SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D2.5

Federation Framework, 2nd version

Project title	SOFIE – Secure Open Federation for Internet Everywhere
Contract Number	H2020-IOT-2017-3 – 779984
Duration	1.1.2018 – 31.12.2020
Date of preparation	30.8.2019
Author(s)	Yki Kortesniemi, Dmitrij Lagutin, Andrea Lisi, Mahdi Ghorbani (AALTO), Vasilios Siris, Nikos Fotiou (AUEB-RC), Giuseppe Raveduto (ENG), Margus Haavala (GT), Filippo Vimini (LMF), Ahsan Manzoor (ROVIO), Sotiris Karachontzitis (SYN)
Responsible person	Yki Kortesniemi (AALTO), Yki.Kortesniemi@aalto.fi
Target Dissemination Level	Public
Status of the Document	Completed
Version	1.00
Project web-site	https://www.sofie-iot.eu/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779984.

Table of Contents

List of abbreviations

API	Application Programming Interface
AS	Authorisation Server
CSO	Charging Station Owner
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DSO	Distribution System Operator
EV	Electrical vehicle
FSC	Food Supply Chain
HTLC	Hash Time-Lock Contract
IAA	Identity, Authentication, authorisation
IoT	Internet of Things
KSI	Keyless Signing Infrastructure (GuardTime)
PoI	Point of Interest
RFID	Radio Frequency IDentification
TD	Things Descriptor
TSO	Transmission System Operator
WoT	Web of Things

1 Introduction

SOFIE (Secure Open Federation for Internet Everywhere) is a three-year EU Horizon 2020 research and innovation project that provides interoperability between existing IoT platforms in an open and secure manner.

This document continues the work started in the SOFIE deliverable D2.3 [Paa2018] and presents the *SOFIE Federation Framework,* an example implementation of the *SOFIE Federation Architecture* introduced in the deliverable D2.4 [Elo2019]. The Architecture consists of 6 internal components and the Framework provides an example implementation of each, as well as a description of the components' purpose, interfaces, and internal structure. Scenarios derived from the SOFIE pilots described in deliverable D5.2 [Oik2019] are used to explain how each component can be used and how the components meet the requirements set in D2.4. Finally, there is a description of how the SOFIE pilots leverage the components. More detailed technical descriptions of the components and their interfaces can be found in the technical documentation accompanying the component code release available at https://github.com/SOFIE-project/Framework.

During the SOFIE project, the architecture and framework will be developed further; the deliverables D2.6 and D2.7 detailing these developments will be released in late 2020.

The rest of the document is organised as follows: Section 2 presents an overview of the SOFIE Architecture and Framework. Then, Sections 3-8 detail the 6 internal components, while Section 9 describes how the SOFIE pilots utilise these components and how the pilots have implemented the interface components in their respective environments.

2 SOFIE Architecture & Framework

This section introduces the SOFIE Federation Architecture used to federate operations across IoT systems (silos) using distributed ledgers (DLTs) as bridges, and the SOFIE Federation Framework, which is an example implementation of the Architecture.

2.1 SOFIE Architecture

The lack of interoperability between IoT systems has long been a significant limitation to the creation of new solutions spanning multiple IoT system. This situation has been further aggravated by the fact that adding interoperability to existing IoT systems can be hard due to the systems not being upgradable, especially when the systems are owned by multiple organisations, which can create further integration challenges due to the related trust and liability issues.

The goal of the SOFIE Architecture is to overcome these challenges. It supports the integration of IoT systems owned and operated by different organisations using a federation approach and overcomes the lack of upgradability by using adapters to link the IoT devices to the architecture. The SOFIE Architecture has been described in detail in deliverable D2.4 [Elo2019]. It defines SOFIE as a *framework architecture*, i.e. an architecture that defines types of functionalities but not a single exact implementation for those functionalities, due to the fact that SOFIE can be used to bridge so many different types of applications in so many fields that no single set of functionalities or APIs is convenient to serve them all. Therefore, the Architecture can be used as a foundation to define suitable functionalities and interfaces for each application domain.

Figure 2.1 provides a functional overview of the SOFIE architecture. It depicts the internal components of the SOFIE framework (orange boxes), the SOFIE interface components (white boxes with orange outline) and their cross-domain interactions with external domains/components (white boxes with black outline).

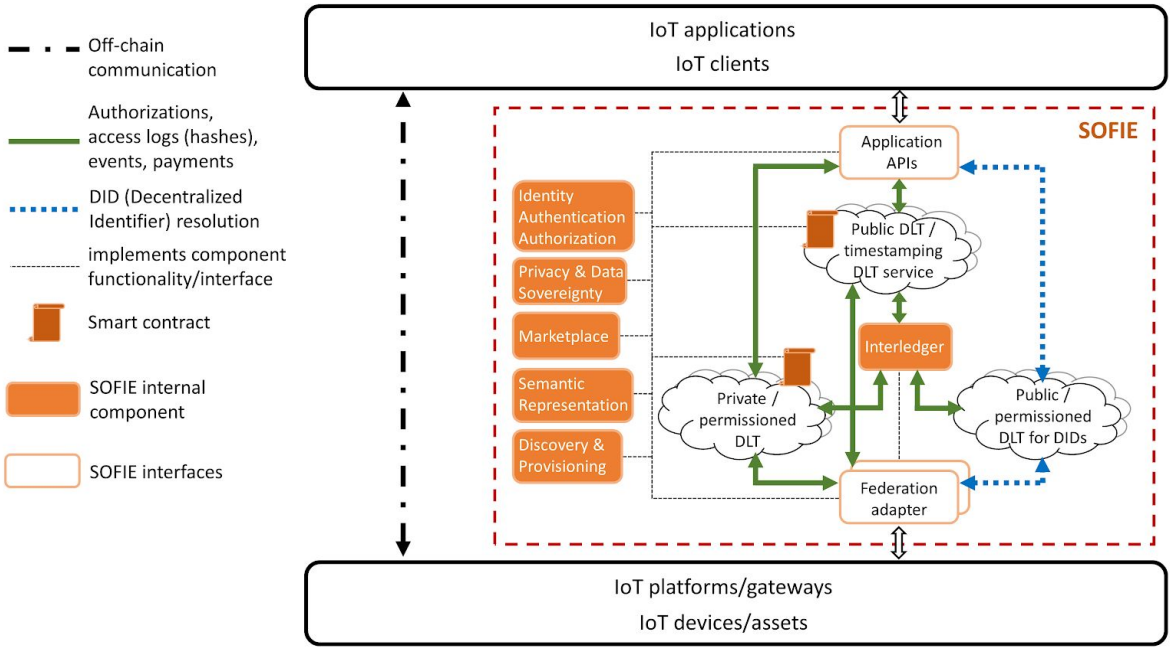

Figure 2.1: The SOFIE framework architecture

The lowest level of the architecture contains the IoT assets (or resources) that include, e.g. IoT sensors for sensing the physical environment, actuators for acting on the physical environment, and boxes with RFID tags used to transport products. IoT platforms include platforms with data stores, where the measurements from sensors are collected and made available to third parties, as well as servers providing IoT services.

The interface component *Federation Adapter(s) (FAs)* are used to interface the IoT platforms with the SOFIE framework. This allows the IoT platforms to interact with SOFIE without requiring any changes to the IoT platforms themselves. Note that a part of the adapter's functionality can be implemented, e.g. in smart contracts. Moreover, different scenarios and pilots require different types of federation adapters, which may implement only the required parts of the SOFIE functionality.

The main functionality of the Architecture is provided by the 6 internal components introduced in Subsection 2.2 and detailed in Sections 3-8.

The Architecture emphasises the interledger functionality responsible for interconnecting the different types of DLTs, which can have quite different features and functionality. Public (or permissionless) DLTs can offer wide-scale decentralised trust and immutability, but this necessitates a large network with many peers and/or a more demanding consensus mechanism, thereby incurring a higher overall computation cost that will lead to longer transaction confirmation times. On the other hand, permissioned or consortium DLTs have a lower, or even zero, transaction cost and low latency; however, trust is determined by the peers in the set of permissioned nodes that participate in the DLT's consensus mechanism. Moreover, the level of privacy afforded also differs: the transactions and data on public DLTs are completely open to everyone, which is necessary to achieve wide-scale decentralised trust and transparency, but forgoes any privacy. On the other hand, permissioned DLTs involve the collaboration of peers that belong to a specific permissioned set and can arrange their records to be inaccessible to others (private), or public (but only allowing the permissioned set to contribute to the DLT). Thus, permissioned blockchains can support different levels of write and read access, which allows them to support different levels of privacy. DLTs can also differ in the functionality they provide: a DLT can focus, e.g. on cryptocurrency payments, recording of IoT events, access authorisation, or providing resolution of decentralised identifiers (DIDs) [Ree2019]. Utilising multiple ledgers that are interconnected through interledger functionality, instead of a single DLT, provides the flexibility to exploit the aforementioned tradeoffs. Finally, providing interledger mechanisms to interconnect different DLTs allows companies and consortiums to select private/permissioned distributed ledgers based on their requirements and constraints. Hence, interledger mechanisms can enhance interoperability across different IoT platforms that utilise different distributed ledger technologies.

The architecture also illustrates the separation of data transfer and control message exchanges. Some IoT data can be transferred directly between the IoT platforms and IoT clients. Control messages related to authorisation logs, events, payments, etc. go through the SOFIE framework. IoT data or hashes of data can also be handled by the SOFIE framework.

Finally, the upper component of the architecture is the interface component *Application APIs*, which provides the interfaces for IoT clients and applications to interact with the SOFIE framework. Similar to FAs, these are application specific.

2.2 SOFIE Framework

The SOFIE Federation Framework is an example implementation of the SOFIE Architecture developed to support the SOFIE pilots and to serve as an example for future Architecture

implementations. It provides a description of each of the 6 internal components and an overview of their implementation, while the related code release available at https://github.com/SOFIE-project/Framework includes the more technical description of the implementation.

In the Architecture, the *internal components* describe types of functions, which can be implemented as separate components or as part of the interface components (Federation Adapters and Application APIs). The 6 internal components are:

- *Interledger*, which provides support for operations spanning two or more ledgers, including support for atomic transactions over multiple ledgers.
- *Identity, Authentication and Authorisation*, which provides IAA functionalities for the different entities in the system by supporting multiple authentication and authorisation techniques.
- *Privacy and Data Sovereignty*, which provides mechanisms that enable data sharing in a controlled and privacy preserving way.
- *Semantic Representation*, which is used to enable interoperability between different IoT devices, services, and data by describing what functions they provide and what interfaces and formats they utilise.
- *Marketplace*, which allows participants to trade resources by placing bids and offers in a secure, auditable, and decentralised way.
- *Discovery & provisioning*, which provides functionality for the discovery and bootstrapping of IoT devices, services, and data.

