to register themselves, the services and the smart contracts they offer and (5, 6) with the entities interested in consuming services offered in the Block.IS. These entities are then directed towards specific service and/or smart contract instances and interact directly with them (7).

The enabler retrieves and registers the interactions with the Blockchain infrastructure, so that integrity, immutability and authenticity are guaranteed (and verifiable using the transaction IDs produced by the Blockchain by interested parties).

4.2.2 Interactions

In the following two types of interactions are described (a) the registration and deletion and (b) the inquiry. The service (or Smart Contract) provided by the Service Provider is represented as a service structure (as described in UDDI definitions) which includes information on the type and the usage of the service. The binding templates model the actual implementation of an offered service (or Smart Contract) and includes information on its access and usage.

Registration interface

According to Figure 9, the following methods are supported:

1. Register entity: The participating entity (Service Provider or Service Consumer) is registered in the Service Registry. The entity structure contains contact information about the entity and a list of services provided. Upon registration of the entity the enabler makes a related registration in the Blockchain (and retrieves the transaction ID).
2. Register service: The participating registry can register a service. The service model includes information on the service, categorical data and the list of the technical descriptions for the web services provided (binding templates).

3. Register bindings: The binding template includes the technical descriptions of the web services provided. Each binding template describes an instance of a Web service offered at a network address, typically given in the form of a URL.

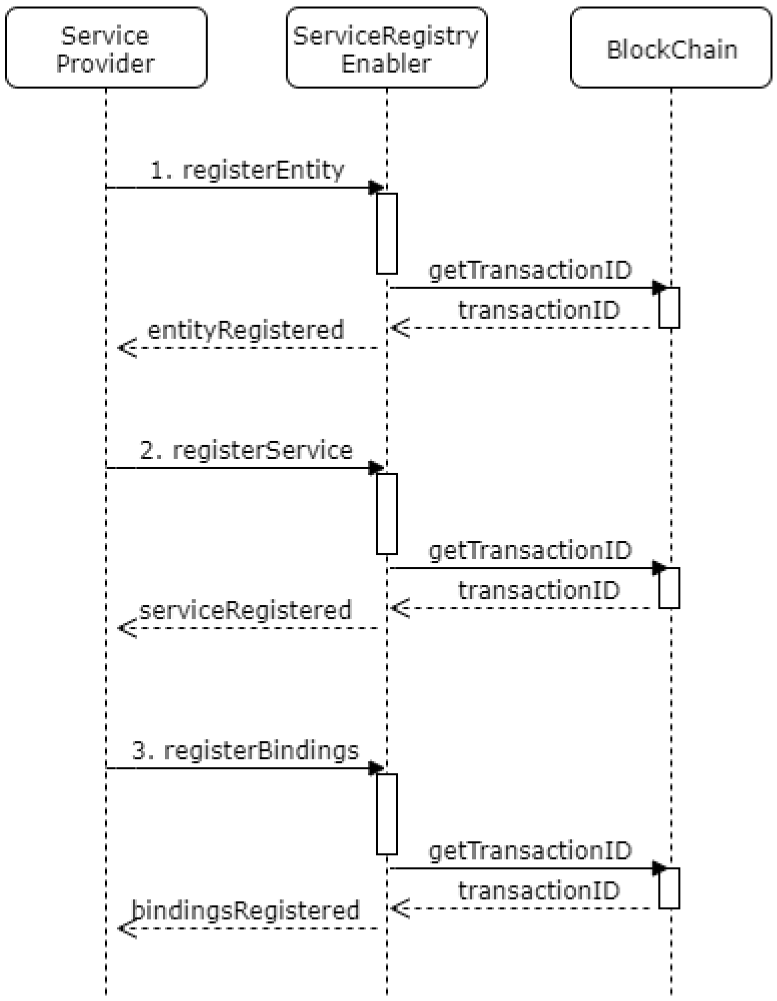


Figure 9: Sequence diagram depicting usage of Service Registry enabler (registration)

As depicted in Figure 10, the provider can delete the registrations performed.

1. Delete bindings: This method allows the deletion from the Registry of one or multiple (already registered) bindings
2. Delete service: This method allows the deletion from the Registry of an already registered service
3. Delete entity: This method allows the deletion from the Registry of an already registered entity.

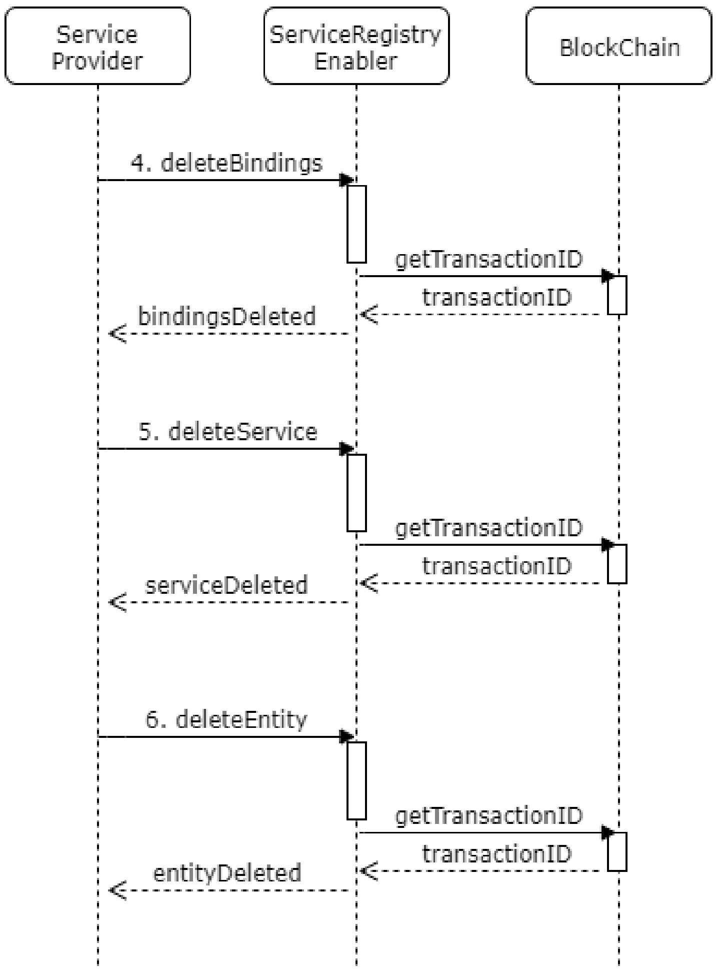


Figure 10: Sequence diagram depicting usage of Service Registry enabler (deletion)