The design and function of the internal components are discussed in Sections 3-8.

The framework also includes 2 interface components:

- *Federation Adapters*, which interface with the IoT devices/platforms.
- *Application APIs*, which interface with the applications utilising the SOFIE Framework.

The interface components are highly application specific and will have to implement different protocols, etc. depending on the application domain and devices used. They can also be used to implement some or even all of the functionalities of the internal components. The interface components used in the SOFIE pilots are discussed in Section 9.

.

3 Interledger Component

The purpose of the SOFIE interledger component is to enable transactions between actors and devices belonging to different (isolated) IoT silos. Each silo either utilises or is connected to one or more ledgers, and the interledger component then enables interaction between the ledgers. The techniques proposed in literature to enable operations and transactions between different ledgers are described in [Sir2019a].

By providing interledger transaction capabilities, SOFIE enables semantic level communication between the different silos. Thanks to the Interledger component, it is possible to, e.g. integrate multiple ledgers to a cohesive storage platforms that enables the most suitable type of DLT to be used for the type of information at hand and to enable cross-ledger transactions, thus harnessing the individual strengths of the different DLTs.

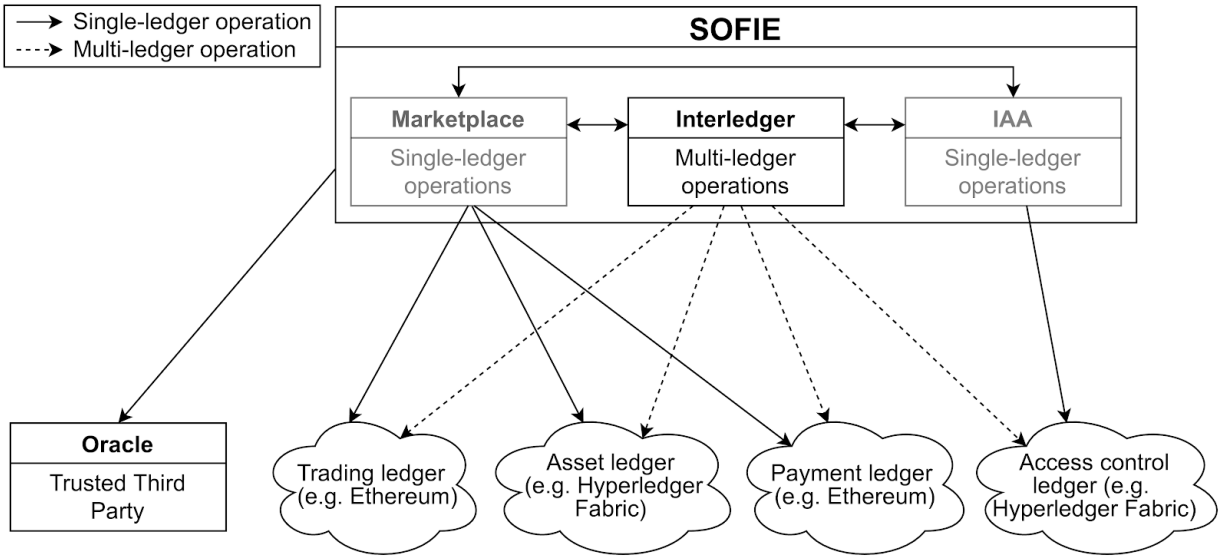

Figure 3.1: Connection between the SOFIE components and the ledgers

Figure 3.1 shows a few examples of ledgers and their connection to the SOFIE components. Different types of ledgers can be used for different purposes, so, e.g. Asset and Access control ledgers can be permissioned ledgers like Hyperledger Fabric to reduce costs, while payments and trades can take place on public ledgers like Ethereum to improve trust.

Each ledger can also be accessed by multiple SOFIE components. The division of work is that Interledger is responsible for implementing the multi-ledger operations while other components implement operations that deal with individual ledgers. In Figure 3.1 the dashed lines represent multi-ledger operations, while continuous lines represent single- ledger operations.

Finally, the *oracle* is a trusted third party in charge of making authoritative statements about the status of some system. For example, the oracle may track the charging events in the Decentralized Energy Flexibility Marketplace pilot and notify the auction once the agreed amount of energy has been consumed in the correct district. The presence of the oracle and its connections with the ledgers depends on the use case.

3.1 Requirements and Scenarios

Table 3.1 summarises the requirements for this component (from SOFIE deliverable D2.4).

Table 3.1. Requirements for the Interledger component

ID	Requirement Description	Priority	Category
RF01	User interaction is not required for interledger operations.	MUST	USABILITY
RF02	There should be support for atomic interledger operations.	SHOULD	SECURITY

Table 3.2 presents example scenarios, derived from the use cases presented in SOFIE deliverable D5.2, to cover all requirements described above.

Table 3.2. Scenarios derived from the pilots use cases

ID	Scenario Content	
SC01	Description	A player of the location-based game wants to sell a Vorpal Sword +2 in exchange for game tokens.
	Stimulus	The player chooses the Sell action on his game app.
	Response	The app sends a request to the game server, which in the asset ledger labels the Vorpal Sword +2 as an asset that is being traded in the game marketplace. This action triggers an Interledger operation which adds the relevant sword information from the asset ledger to the public trading ledger to initiate the trade.
	Derived from use case	MRMG _UC4: Asset Trading
	Covers requirements	RF01
SC02	Description	At the end of the food supply chain (FSC), the transportation company (TR) employee gives the smart box to the supermarket (SM) employee.
	Stimulus	The TR employee gives the smart box to the SM employee and records the action.
	Response	Based on the handover event, the Supervisor (see Section 9.1 for more details) of the supply chain first produces the necessary smart box-related data and its fingerprint (hash). Then, an interledger operation is triggered to atomically store the data in the consortium ledger, and the fingerprint together with handover related public data to the public ledger.
	Derived from use case	FSC_UC8 : Hand over product: TR-SM
	Covers requirements	RF02

3.2 Services and Interfaces

As shown in Figure 3.2 and listed in Table 3.3, the Interledger component provides the following interfaces:

- **Atomic transactions:** this interface provides the operations to atomically send multiple transactions to two or more ledgers. The Interledger component should guarantee that either all or none of the data is stored, as otherwise the system may end up in an inconsistent state.
- **Data transfer:** this interface provides the operations to transfer some data from one DLT to another. With it, it is possible to implement atomic data exchange between two ledgers, like Atomic Swaps [Sir2019a].
- **Proof of Integrity:** this interface provides operations to verify that a set of data (e.g. stored in an internal ledger) still matches a fingerprint previously stored in a (public) ledger, thus guaranteeing that the data has not been modified since the fingerprint was created.

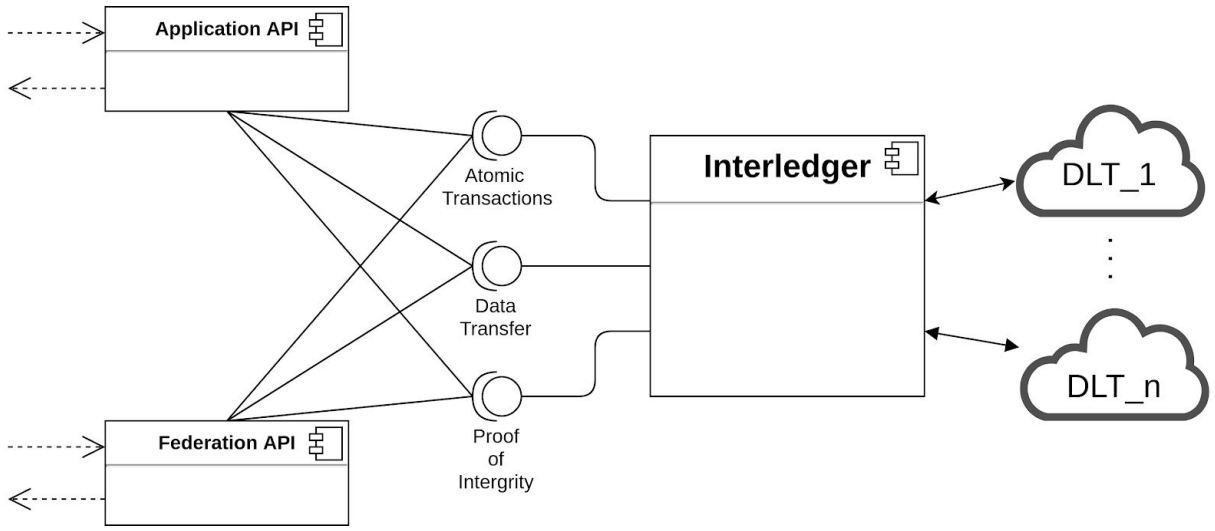

Figure 3.2: The interfaces of the Interledger component

The Interledger interfaces required by the system are invoked through either the SOFIE Application or Federation APIs, depending on the use case. Moreover, the Interledger component interacts with the other SOFIE components, e.g. when it queries the IAA component (described in Section 4) during a data audit to verify whether a caller has the right to access the data stored in the consortium ledgers, or when the Marketplace component (described in Section 7) relies on the Interledger component to trigger multi-ledger marketplace operations. Finally, the Interledger component is connected to two or more DLTs.

Table 3.3. Interfaces of the component

ID	Interface Content	
IF01	Name	**Atomic transactions**
	Description	Provides operations to execute a set of transaction on multiple ledgers in an atomic way: either all or none of the transactions are finalised.
	Key inputs	The transaction set and the ledger identifiers.
	Response	Either all or none of the transactions are executed successfully on all the ledgers involved, modifying the state of the ledgers accordingly.
IF02	Name	**Data transfer**
	Description	Some data stored in a ledger will be transferred to another ledger. This interface supports only two ledgers.
	Key inputs	The ledger identifiers, the data to be transfered.
	Response	The data will be removed from the starting ledger, and it will only be present in the destination ledger.
IF03	Name	**Proof of Integrity**
	Description	Verify if a set of data has not been tampered with.
	Key inputs	The ledger addresses of the data to verify (stored in one ledger), the ledger address of the associated fingerprint (stored in another ledger), and the identifier of the hash function used.
	Response	True if the computed fingerprint over the data matches the stored fingerprint, False otherwise.

3.3 The internal structure

Since the Interledger component communicates with multiple DLTs, it should abstract each ledger API (e.g. Web3 for Ethereum[1]) in order to be reusable. As shown in Figure 3.3, the most common operations concerning ledgers are abstracted by an interface, called ILedger in the figure. For example, if the Interledger component needs to support Ethereum, then an abstraction of the Web3 library should be implemented (EthereumLedger in the figure) following the ILedger interface. In this way, it is possible to have a general baseline of the component.

Instantiating an ILedger implementation creates a connection to a specified ledger. As a result, the Interledger component needs to instantiate multiple ILedger objects to connect to all ledgers. Each ILedger object instance should be paired with a unique identifier, which in turn will be used by external components requesting Interledger operations.

[1] https://web3py.readthedocs.io/en/stable/

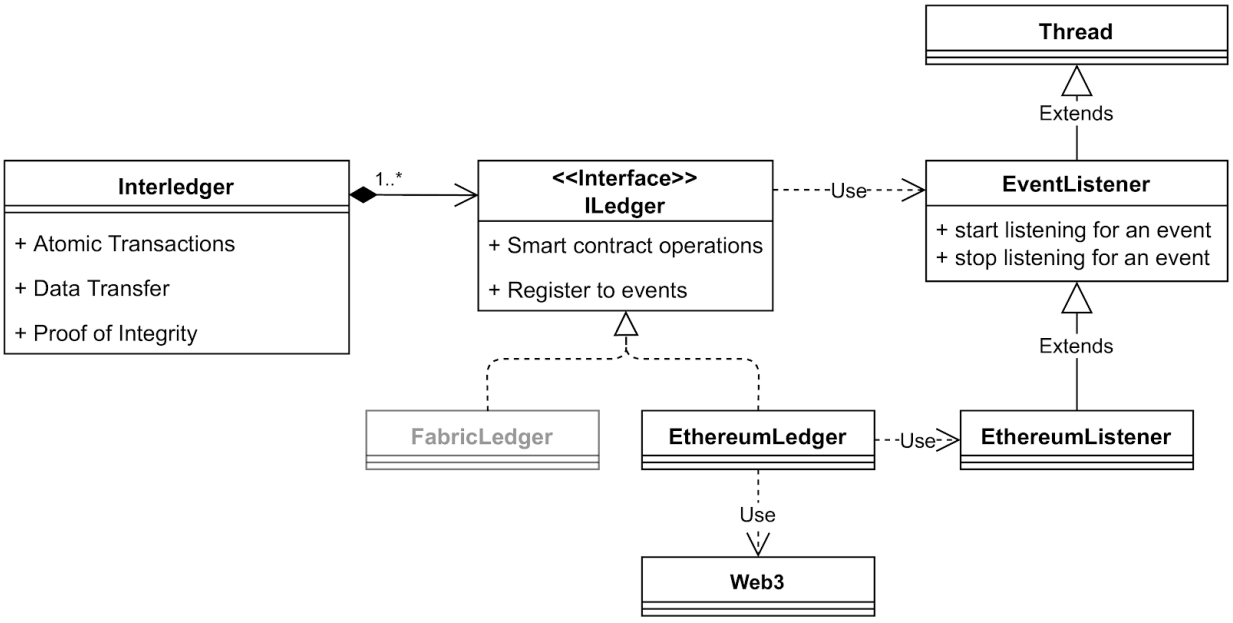

Figure 3.3: The Interledger component's internal structure

Atomic transactions

When two or more transactions are issued with the requirement to be atomic, the Interledger component has to provide a technique to ensure that either all or none of them will be executed in the target DLTs. Such atomicity is achieved by means of Hash Time Locked Contracts (HTLC) (see SOFIE deliverable D2.4 and [Sir2019a] for more details). Within the Interledger component, HTLC functionality is implemented using both software modules and smart contracts.

Data transfer between DLTs

During the transfer of data, the Interledger component acts as a bridge between the two involved ledgers. The Interledger component supports the registration of callbacks that are triggered when a smart contract in one DLT executes a particular operation, e.g. a trading operation, and emits an event. The callback should then implement the transfer of data. The EventListener object, instantiated by the ILedger interface as shown in Figure 3.4, is responsible for catching the events emitted by the smart contracts. When defining a ILedger implementation, such as EthereumLedger, the event loop is implemented by extending the EventListener class (EthereumListener in the figure).

Figure 3.4 describes the flow of the data transfer operation. The user performs an operation which needs to transfer some data *D* from DLT1 to DLT2. The smart contract running on DLT1 executes an internal operation which triggers a callback registered by the Interledger component: this callback is in charge of transferring *D* (or a subset of it) to DLT2. Finally, the transaction receipt (*tx_receipt)* resulting from the (positively concluded) transaction will be returned to the application.

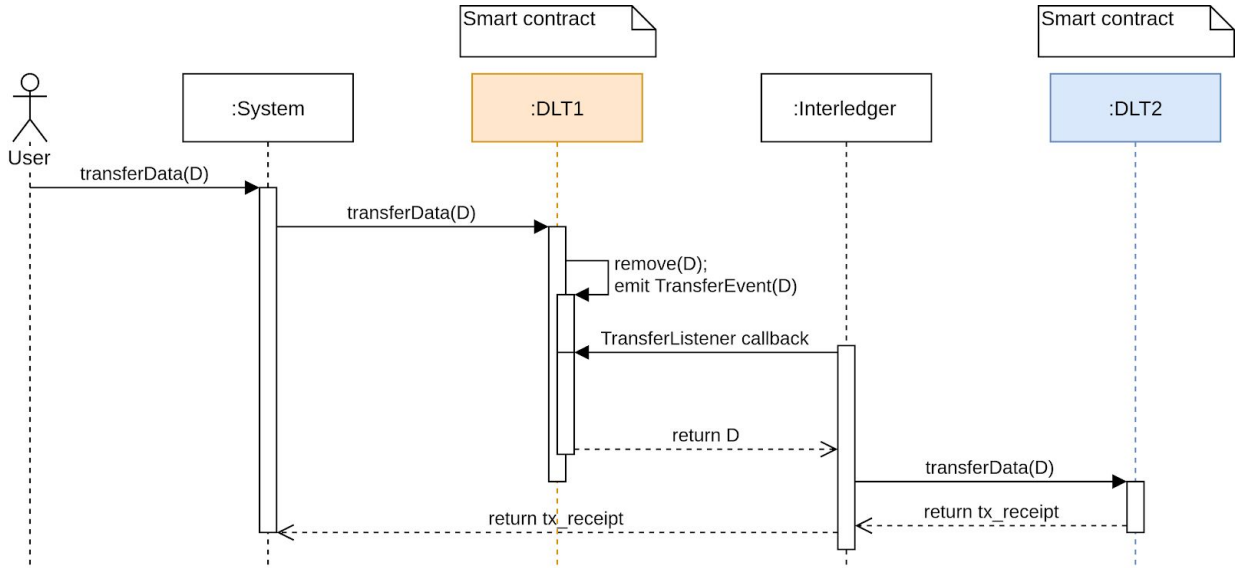

Figure 3.4: Sequence diagram of data transfer between DLTs

Proof of Integrity

To verify the integrity of a set of data, the Interledger component requires the knowledge of the hash algorithms used to produce the fingerprint of that data. Hence, the Interledger component keeps an internal repository of hash functions used within the application domain. As shown in Figure 3.5, the auditor requesting to verify the integrity of a set of data provides the identifier of the transaction generating that data. The system retrieves the id of the hash function used to produce the fingerprint, and the data and the fingerprint ledger addresses connected to the input transaction id. The Interledger component receives these parameters, retrieves the data and its fingerprint from the consortium and public ledgers respectively and finally applies the relevant hash function. The result of the comparison between the calculated fingerprint and the original one is returned back to the auditor.

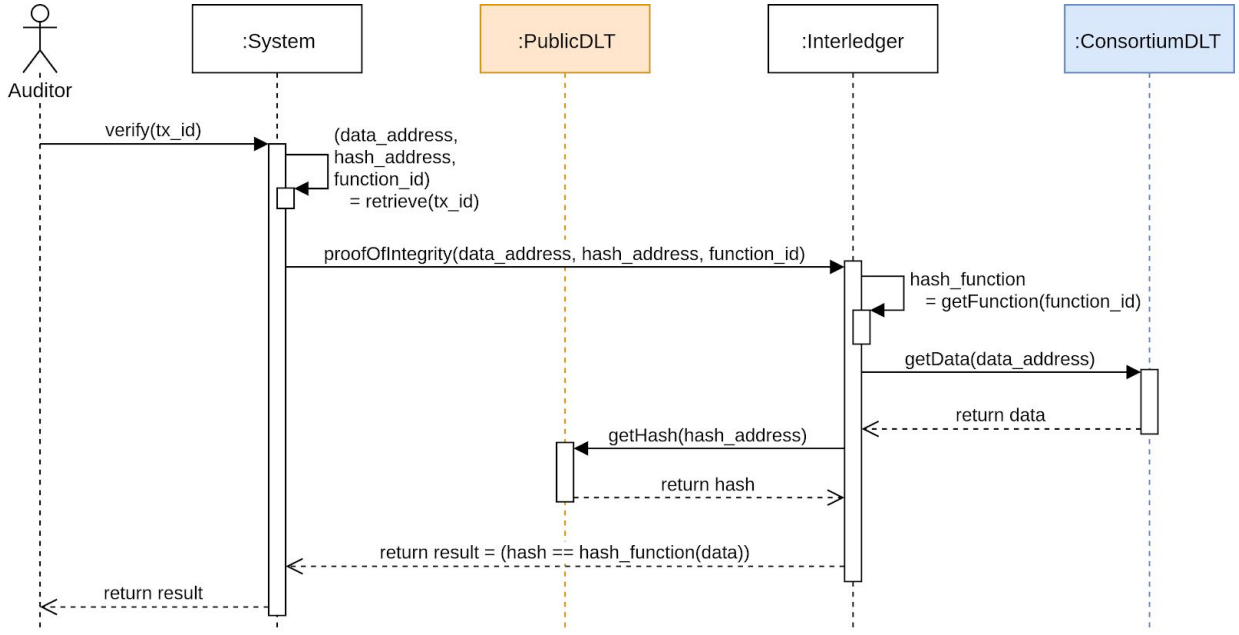

Figure 3.5: Sequence diagram of verifying the proof of integrity using Interledger

3.4 Scenario walkthrough

Table 3.4 describes how the Interledger component satisfies the scenarios presented in section 3.1.