Inquiry Interface

The inquiry interface is used when an entity is searching for services to consume, using the available metadata. Upon finding candidate services, the application retrieves the formal description of their bindings and then contacts and possibly consumes them (with the contact and consumption of the services being outside the scope of the enabler).

The interface is presented in Figure 11, according to the following:

1. Find entity: The service consumer searches for entities, according to criteria. The enabler (performing search in the Registry) returns a set of entities.
2. Get entity details: The service consumer asks for details of a specific entity; the details are provided by the enabler.
3. Find service: The service consumer searches for services, according to criteria. The enabler (performing search in the Registry) returns a set of services.
4. Get service details: The service consumer asks for details of a specific service; the details are provided by the enabler.
5. Find binding: The service consumer searches for bindings, according to criteria. The enabler (performing search in the Registry) returns a set of bindings.
6. Get binding details: The service consumer asks for details of a specific binding; the details are provided by the enabler.

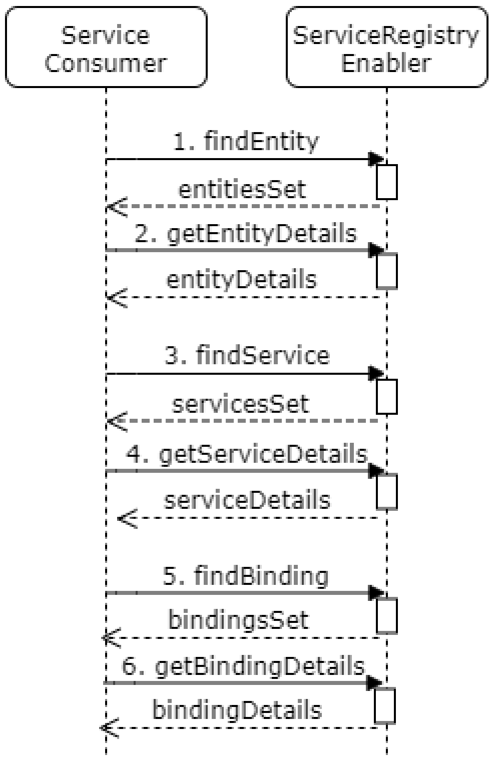


Figure 11: Sequence diagram depicting usage of Service Registry enabler (inquiry)

The role of the Service registry enabler concludes with the provision of the registered information on the entities, services, smart contracts and respective bindings. The service consumer is consequently interacting with them (outside the context of the Service Registry enabler).

4.3 Installation and Administration

The software and instructions for its installation / deployment and administration are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/service-registry-enabler

The group is administered by INTRASOFT International.

4.4 Open API

The technical documentation and the software of the Storage enabler are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/service-registry-enabler

The group is administered by INTRASOFT International.

5 Trusted Negotiations Enabler

5.1 Basic Concepts

The *Trusted Negotiations* enabler supports and enhances (in terms of integrity, authenticity and non-repudiation guarantees) the *critical (key) operations* taking place during the interactions with a Marketplace. With the concept of "critical operations" we refer to operations related to registration of information (during the operation of and the interactions taking place in a Marketplace). These operations include the making of an "offer", a "bid", as well as the establishment of bilateral agreements between parties that are already registered in the Block.IS-enabled platform.

The Trusted Negotiations enabler offers:

1. Integrity: Ensuring that the offer, the bid and the agreement have the exact same content as the original.
1. Authenticity: Proving who created or changed the offer, the bid or the agreement.
2. Tracing and tracking through timestamping
3. Non-repudiation: Not allowing a user to deny that he created or changed an offer, a bid and an agreement (as long as each interaction is stored in the blockchain, using the enabler).

The Trusted Negotiations enabler provides the following:

1. Set of Distributed Applications (DAPPs) supporting the critical marketplace processes (initialization of marketplace, creation of an offer, a bid and an agreement).
2. Set of DAPPs supporting the secure storage of the relevant agreement terms.
3. Activation of external services when a DAPP event (e.g. agreement breakage or revocation) takes place.

Considering the rich set of configuration options for a Marketplace (e.g. related to the types of offers and bids, the mechanisms supporting the decisions made and other), it has been decided not to include a specific Marketplace in the enabler. The enabler offers the methods that allow for collaboration with external marketplaces. In this view, the potential adopters of this specific enabler may be expected to perform adaptations in order to smoothly use the enabler with the selected marketplace. The Trusted Negotiations has pursued to leverage concepts and approaches of currently used marketplaces (such as [7]) in order to facilitate adoption.

We assume that the participating entities are authenticated and authorized in the Blockchain infrastructure (for example employing the *ID manager with AAA support based on uPort* enabler).

5.2 Architecture and Interactions

In this Section we describe the high-level architecture and the main interactions (using sequence diagrams).

5.2.1 High Level Architecture

The *Trusted Negotiations* enabler enhances critical marketplace procedures and mechanisms with the data integrity and authenticity based on Blockchain, referring to the marketplace-related artefacts, including the offers, the bids, and the agreements (transactions) taking place through the marketplace mechanisms. It also offers tracking and tracing as the exchange through the Trusted Negotiations are digitally signed and time-stamped, history tracing of the exchanged information and of the transactions can take place. Data can be securely stored (using the Storage enabler) and verified through the Blockchain mechanism. Furthermore, the enabler can create events when specific market-place operations take place (to be offered to external interested applications and stakeholders, e.g. through pub/sub infrastructures).