Table 3.4. Scenario Walkthrough

ID	Scenario Validation	
W01	Description	A player sets up for sale a gaming asset. [SC01]
	Walkthrough	When a player wants to sell a Vorpal Sword +2 he interacts with the game app and presses the sell button on his asset. The game sends a sell request to the game server, which marks the asset as 'in trade' on the asset ledger by calling the dedicated smart contract. The smart contract marks the asset's status as in 'in trade' in the asset ledger and emits an event notifying that the asset is being traded. This event triggers the Interledger callback registered to the trading event by means of interface IF02. The callback executes the smart contract stored on the trading ledger. This smart contract marks the Vorpal Sword +2 item as 'in trade' in trading ledger, meaning that interested parties may bid for the asset. Finally, the Interledger component retrieves a transaction receipt (that the asset is now being traded) from the trading ledger operation, and forwards it to the game server.
W02	Description	Registration of the last handover operation in food chain pilot. [SC02]
	Walkthrough	At the last step of the food supply chain, the TR employee hands over a smart box to the SM employee. Therefore, the system invokes the operations provided by IF01 to store the smart box metadata and fingerprint in the consortium and public ledgers, respectively, according to the pilot's specifications. The key identifiers of the DLTs involved will be provided as input. The Interledger component initiates a series of transactions meant to be atomic. If the operation succeeds, both the consortium and the public ledger will have the smart box information properly stored; otherwise, neither of the ledgers will have the state changed.

4 Identity, Authentication and Authorisation Component

The goal of the *Identity, Authentication and Authorisation* (IAA) component is to provide mechanisms that can be used for identifying communicating endpoints, as well as for authenticating and authorising users wishing to access a protected resource.

In its present form, the IAA component can authenticate users using username/passwords, or *Decentralised Identifiers* (DIDs) and it uses the OAuth 2.0 protocol for user authorisation. The user authorisation process is enhanced by a smart contract that provides conditional release of an access token (e.g. when a user performs a payment). More details about the operations performed by this smart contract are presented in section 4.3.1, as well as in [Sir2019b].

4.1 Requirements and Scenarios

Table 4.1 summarises the requirements for this component.

Table 4.1. Requirements for the IAA component

ID	Requirement Description	Priority	Category
RF03	Resource owners must be able to delegate the authentication and authorisation tasks for their resources.	MUST	OPERA-TIONAL
RF04	The IAA component must provide users the capability to revoke authorisations.	MUST	SECURITY
RF05	The IAA component must allow individuals to control their personal information and digital identities (e.g. support self-sovereign identity technology).	MUST	SECURITY
RF06	The IAA component must support secure, tamper-proof, and verifiable logging of transactions and events.	MUST	SECURITY
RF07	The IAA component must support Role Based Access Control (RBAC).	MUST	SECURITY
RF08	Cryptographic algorithms used by SOFIE should be open-source and transparent and as independent as possible of any particular architecture.	SHOULD	AUDITA-BILITY
RF09	SOFIE should support the execution of authorisation and authentication functionality on devices with constrained processing, storage, battery, and network connectivity.	SHOULD	OPERA-TIONAL

Table 4.2 presents example scenarios derived from use cases in SOFIE deliverable D5.2 to cover all requirements from the previous subsection.

Table 4.2. Scenarios derived from the pilots use cases

ID	Scenario Content	
SC01	Description	A producer accesses the Food Supply Chain (FSC) web application to specify the box(es), which contain products to be delivered to a transportation (TR) employee. Then, the TR employee accesses the FSC web application, authenticates himself, accepts the responsibility of these boxes, and confirm the transaction.
	Stimulus	The producer interacts, through the FSC web application, with the appropriate interface of the authorisation server.
	Response	The producer has delegated access control decisions to the authorisation server, and a set of Role based Access Control (RBAC) policies have been defined. Access control decisions and confirmations of the transactions are recorded in the blockchain.
	Derived from use case	FSC_UC3
	Covers requirements	RF03, RF06, RF07
SC02	Description	A smart meter data owner (owner) interacts with an authorisation server and removes access rights to his data that had been previously granted to a data consuming service.
	Stimulus	The owner interacts with the server administration module of the authorisation server.
	Response	RBAC policies are updated.
	Derived from use case	DEDE_UC4
	Covers requirements	RF04
SC03	Description	A smart meter data owner (owner) configures her identifier.
	Stimulus	The owner interacts with the interface of the client entity of the IAA framework that is responsible for managing DIDs.
	Response	DID created.
	Derived from use case	DEDE_UC1
	Covers requirements	RF05
SC04	Description	A gamer joins a challenge and starts competing for the reward. This procedure requires player authentication. Moreover, players interact with the game using various types of mobile devices.
	Stimulus	The game player interacts with the interface of the client entity of the IAA framework that is responsible for accessing an IoT platform.

	Response	Access granted and game clues are provided, or access is denied.
	Derived from use case	MRMG_UC1
	Covers requirements	RF08, RF09

4.2 Services and Interfaces

The IAA component provides the following 3 interfaces as shown in Figure 4.1 and detailed in Table 4.3:

- **Client functions:** This interface can be invoked by a client application (or other internal components) using internal procedure calls, in order to access an IoT platform, as well as to register a DID.
- **Agent I/O:** This interface is responsible for interacting with ledgers (or the interledger component), in order to access blockchain-specific functions, as well as for notifying the component about blockchain specific events.
- **AA API:** This is the REST API of the "Authorisation Server". It can be accessed directly by an OAuth2 client, or indirectly through a smart contract. Its goal is to provide client authentication and authorisation.

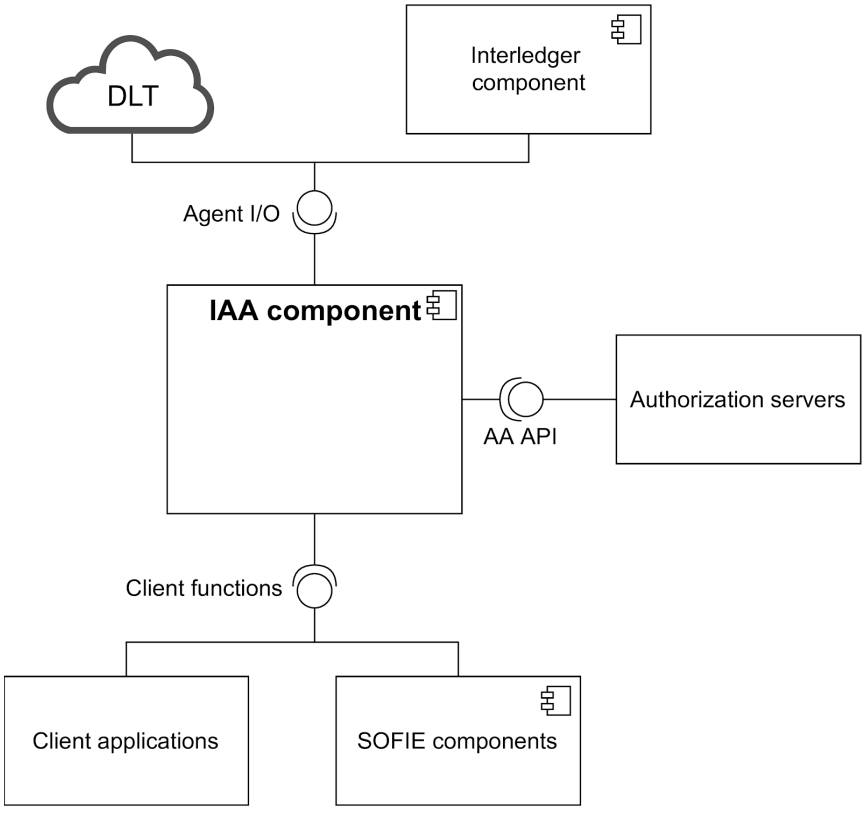

Figure 4.1: The IAA component's interfaces

The goals of the interfaces listed above are: (i) to enable generic client applications, or other components, to access the IAA operations without significant effort, (ii) to abstract blockchain access and to facilitate the incorporation of multiple ledger technologies as well as interledger approaches, and (iii) to facilitate integration with existing, external, authorisation server implementations.

Table 4.3. Interfaces of the component

ID	Interface Content	
IF01	Name	**Client functions**
	Description	It is used by client applications for accessing IoT platforms and for registering DIDs.
	Key inputs	Platform identifier, access method, client authentication data.
	Response	Access token.
IF02	Name	**Agent I/O**
	Description	The interface used for the communication with DLTs
	Key inputs	Operation, metadata.
	Response	Operation output, blockchain events.
IF03	Name	**AA API**
	Description	The REST-based interface of the Authorisation Server used for client authentication and authorisation.
	Key inputs	User authentication data.
	Response	Access token, metadata.

4.3 The internal structure

As depicted in Figure 4.2, the IAA component is composed of 5 entities: Client modules, Smart contracts, IAA blockchain agent, Authorisation Server and IoT platform modules.

Client modules

This entity includes modules that can be used by an external *client application* in order to access a platform using the IAA component. This entity is composed of the following modules:

1) *Client functions*: This module implements methods that allow client applications to interface with the IAA component (see also section 4.2, IF01)
2) *DID client*: This module provides methods that allow DID registration, as well as DID ownership verification
3) *OAuth2 client*: This module implements the OAuth2 client-side operations and operations for accessing the functionality of the authorisation smart contract running on the blockchain.

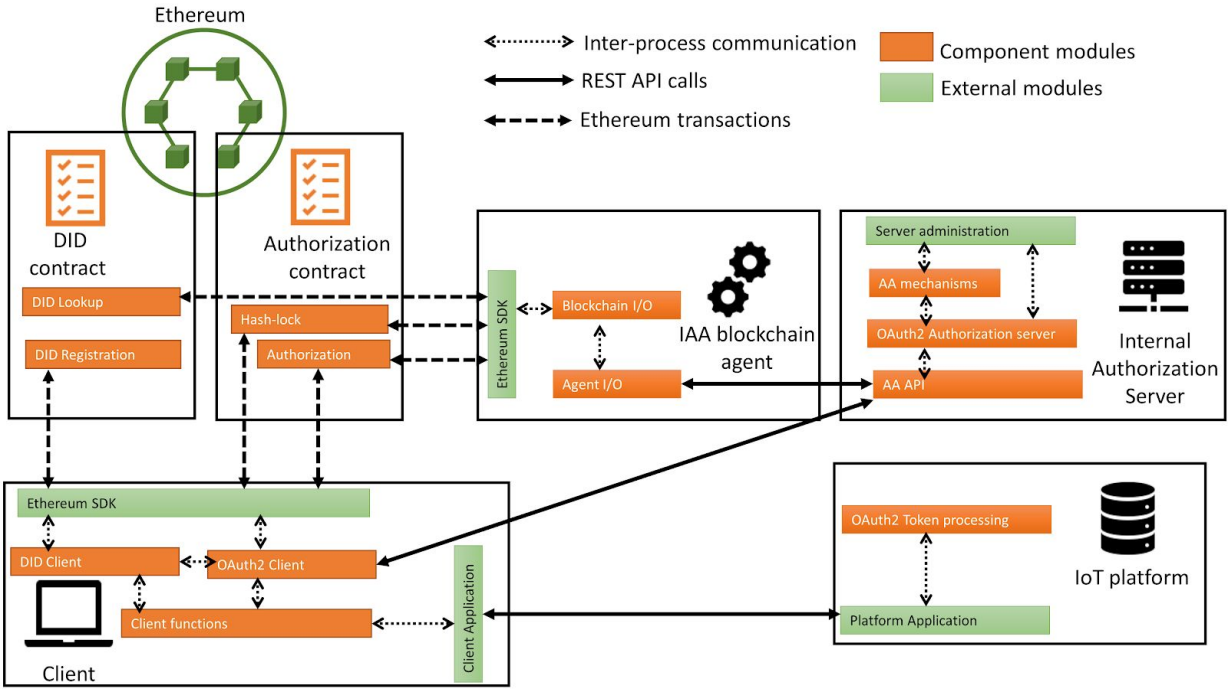

Figure 4.2: IAA component internal structure

Smart contracts

The IAA component includes two smart contracts, one for handling DIDs, and another that implements the functionality required by OAuth2-based authorisation using blockchains (this process is described in Section 4.3.1 and in [Sir2019b]).

IAA blockchain agent

This entity is responsible for mediating the communication between the authorisation server (see next) and the smart contracts. It includes the following modules:

1) *Blockchain I/O*. This module is responsible for handling the communication to and from the smart contracts (using the appropriate SDK). This module abstracts blockchain operations and its purpose is to facilitate future integrations with other blockchain.
2) *Agent I/O*. This module implements IF02 and is responsible for translating smart contract events into the appropriate REST API calls to the authorisation server (see below), as well as for implementing a REST API that can be invoked by an authorisation server when it requires functionality provided by the smart contracts.

Authorisation server

This entity is an enhanced version of the OAuth2 php server[2] and includes the following modules:

1) *AA API*. This component implements IF03, i.e. a REST API that can be used either by OAuth2 clients or by the Agent I/O module. A client API call may result in messages from the authorisation server to the IAA blockchain agent (c.f. the sequence diagrams below).
2) *OAuth2 Authorisation server.* This module implements the functionality required by the OAuth2 protocol. This includes generating access tokens.

[2] https://github.com/bshaffer/oauth2-server-php

3) *AA mechanisms*. This module implements user authentication and authorisation mechanisms. Currently this module implements password-based and DID-based authentication, and simple authorisation policies.

IoT platform modules

These are IAA modules that have to be implemented by an IoT platform in order to interact with the component. In order to be compatible with the IAA component, an IoT platform should be extended with an "*OAuth2 token processing*" module, which is responsible for validating the OAuth2 tokens sent by the platform clients.

4.3.1 Operations

This subsection describes three key operations of the IAA component: DID creation and registration, Entity Authentication, and User Authorisation.

DID creation and registration

One of the goals of the IAA component is to enable the management of DIDs which are compatible with W3C's draft specifications.[3] In its present form the IAA component relies on an Ethereum smart contract that conforms to ERC-1056,[4] i.e. an Ethereum standard for smart contracts used for creating and updating DIDs. The *DID-client* internal module is responsible for interacting with this smart contract.

User Authentication

The IAA component currently supports two authentication methods: username/password based and DID-based.

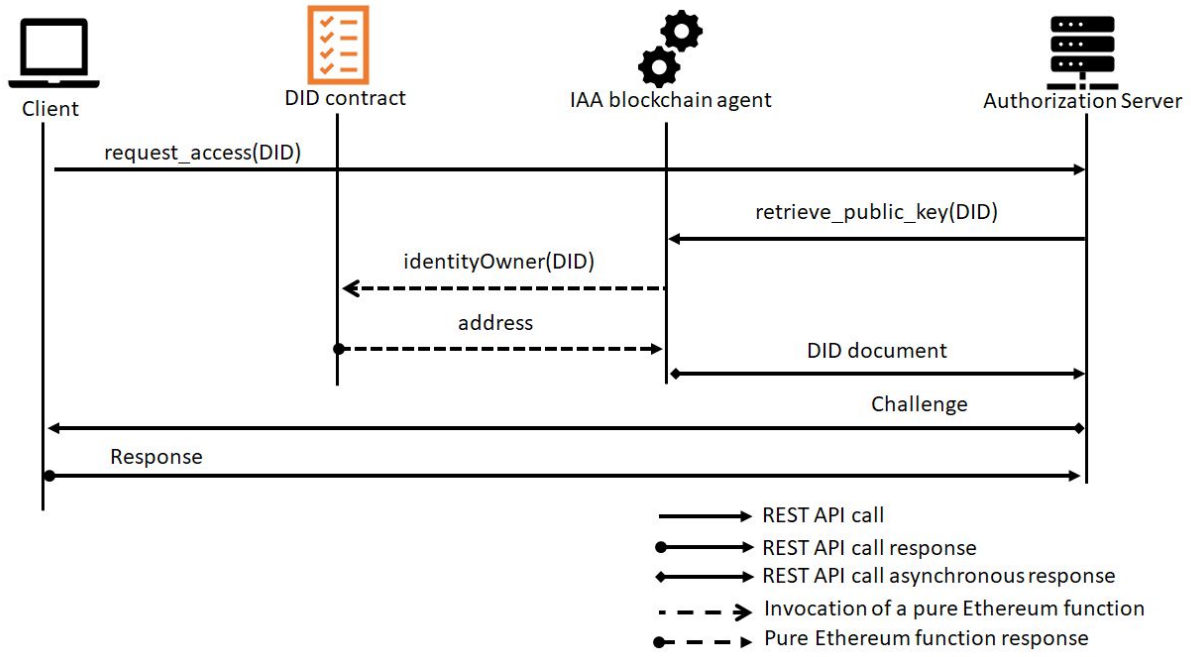

Figure 4.3. DID-based user authentication.

[3] https://w3c-ccg.github.io/did-spec/
[4] https://github.com/ethereum/EIPs/issues/1056

Username/Password based authentication: For this authentication method the component relies on the *Client Credentials* authentication method (section 1.3.4, RFC 6749) as implemented by the OAuth2 php server.

DID-based based authentication: For this authentication method the component relies on DIDs as specified by W3C. In particular, a DID is treated as a key to a (key, value) data structure maintained by a smart contract. The value part of this pair contains a *DID document*, i.e. a JSON-LD[5] encoded document that contains information about a DID, including a public key. Therefore, as shown in Figure 4.3, whenever a client requests access and presents her DID, the authorisation server retrieves the corresponding public key and generates a challenge that the client must digitally sign in order to be considered authenticated.

User Authorisation

The IAA component relies on OAuth 2.0 for user authorisation. A smart contract can be used i) for controlling the decryption of an access token (e.g. the token decryption key is provided only after a user has paid a specific amount of money), or ii) to handle authorisation requests (e.g. in order to provide resilience against DoS attacks).

Model 1: Smart contract implements conditional token decryption key reveal

With this model the initial communication between the client and the authorisation server (AS) occurs as in the normal OAuth 2.0 framework as shown in Figure 4.4. However, instead of the AS providing the client with authorisation credentials after consent is given by the resource owner, the authorisation credentials are disclosed after a specific, pre-defined event (e.g. a payment) is recorded on the blockchain. Hence, the resource owner does not need to be online to provide consent, as in the case of the normal OAuth 2.0 procedure.

Specifically, initially the client requests resource access from the AS over a secure channel. The AS generates a random Proof-of-Possession (PoP) key, which it sends to the client together with its encryption with the secret key[6] *Thing*, shared by the IoT platform and the AS; the client can use this PoP key to establish a secure communication link with the IoT platform. Also, the AS sends to the client the access token encrypted with a secret *s*, $E_s(token)$, the hash $h = Hash(s)$ of the secret *s*, and *a condition for revealing s*. In the current implementation the latter condition is expressed with a *price* that a client should pay, but other forms of conditions (not necessarily only monetary) can be considered. The secret *s* is a one-time secret randomly generated by the AS for each individual request, and is required for the client to decrypt $E_s(token)$ and obtain the access token; the AS will reveal the secret *s* once it confirms that the required condition has been met, i.e. in the current implementation the payment for resource access is performed on the blockchain. The difference with normal OAuth 2.0, in addition to the AS responding immediately to the resource access request without obtaining consent from the resource owner, is that the AS sends the encrypted access token $E_s(token)$ instead of the access token in plaintext. Also, the AS sends the hash *h* and the condition for resource access.

As a next step, two hashes are submitted to the blockchain: the first one is the hash of the token that the AS will reveal once the condition has been satisfied. The second one is the hash of three items: the PoP key encrypted with the secret key the AS shares with the Thing $E_{Thing}(PoP)$, the PoP key, and the encrypted token $E_s(token)$; the second hash serves as proof of the information that is communicated using OAuth between the AS and the client. The two

[5] https://www.w3.org/TR/2014/REC-json-ld-20140116/

[6] The secret key that the IoT platform and AS share is established during a configuration phase, during which the IoT platform is bound to the AS.

hashes immutably record on the blockchain the information that has been exchanged, which can be validated in the case of disputes; however, they do not ensure that the access token the client obtains from the AS indeed allows access to the IoT platform.