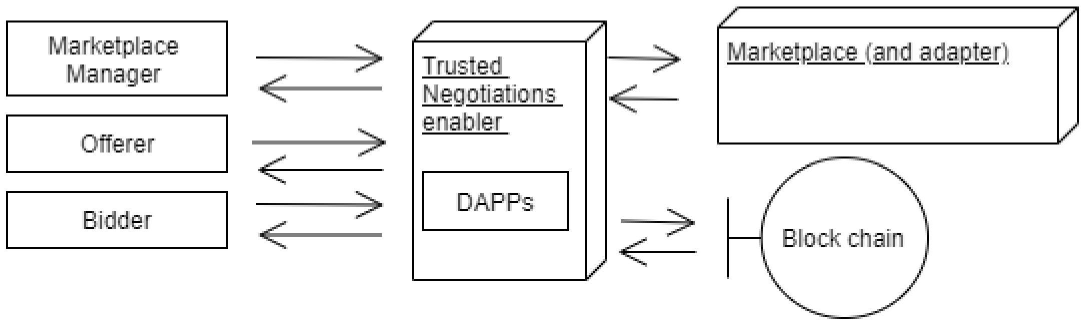


Figure 12: Trusted Negotiations enabler high level architecture

As depicted in Figure 12: Trusted Negotiations enabler high level architecture, the Trusted Negotiations enablers collaborates with (a) the Marketplace manager who is responsible to initiate and terminate the Marketplace, (b) the Offeror(s) and (c) the Bidder(s). In this view the enabler interacts with these actors and with the Marketplace through the API it offers (and upon the triggers offered by the actors). The enabler includes the DAPPs for secure storage of information being responsible for registering the hash of this information with the Blockchain infrastructure. The Blockchain infrastructure (as depicted in the figure) is based upon the Ethereum platform.

5.2.2 Interactions

The interactions performed by the enabler are presented in Figure 13.

1. Initiate marketplace: The Marketplace Manager initiates the Marketplace, including the necessary parameters for its operation. The type of the marketplace, auctioning or booking type is described as well as the mechanism for accepting bids. The initiation and the parameters involved are stored in the Trusted Negotiations enabler and the secure hash is registered (by the enabler) to the Blockchain infrastructure.
2. Create Offer: The offeror creates an offer (listing). The listing includes public (title, description, geolocation, price, createdAt, duration, state) and private metadata. The information on the Offer (listing) is securely stored by the enabler and the hash is registered in the Blockchain.
3. Edit offer: The offeror edits an existing offer. During editing, the offeror can change the value of the related metadata and/or cancel the offer. The edits performed upon the offer are securely stored by the enabler and the hash is registered in the Blockchain.
4. Make bid: The Bidder makes a bid related to a specific offer. The Marketplace verifies that the conditions are met. The bid is securely stored by the enabler DAPP and the hash is registered in the Blockchain.
5. Close offer: Depending on the type of the Marketplace, the offer can be closed by the Marketplace Manager. In the case of peer to peer marketplace it can be closed by the offeror or automatically through specific conditions.