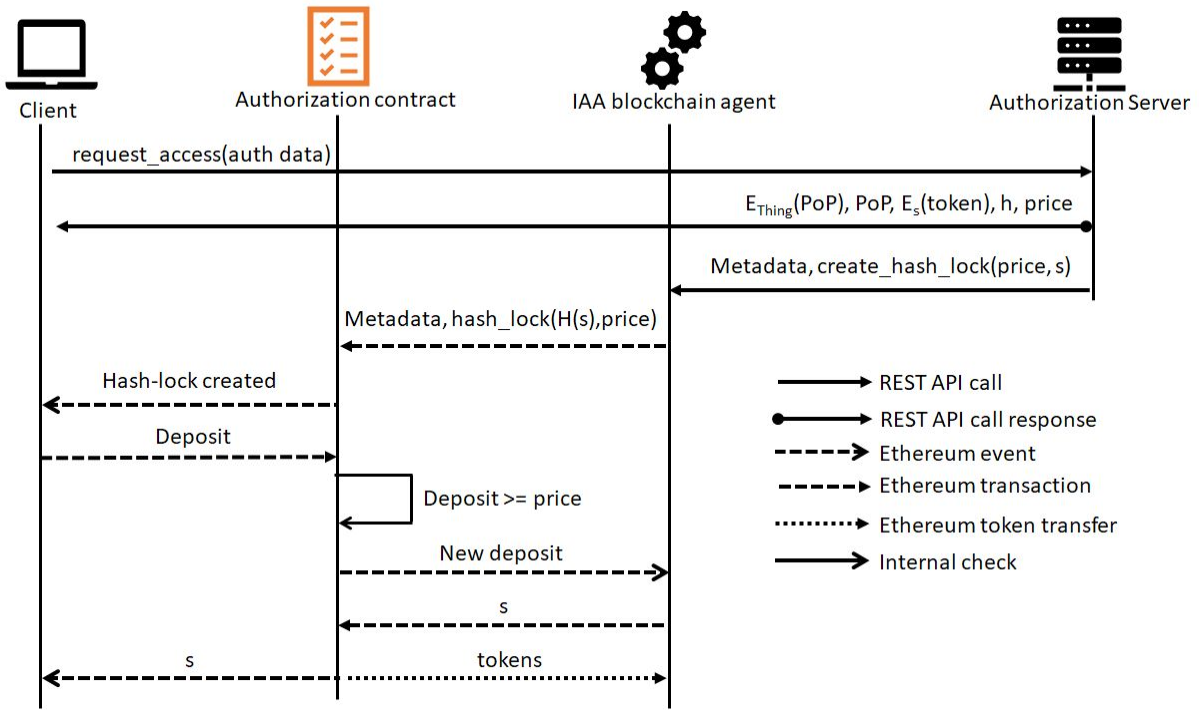

Figure 4.4: User Authorisation model 1: Authorisation grants are linked to blockchain payments and the hash of the information communicated using OAuth is immutably recorded on the blockchain.

Additionally, in this step a hash time-locked contract is created on the blockchain, which allows the client to trigger an event that satisfies the imposed condition (i.e. in the current implementation, to pay the appropriate amount of money). If the hash time-locked involves money/token transfer from the client to another account, as in the current implementation, the transfer takes place iif the secret *s* (hash-lock) is submitted to the contract by the AS within some time interval. If the time interval is exceeded, then the client can request a refund of the amount it deposited. Once the secret *s* is revealed, the client can obtain *s* from the blockchain and decrypt $E_s(token)$, thus obtain the access token. At this point, the client has all the necessary information to request access from the IoT platform.

Model 2: Smart contract handles authorisation requests

In the second model, a smart contract is used to transparently access conditions and other authorisation policies defined by the resource owner, which is also the owner of the smart contract. Examples of such policies include permitting resource access to specific clients determined by their public keys on the blockchain, and dependence of access authorisation on IoT events that are recorded on the blockchain. Whereas in the previous model the client and the AS communicated directly, in this model the interaction is through the smart contract, as shown in Figure 4.5. The smart contract code is executed by all blockchain nodes, providing a secure and reliable execution environment; this provides higher protection against DoS attacks. An additional advantage achieved by allowing a smart contract to handle resource

authorisation requests is that the smart contract can securely bind the protected resource with the AS responsible for handling authorisation requests.

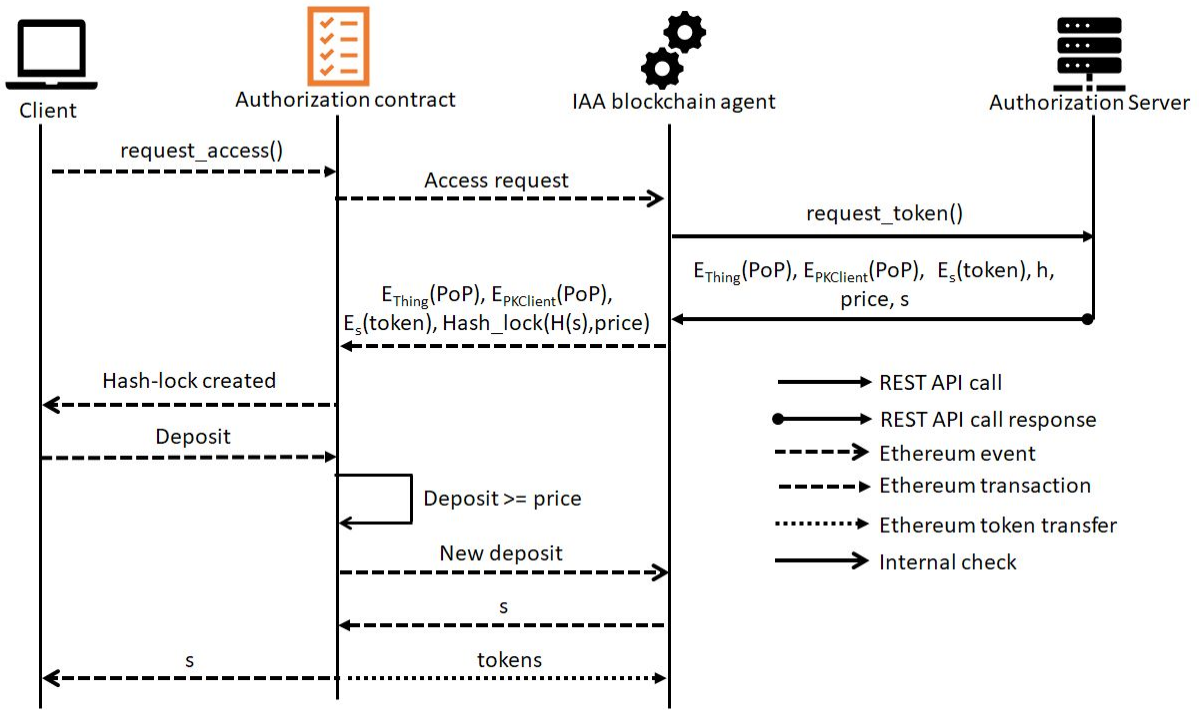

Figure 4.5: User Authorisation model 2: Smart contract handling authorisation requests and encoding authorisation policies

As in the previous model, a hash time-lock is enabled, allowing the client to trigger an event that results in the hash-lock being unlocked by the AS by revealing the secret *s*. Once revealed, the client can obtain the secret *s*, together with the other necessary authorisation information to access the protected resource. If the blockchain is public, then *s* can be read by anyone, hence everyone can obtain the access token. However, the access token cannot be used alone, since the PoP key is also required for accessing the resource. Nevertheless, privacy concerns might require that the token is kept secret; this can be achieved by encoding the token with the client's public key. In this scenario, the AS sends to the smart contract the PoP key encrypted both with the IoT platform's key, $E_{Thing}(PoP)$, and with the client's public key, $E_{PKclient}(PoP)$. Hence, only the IoT platform and the client can obtain the PoP key. On the other hand, in the first model, the PoP key was sent from the AS to the client over a secure communication link, hence its encryption was not necessary

4.4 Scenario walkthrough

The following table describes how the IAA component satisfies the scenarios presented in section 4.1.

Table 4.4. Scenario walkthrough

ID	Scenario Validation	
W01	Description	A producer hands over box(es) containing produce toa transportation employee. [SC01]
	Walkthrough	The IoT resources of this scenario are the boxes that contain the products, and the client is the TR employee. The producer has interacted with the server administration (external) module of the authorisation server and has specified the appropriate access control policies. The TR employee requests access to a box through IF01. The sequence of messages illustrated in Figure 4.4 (or 4.5) takes place. In this scenario there is no payment. Instead, the condition specified by the authorisation server, in order for the TR employee to gain access to the decryption key of the authorisation token, is that the TR employee should 'accept the responsibility of these boxes'.
W02	Description	A smart meter data owner removes the access rights for his data. [SC02]
	Walkthrough	This process is currently not implemented by the IAA component. Instead, the server administration module of the authorisation server, which is an external module, is responsible for that.
W03	Description	A smart meter data owner (owner) configures her identifier. [SC03]
	Walkthrough	Currently, in this scenario we consider DIDs. The application of the smart meter owner interacts with IF01 and specifies her DID. The DID client module is eventually invoked, which interacts with the appropriate functions of the DID contract. After successfully completing the procedure, the client application receives the appropriate event generated by the smart contract and forwarded through the DID client module.
W04	Description	A gamer joins a challenge. [SC04]
	Walkthrough	The IoT resources of this scenario are the game clues and the clients are the gamers. The walkthrough of this scenario is the same as in SC01 with the following two differences: (i) the clients are using diverse mobile platforms (which may be constrained), and (ii) the condition for unlocking the decryption key of the authorisation token is game specific (e.g. the user has enough game credits).

5 Privacy and Data Sovereignty Component

The Privacy and Data Sovereignty component provides mechanisms that allow actors to better control their data, as well as mechanisms that protect clients' privacy.

In its present form, this component extends the IAA component in the following ways: (i) it enables flexible authorisation server delegation, and (ii) it enables client authentication and authorisation using *verifiable credentials* (VCs). This component is based on research work published in [Fot2018] and in [Lag2019].

5.1 Requirements and Scenarios

Table 5.1 summarises the major requirements for this component

Table 5.1. Requirements for the IAA component

ID	Requirement Description	Priority	Category
RF10	SOFIE must follow the data minimisation principle for personal data and only request or process what is necessary for the situation and purpose.	MUST	OPERA-TIONAL
RF11	Processing of individual's personal data must require a valid consent from the individual.	MUST	POLICY & REGULATION
RF12	Consent of the actors to process their private data must be revocable at any time.	MUST	POLICY & REGULATION
RF13	SOFIE must allow organisations and actors to manage (create, update, delete) their own data privacy policies.	MUST	POLICY & REGULATION
RF14	SOFIE should support user privacy even when aggregate statistics are made public (e.g. using differential privacy mechanisms).	SHOULD	POLICY & REGULATION

Table 5.2 presents example scenarios derived from use cases in SOFIE deliverable D5.2 to cover all requirements from the previous subsection.

Table 5.2. Scenarios derived from the pilots use cases

ID	Scenario Content	
SC01	Description	A smart meter data owner configures the component with policies regulating access to his/her smart meter data and accessing rights. Later on, she modifies them.
	Stimulus	The data owner interacts with the appropriate interfaces and manages her own policies
	Response	Access control policies created/modified/deleted.