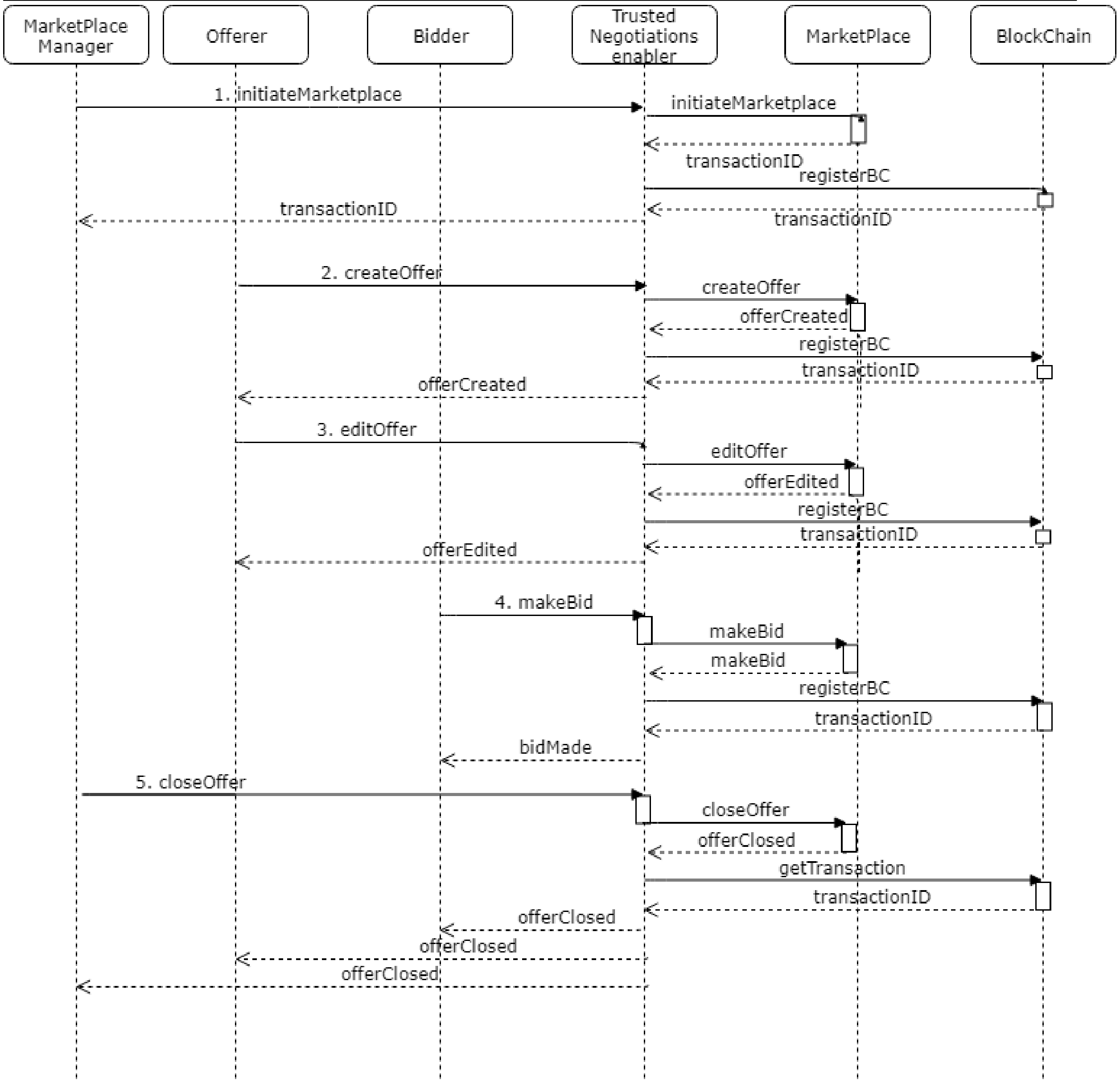


Figure 13: Sequence diagram depicting the usage of Trusted Negotiation enabler

The set of operations supported by the *Trusted Negotiations* enabler can be configurable depending on the characteristics and needs of the application. For example, in some cases all involved interactions may need to be registered in the Blockchain infrastructure (focusing on the security aspects), while in other cases the requirements may be more relaxed in terms of registering fewer interactions in the Blockchain (e.g. opening and closing of offeres).

5.3 Installation and Administration

The software and instructions for its installation / deployment and administration are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/trusted-negotiations-enabler

The group is administered by INTRASOFT International.

5.4 Open API

The technical documentation and the software of the Storage enabler are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/trusted-negotiations-enabler

The group is administered by INTRASOFT International.

6 Identity and business process management based on SSIF Enabler

6.1 Basic Concepts

An identity consists of different parts of information, i.e. attributes about a person or organization), and these parts can be proven by other parties. Self-sovereign identity promises to empower European citizens with new means to manage privacy, to eliminate logins, and to enjoy much faster and safer electronic transactions via the internet as well as in real life. SSI promises to empower European organisations to speed up, secure and automate transactions with customers, suppliers and partners, resulting in tens of billions of euros savings annually on administrative costs in Europe. SSI promises to drive a new business ecosystem with thousands of new jobs, new job categories and new business opportunities for existing and new European companies [8].

SSI helps speed up business transactions. For example, verifiable credentials can be used to fill in digital forms, thus reducing mistakes and saving time. The receiver of the form can learn what has been filled in automatically via which credential, so he knows whether it is trustworthy and that no mistakes were made while filling in the form. SSI can also be used to help organisations with the GDPR privacy requirements. Since information about customers or clients can be retrieved from verifiable credentials, an organisation does not have to save customer details herself. Also, the selective-disclosure features of SSI enable the organisation to request only the information that they strictly need for their business processes and business decisions.

The relevance of SSI in the context of the Block.IS project is that SSI allows answering the question "which party has signed this (financial, logistic or agrifood) transaction" without revealing more information about that party than strictly needed. For example, a fruit grower that needs to prove that he is located in Spain can get proof from its municipality, revealing just the address of the orchard and no personal information about the owner.

6.2 Architecture and Interactions

In this Section we describe the high-level architecture and the main interactions (using sequence diagrams).

6.2.1 High Level Architecture

Self-sovereign identity offers, as depicted in Figure 14:

1. Pairwise trusted communication with any other citizen or organization;
2. Mutual verifiable credentials to speed up and reduce risk in transactions.

The first point is achieved with Distributed Identifiers (DID) and its associated resolution technology to obtain public keys and service endpoints in a DID Document (DDO).

The second point works via verifiable credentials, a 'proof' can be created from a subset of one or more credentials. These credentials are cryptographically signed by the issuer, privacy preserving and machine verifiable.

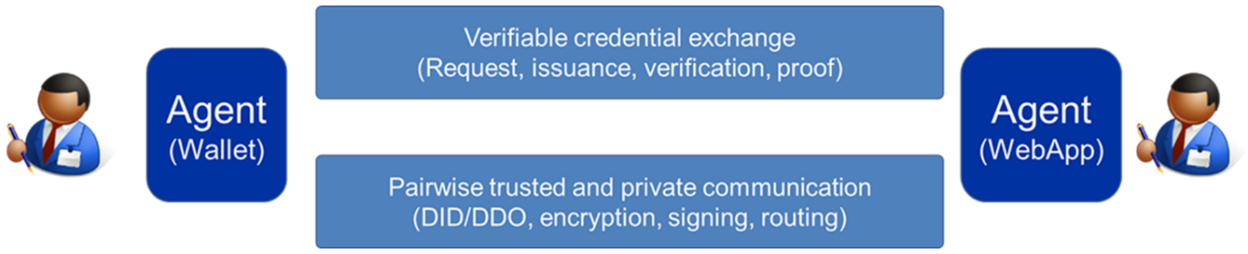

Verifiable credential exchange
(Request, issuance, verification, proof)
Agent
(Wallet)
Pairwise trusted and private communication
(DID/DDO, encryption, signing, routing)
Agent
(WebApp)

Figure 14: SSI - verifiable credentials and distributed identifiers

In order to describe the high-level architecture of SSI, we consider the involved stakeholders, namely the holder, the issuer and the verifier (as depicted in Figure 15).

The holder can collect and control verifiable claims. They are issued by an issuer and can be requested by the verifier. The holder of the claim can store them in for example an application on his phone. The verifier can check on a public ledger whether the received claim has not been revoked but that is not necessary.