	Derived from use case	DEDE_UC1
	Covers requirements	RF13
SC02	Description	An energy services provider, or a smart meter system operator requests access rights for some data from their owner.
	Stimulus	The interested entity authenticates itself and requests authorisation for accessing some particular data.
	Response	The data owner consents or denies access to her data. The appropriate verifiable credentials are constructed and delivered to the interested party.
	Derived from use case	DEDE_UC3
	Covers requirements	RF11
SC03	Description	An authorized service provider starts downloading some smart meter data and uses them based on an agreed contract.
	Stimulus	The energy service provider authenticates itself by presenting the appropriate verifiable credentials and requests data.
	Response	Access granted and data start to flow, or access denied.
	Derived from use case	DEDE_UC2
	Covers requirements	RF10, RF14
SC04	Description	The data owner revokes an actor’s authorisation to access her data.
	Stimulus	The data owner interacts with the appropriate interface, authenticates herself, and requests the revocation of some specific verifiable credentials.
	Response	The “verifiable credentials” are revoked, or operation not permitted (e.g. an owner tries to revoke some credentials not issued by her).
	Derived from use case	DEDE_UC4
	Covers requirements	RF12

5.2 Services and Interfaces

The Privacy and Data Sovereignty component includes the same interfaces as the IAA component, extended to support the additional functionalities offered. In particular the interfaces of the IAA component are extended as follows:

- **Client functions:** This interface is extended to handle VC management.
- **Agent I/O:** This interface is extended to support interactions with a Hyperledger Indy pool.
- **AA API:** This interface is extended to handle client authentication using VCs.

Table 5.3. Interfaces of the component

ID	Interface Content	
IF01	Name	**Client functions**
	Description	It is used by client applications for accessing IoT platforms, for registering a DID, and for managing VCs.
	Key inputs	Platform identifier, access method, client authentication data.
	Response	Access token, delegation token.
IF02	Name	**Agent I/O**
	Description	The interface of the “PDS blockchain agent” used for communicating with DLTs.
	Key inputs	Operation, metadata.
	Response	Operation output, blockchain events.
IF03	Name	**AA API**
	Description	The REST-based interface of the Authorisation server used for client authentication and authorisation.
	Key inputs	User authentication data, delegation information.
	Response	Access token, metadata.

5.3 The internal structure

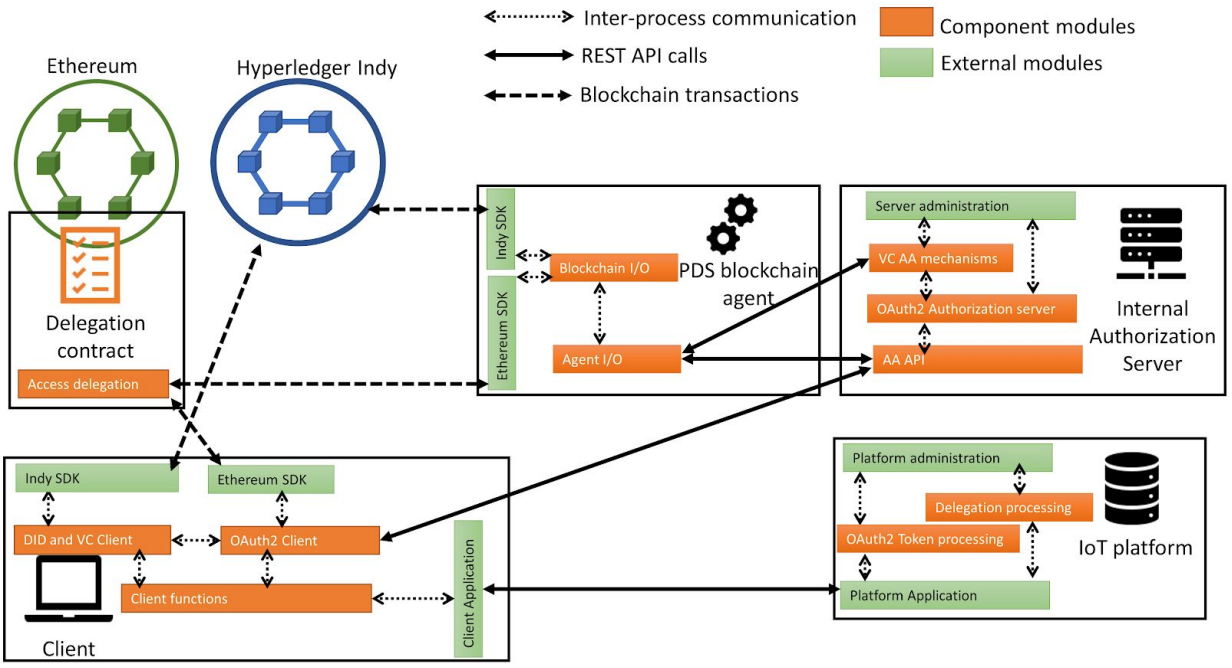

Figure 5.1: Privacy and Data Sovereignty component *internal structure*

The Privacy and Data Sovereignty component is composed of the following 6 entities:

Client modules
These are in essence the same modules as in the IAA component, extended with an additional module that adds support for verifiable credentials.

Smart contracts
The Privacy and Data Sovereignty component currently includes a single Ethereum smart contract which manages the "access control delegation" functionality of the component. The principal operation of this smart contract is that it allows resource owners to register the addresses of the authorisation servers that are responsible for managing their resources.

DID and VC management system
DIDs and VC credentials are managed by an instance of Hyperledger Indy (usually referred to as a *pool*). The component can be configured to be used with any Indy pool, i.e. a different installation of Hyperledger Indy managed by different trusted nodes.

PDS blockchain agent
Similarly to the IAA component, this component includes a blockchain agent entity that mediates the communication between the authorisation server (see next), the authorisation delegation smart contract, **and the Hyperledger Indy pool.** This entity also includes a *Blockchain I/O* module and an *Agent I/O* module.

Authorisation server
The authorisation server of this component is the same as in the server of the IAA component, but it includes an enhanced *AA mechanisms* module. The enhanced module includes (i)

mechanisms for enabling VC-based authorisation, and (ii) mechanisms for handling authorisation delegation.

IoT platform modules
These are the modules used by the IAA component, enhanced to support the functionalities required for delegating authorisation (see next for more details).

5.3.1 Operations

This subsection describes three key operations of the Privacy & Data Sovereignty component: User authentication using VCs, Access delegation setup, and User authorisation with delegation enabled.

User authentication using VCs
In order to enhance the clients' privacy, this module adds support for user authentication based on VCs. This process is implemented as follows (see [Lag2019] for more details):

- The client requests authorisation from the AS to access a protected resource.
- The AS examines the access control policy that protects the requested resource, determines the *credentials* that the client should have and generates a *proof request.*
- The client creates the appropriate proof and sends it back to the AS.

The client's privacy is enhanced in the sense that: (i) the AS learns the minimum information required in order for the client to get access, and (ii) certain client credentials can be proven using zero-knowledge proofs (e.g. a client may prove that she is over 18 years old without revealing her actual age).

Access delegation setup
Access delegation enables resource owners to delegate access control decisions to a selected AS, which becomes responsible for authorising clients to access a protected resource.The client authorisation operations described in the following assume a setup phase. During this phase, the delegation smart contract is configured with the available resource identifiers (referred to as $URI_{resource}$), the corresponding pointers to access control policies, referred to as URI_{policy}, and the public keys of the authorisation servers that manages each policy, referred to P_{AS}. For simplicity of presentation it is assumed that each resource is protected by a single URI_{policy} provided by a single P_{AS}.

User authorisation with delegation enabled
This operation can be implemented using two models that both extend a 'straw man approach' by enhancing its security properties.

The strawman approach: In this approach, initially a client P_{client} requests a protected resource from an IoT platform and the platform responds with a *delegation token* ($token_d$) and the URI of the delegation smart contract. Then, the second model for user authorisation, described in section 4.3.1 is used, with the following additions: the request_access() method is now implemented by the delegation contract, the client includes $token_d$ when invoking request_access(), and the AS instead of encrypting the PoP key with the key shared between the AS and IoT platform, calculates a new key which is the HMAC of $token_d$ using the shared key as the HMAC key. The latter key is referred to as *ThingSK*.

Model 1. Platform Identity verified by the authorisation server

One way of improving the straw man approach is by allowing an AS to verify that a client is communicating with a legitimate IoT platform (Figure 5.2). In order to achieve this goal, the request_access() method of the smart contract is extended to include an additional field, i.e. $H_{ThingSK}(token_d)$. The value for this field is provided by the IoT platform, in its response to a client request. Now an AS, after generating the *ThingSK*, calculates $H_{ThingSK}(token_d)$, and checks if the value of the latter calculation is equal to the value provided by the platform. If this is true, then the platform is considered legitimate.

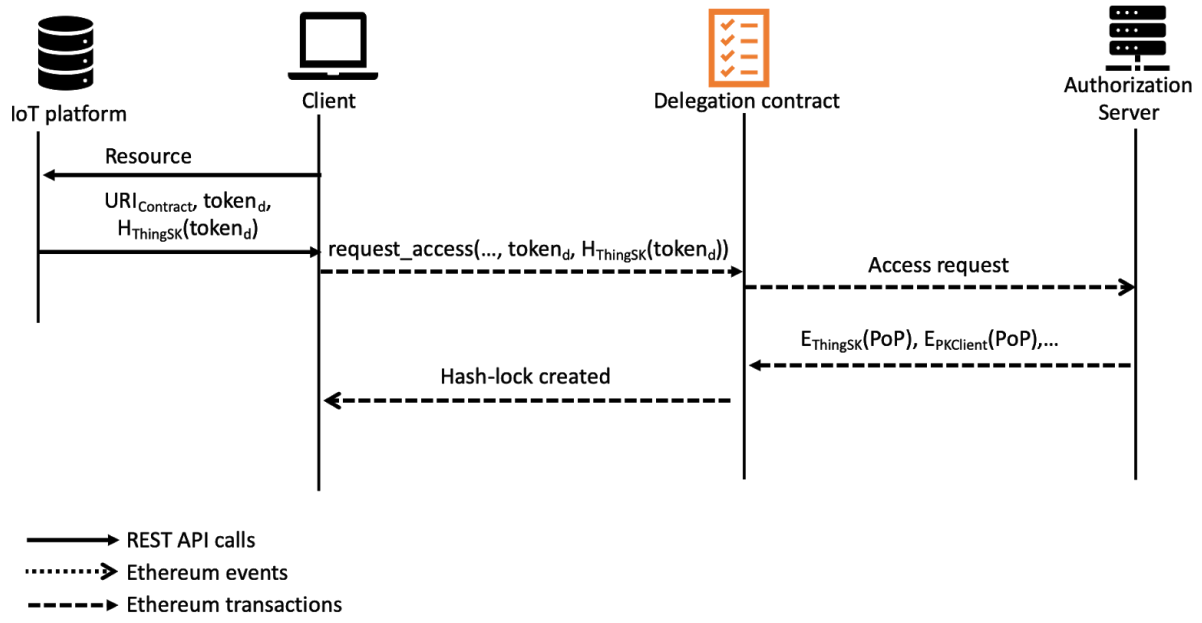

Figure 5.2: Client authorisation with delegation enabled model 1. The authorisation server verifies the identity of the IoT platform (the PDS blockchain agent module is omitted for brevity).