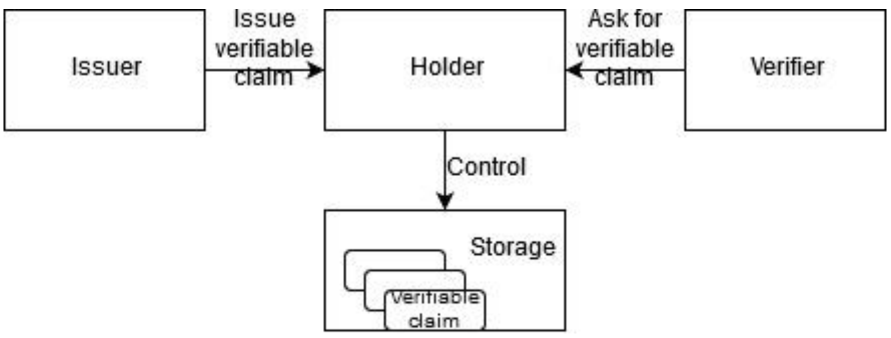


Figure 15: SSI high-level architecture

While SSI-based 'solutions' are emerging worlwide, the majority has a local scope, i.e. they solve a problem in a specific domain, they do not scale (at every level they need to, e.g. at the technical, process, information and business levels), and scarcely interoperate. Addressing these issues is a top priority for the eSSIF-Lab project, an EU H2020 project that TNO is working on.

6.2.2 Interactions

In a communication session, where party A has to prove something to party B, the following interactions will occur.

1. The holder, party A, does a transaction request to a verifier, party B.
2. The verifier gives a list of statements and attestation specs to the holder, who then has to collect all statements that were asked. The missing statements can be collected by a credential issuer.
3. The holder then sends all statements to the verifier, who can check if any credentials are revoked on a public ledger.
4. He constructs a valid argument and sends an ok/not ok message back to the holder.

The sequence of these interactions is depicted in Figure 16.

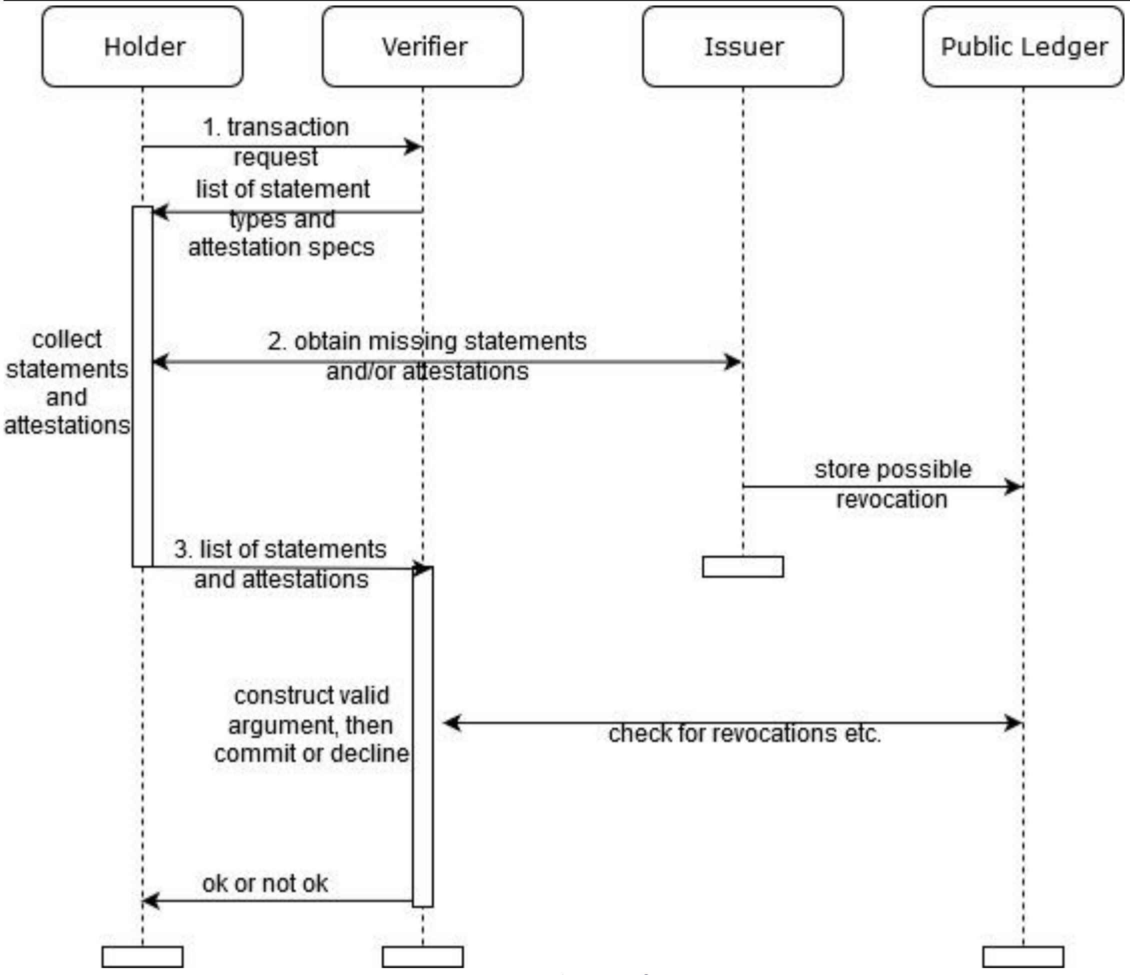

Holder
Verifier
Issuer
Public Ledger
1. transaction request
list of statement types and attestation specs
collect statements and attestations
2. obtain missing statements and/or attestations
store possible revocation
3. list of statements and attestations
construct valid argument, then commit or decline
check for revocations etc.
ok or not ok

Figure 16: Sequence diagram for transaction request

6.3 Installation and Administration

The software and instructions for its installation / deployment and administration are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/id-bp-management-ssif-enabler

The group is administered by TNO.

TNO has developed an SSI framework in Ampersand (the Ampersand programming language is based on relation algebra), to build, for example digital forms. Credentials can be loaded in the MyID app, and the proxy is also available[1] (details included in the dedicated gitlab group). Sovrin is based on Hyperledger Indy[2] and Hyperledger Aries[3] an open-source projects/toolkits to create, store and share credentials. Tooling for the Sovrin blockchain is available in the Sovrin GitHub[4].

6.4 Open API

The technical documentation and the software of the Storage enabler are available in the following Gitlab group:

[1] https://ci.tno.nl/gitlab/ampersand/ssif
[2] https://www.hyperledger.org/projects/hyperledger-indy
[3] https://www.hyperledger.org/projects/aries
[4] https://github.com/sovrin-foundation

https://gitlab.com/block.is-enablers/id-bp-management-ssif-enabler

The group is administered by TNO.

7 Semantic Ledger Enabler

7.1 Basic Concepts

The *Semantic Ledger* enabler is a data-sharing platform that facilitates the implementation of a transparent and immutable supply-chain ecosystem. Semantic Ledger is built as a software layer on top of existing distributed ledger technology.

Within the Semantic Ledger ecosystem it is possible to specify visibility for the published data, making it possible to publish confidential data on the ledger. Also, the platform supports transactional confidentiality. This feature ensures it is not obvious which parties interact through the platform.

7.2 Architecture and Interactions

In this Section we describe the high-level architecture and the main interactions (using sequence diagrams).

7.2.1 High level Architecture

Semantic Ledger utilizes semantic technology to achieve data interoperability within the ecosystem. By semantically structuring the data, it is possible to refer to any other data within or outside the system, given this data is also semantically annotated and referenceable. The use of semantics within the platform enables users of the platform to specify or use ontologies to describe their data, which enables them to model and specify business rules with the published data. By utilizing semantic rules and ontologies, published by trusted parties, the data becomes inherently more valuable. Anyone with access to the data can verify compliance to the specified rules. By using semantics this way, the Semantic Ledger offers similar functionality to smart-contracts, but with more flexibility and better scalability.

The semantic ledger technology is, in principle, a ledger agnostic technology. Meaning any ledger that conforms to the requirements specified by the ledger should be able to support the semantic ledger technology.

In the current implementation, the semantic ledger is implemented on the BigchainDB ledger [10].

- The Semantic Ledger technology exists as an API and a semantic-data validator.
- The semantic ledger API is built using django-rest-framework.
- The SHACL (Shapes Constraint Language) Validator is built in Typescript and NodeJS.
- The domain-specific API can be built using any technology that supports REST-API calls.

7.2.2 Interactions

The semantic ledger technology provides an API, to enable other applications to connect to the ledger. The Semantic Ledger includes a semantic validator API, which validates publications against provided rules (which are stored on the ledger), as depicted in Figure 17. The interactions include the following:

1. A user wants to publish data, the user enters data into a GUI or composes an API call.

2. The domain application translates this input-data into semantic data, and attaches a rule-reference to the publication (specified as an application-wide setting). This is published as API call towards the Semantic Ledger API
3. The semantic ledger validates the publication against the provided rules. If rules are broken, a validation-report is returned to the domain-application, which then forwards this to the user as a publication-error.
4. If validation succeeds, the publication is published to the ledger API. The ledger creates a new asset on the ledger, and returns an asset-id, which is returned to the domain-application, which in turn confirms this to the user

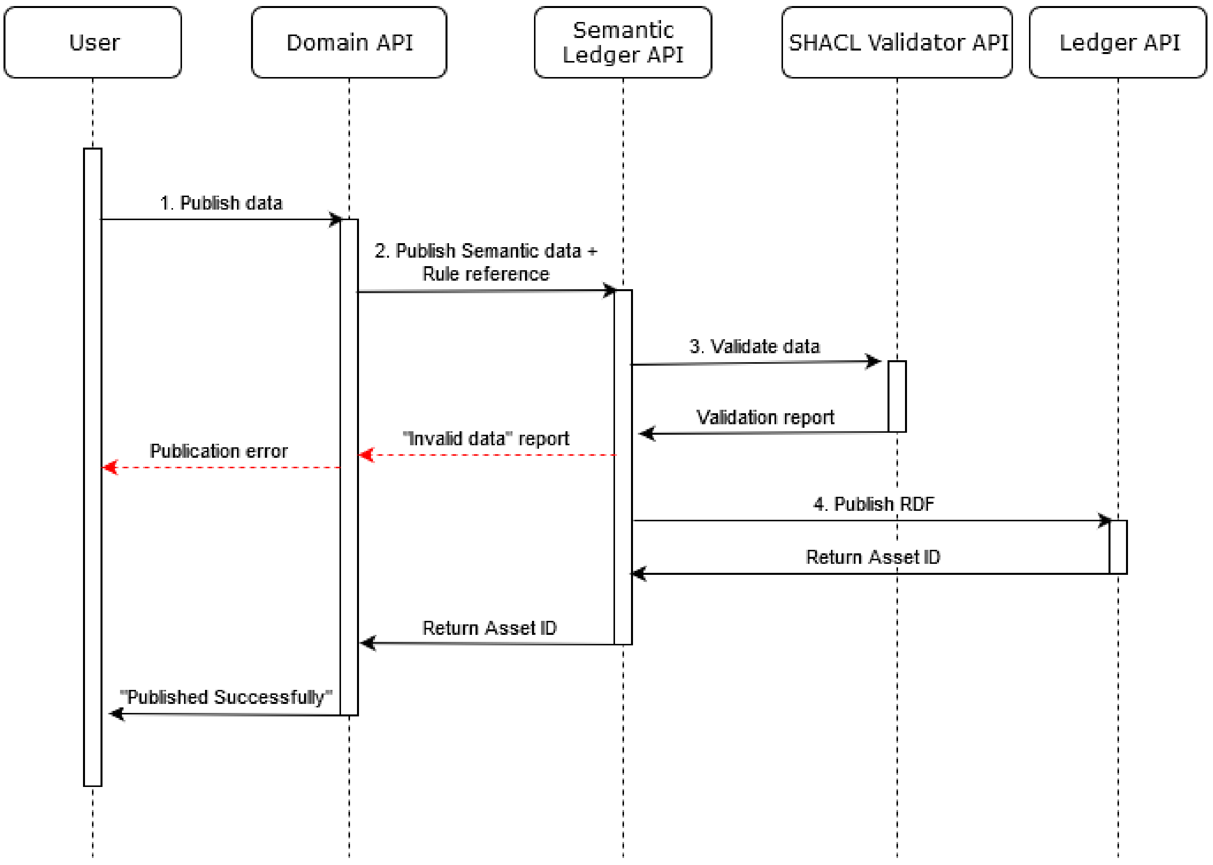


Figure 17: Interactions from the semantic ledger enabler

Invalid data can be present on the ledger (by directly publishing data to the underlying ledger). So, whenever data is read through the Semantic Ledger layer, data is validated (Figure 18). The interactions include the following:

1. A user request data using a GUI, or by calling a domain-api endpoint.
2. The domain-api calls the “request asset” endpoint from the semantic-ledger API
3. The semantic-ledger application request the specified asset from the ledger
4. The returned asset is read, and when rules are specified, the rules are also requested from the ledger.
5. The requested data is validated against the rules that are received from the ledger and returned to the domain-application, which returns the data to the user (in GUI or as API-response). If the data is not validated, the semantic-ledger API also returns a validation report.

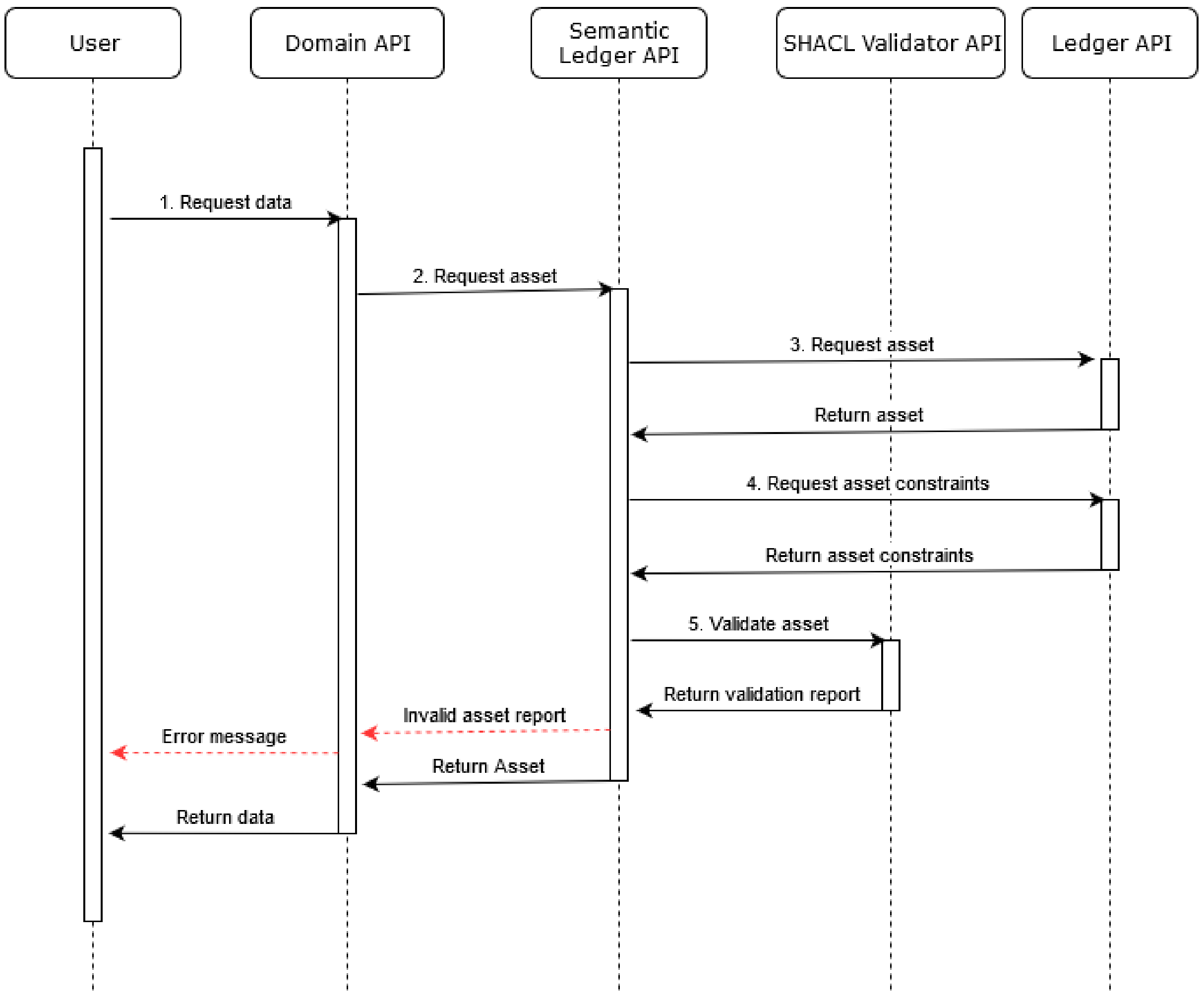


Figure 18: Sequence diagram for semantic ledger (validation)

The rules are provided to the Semantic Ledger using by the following process (Figure 19):

1. The rules are published directly to the Semantic Ledger API, formatted as SHACL constraints.
2. The semantic ledger validates the rules and, when valid, publishes these to the ledger. The ledger returns an asset-id, representing the rules.
3. The user specifies the asset-id of the rules to the domain-API. These rules are now attached to the domain-publications.

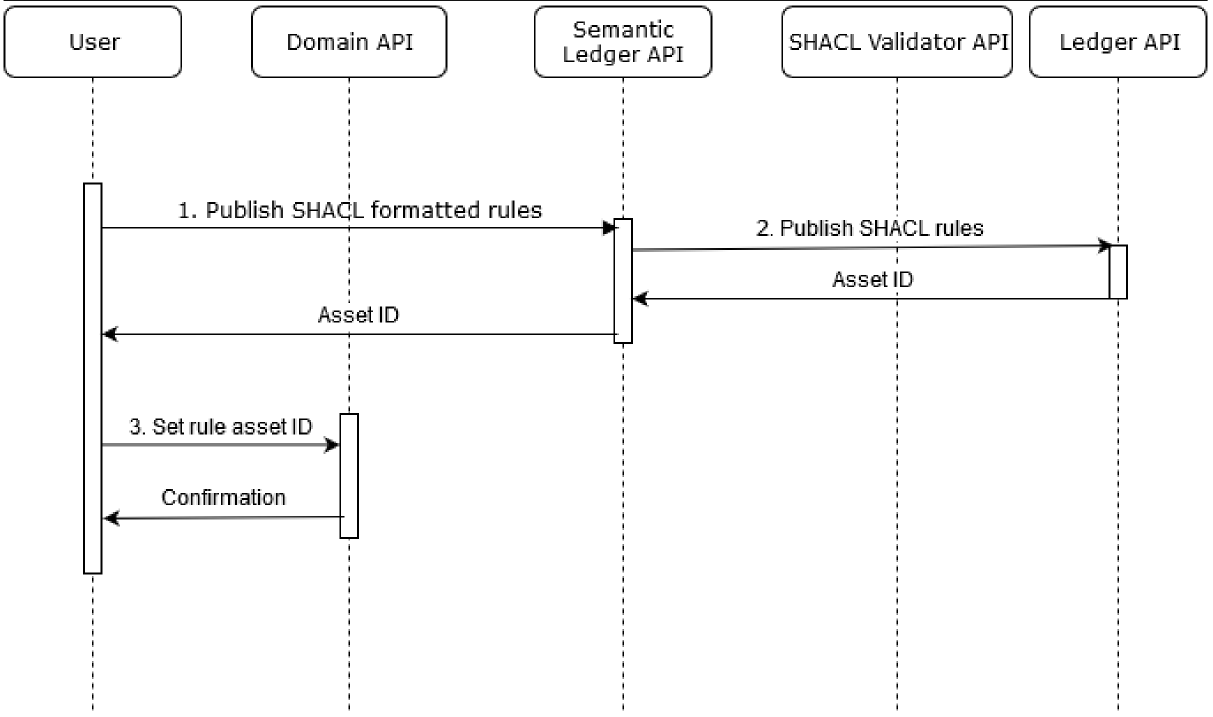


Figure 19: Sequence diagram for the semantic ledger enabler (provision of rules)

7.3 Installation and Administration

The software and instructions for its installation / deployment and administration are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/semantic-ledger-enabler

The group is administered by TNO.

7.4 Open API

The technical documentation and the software of the Storage enabler are available in the following Gitlab group:

https://gitlab.com/block.is-enablers/semantic-ledger-enabler

The group is administered by TNO.

8 Discussion and Future Steps

Deliverable D2.2 describes the Blockchain-based enablers offered by the Block.IS consortium members (and specifically INTRASOFT International and TNO). As discussed, in the context of the work of T2.2, 6 enablers have been selected and specified. The functionality indicated in the Description of Action has been covered (offering an increased number of enablers from those initially expected).

For each of the enablers, a dedicated section has been prepared elaborating the basic concepts, the high level architecture and the interactions (using sequence diagrams). Links to gitlab groups have been provided that accommodate the API and further instructions related to deployment, configuration and collaboration with the Blockchain infrastructure.

The enablers described in the current deliverable are being offered in the open calls of the project. Starting with the 1st open call we expect to gather the experiences and the lessons learnt from the potential adopters in order to further enhance and improve the enablers. A report on the lessons learnt and the continuous improvements in the deliverables are expected to be documented in dedicated section in the forthcoming WP2 deliverables namely D2.3 (Block.IS technical and business support service design – v2) and D2.4 Block.IS (technical and business support service design – lessons learnt).

We also expect that the gitlab groups and repositories, in alignment with the technological and business support services provided by the project, will continuously evolve and allow for fruitful collaboration between the Block.IS partners and the interested SMEs and start-ups.

References

1. Block.IS Project “Description of Action” (DoA)
2. uPort, Identity system for decentralised web, Technical Documentation: https://github.com/uport-project/uport-identity/blob/develop/docs/reference/ (accessed September 2019)
3. Ethereum open source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality, www.ethereum.org (accessed September 2019)
4. IPFS, InterPlanetary File System, peer-topeer hypermedia protocol, https://ipfs.io (accessed September 2019)
5. OASIS. UDDI Version 3.0.2. UDDI Spec Technical Committee Draft: http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm (2004)
6. Apache Zookeeper. https://zookeeper.apache.org/ (accessed September 2019)
7. Google Marketplace API. Retrieved from https://developers.google.com/authorized-buyers/apis/guides/v2/marketplace-api (accessed September 2019)
8. Deventer, D. I., Self-Sovereign Identity – The Good, the Bad and the Ugly, Retrieved from https://blockchain.tno.nl/blog/self-sovereign-identity-the-good-the-bad-and-the-ugly/
9. Django REST Framework. (2019). Technical documentaion https://www.django-rest-framework.org/ (accessed September 2019)
10. BigChainDB. Technical documentation https://www.bigchaindb.com/ (accessed Septembe 2019)