Model 2. AS-platform trust relationship verified by the smart contract

The first model can be further extended to enable the delegation smart contract to verify the relationship between an IoT platform and an AS, i.e. the contract can verify that the platform and the AS indeed share a secret key (Figure 5.3). This functionality is achieved by having the client "challenge" the platform during her request. The challenge used is a random number, which the platform should obfuscate in a way that only an AS that shares a secret key with the platform could read. The smart contract should therefore learn the challenge from the client and should expect it from the AS. In order to "hide" the challenge a hash function is leveraged using the process described below.

The platform responds to a challenge with $H(H_{ThingSK}(challenge))$. Given a challenge, only an entity that can generate *ThingSK* can calculate $H_{ThingSK}(challenge)$. Note that, in addition to the platform, this key can be calculated by the AS that protects the resources stored in that platform. Furthermore, given $H_{ThingSK}(challenge)$ any entity, including the delegation smart contract, can easily calculate $H(H_{ThingSK}(challenge))$ (but the reverse process is not possible due to the properties of the hash functions). Hence, the request_access() method is extended to include $H(H_{ThingSK}(challenge))$ and the authorize() method is extended to include $H_{ThingSK}(challenge)$. Then, the smart contract can calculate the hash of the $H_{ThingSK}(challenge)$,

received by the AS, and compare the output with the hash value it received from the client. If both hash outputs are the same, the contract proceeds with the subsequent steps.

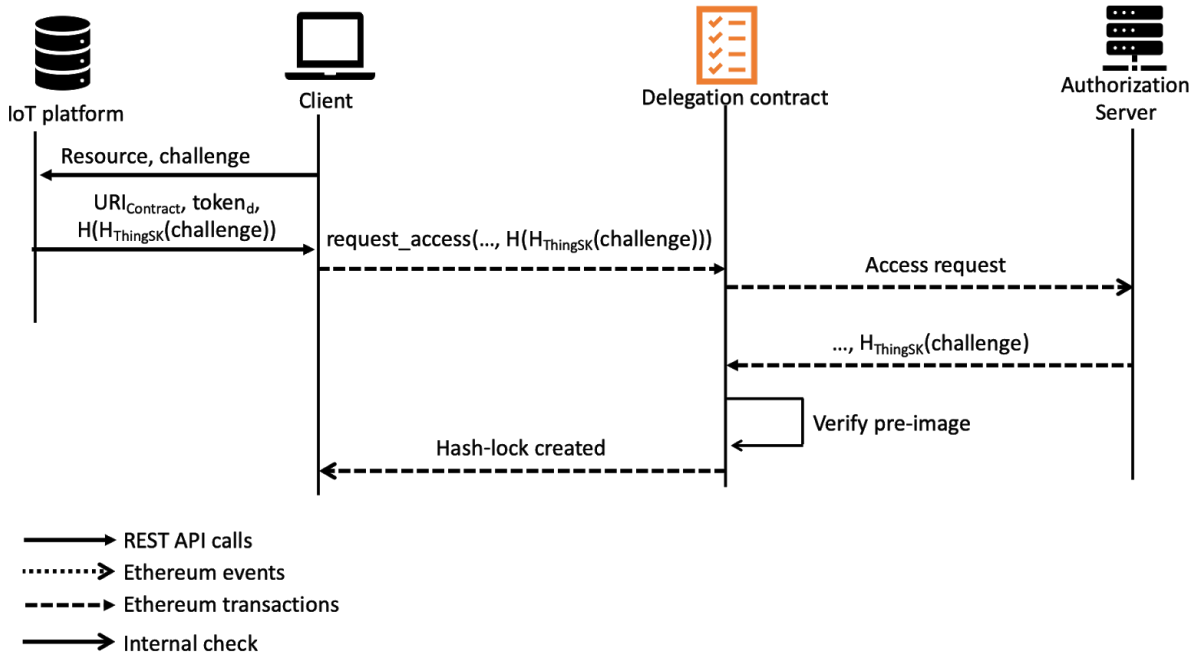

Figure 5.3: Client authorisation with delegation enabled model 2. The delegation contract verifies the trust relationship between the AS and the IoT platform (the PDS blockchain agent module is omitted for brevity).

5.4 Scenario walkthrough

The following table describes how the Privacy and Data Sovereignty component satisfies the scenarios presented in section 5.1

Table 5.4. Scenario walkthrough

ID	Scenario Validation	
W01	Description	A smart meter data owner configures the component with policies regulating access to his/her smart meter data and access rights. Later on, she modifies them. [SC01]
	Walkthrough	This process is currently not implemented by the Privacy and Data Sovereignty component; instead, the “server administration” module of the authorisation server, which is an external module, is responsible for that.
W02	Description	An energy services provider, or a smart meter system operator requests access rights for some data by their owner. [SC02]
	Walkthrough	The smart meter of this scenario holds the role of the IoT resource and the energy service provider (or the smart meter system operator)

		holds the role of the client. The sequence of messages illustrated in Figure 5.3 (or 5.4) takes place. Requirement RF14 is not currently satisfied; in the next version of the component, this requirement will be satisfied using *differential privacy* mechanisms.
W03	Description	An energy service provider is interested in providing energy service to a smart meter data owner and needs access to the energy consumption data. After access rights are given, the provider can start downloading the data and use this data to fulfill the contract. [SC03]
	Walkthrough	Data owners, currently, can issue VCs for any entity requesting access by interacting with IF02 (i.e. the Agent I/O interface). VCs are issued by following Hyperledger Indy's procedure. Then these VCs can be used by the authorised entity (i.e. the energy service provider in this scenario) in order to access the desired data. This is implemented using the operations described in section 5.3.1.
W04	Description	The data owner revokes an actor's authorisation to access her data. [SC04]
	Walkthrough	Currently, data owners can revoke authorisation by revoking issued VCs. This can be done by interacting with IF02 (i.e. the Agent I/O interface) and by following Hyperledger Indy's procedures.

6 Semantic Representation Component

Semantic representation (SR) is a mechanism for describing the data model and the services of IoT devices. It defines a common representation model for IoT Things devices, their services and their data, which enables interoperability and automation in the deployment of services and applications on top of federated IoT environments.

The needs of this mechanism can be easily understood by considering the food supply chain pilot. In that context, a stakeholder could join the supply chain with its own IoT system. The new stakeholder's IoT system has a different data model and services compared to the one already in place in the FSC. This new stakeholder will use the semantic representation to translate their data model to one compatible with the SOFIE system, enabling the communication between their IoT system and the existing one.

6.1 Requirements and Scenarios

Table 6.1 summarises the requirements for this component (from SOFIE deliverable D2.4).

Table 6.1. Requirements for the Semantic Representation component

ID	Requirement Description	Priority	Category
RF15	The system must define an IoT things description model which is based on well-known standards (e.g. W3C standards).	MUST	AUDITA-BILITY
RF16	The system must implement standardised metadata and data representation formats and support various data modalities.	MUST	AUDITA-BILITY
RF17	The semantic representation model of the system must be open and extensible by third parties (e.g. support the extension of the existing knowledge base and associations by extracting supplementary triples from RDF documents).	MUST	AUDITA-BILITY
RF18	The system must provide service discovery and resources selection processes based on multiple-criteria over features, associations and interaction patterns of integrated resources.	MUST	INTEROPE-RABILITY
RF19	The system should support the semantic update and enhancement of resources' descriptions and associations in a dynamic way.	SHOULD	INTEROPE-RABILITY

Table 6.2 presents example scenarios, derived from the use cases presented in SOFIE deliverable D5.2, to cover all requirements described above.

Table 6.2 scenarios derived from the pilot use cases

ID	Scenario Content	
SC01	Description	A new participant joins the food supply chain. She has to enable the data interaction between her IoT silo and the existing system.
	Stimulus	The new participant implements a Federation Adapter using the Semantic Representation data models.
	Response	The actors of the FSC system can share information with the new participant and vice versa.
	Derived from use case	None
	Covers requirements	RF15, RF16
SC02	Description	During gameplay, a player's smartphone automatically discovers any new BLE beacons. The smartphone reads the BLE beacon information and sends it to the gaming company's server, which can then review the eligibility of the beacon to be included in future game challenges. The game administrator needs to check, e.g. whether the device can be utilised by the user and does this based on the license which is included in the BLE beacon information.
	Stimulus	The player is playing the game and the device automatically scans for new BLE beacons.
	Response	Rovio obtains the information about the BLE beacon.
	Derived from use case	MRMG _UC5
	Covers requirements	RF18
SC03	Description	A partner of the food supply chain updates her IoT system with new functionalities which are not supported by the current SOFIE system Semantic Representation.
	Stimulus	The partner implements an extended semantic representation module.
	Response	The participants in the system can exchange data with the partner's IoT system.
	Derived from use case	None
	Covers requirements	RF17, RF19

6.2 The internal structure

The Semantic Representation is not a separate software module with a defined internal structure and a set of APIs which can be called by other elements in the system. Rather, it is a logical component that can be implemented as part of other components to define the data and service models of an IoT system.

The SR can vary between different instances of the SOFIE framework because there is no universal semantic representation for everything. The SR can be derived from well known standards (e.g. W3C WoT, Fiware) or developed adhoc for a project.

The pilots use Web of Thing (WoT) standards to enable device interoperability. “WoT provides mechanisms to formally describe IoT interfaces to allow IoT devices and services to communicate with each other, independent of their underlying implementation, and across multiple networking protocols. In addition, WoT offers a standardised way to define and program IoT behaviour.” [Kov2019] However, the pilots do not implement all the WoT components, but focus only on the WoT’s Thing Description (TD). “TD is the central building block, as it allows to describe the metadata and network-facing interfaces of Things. The WoT Thing Description specification defines an information model based on a semantic vocabulary and a serialised representation based on JSON. TDs provide rich metadata for Things in a way that is both human-readable and machine-understandable. For semantic interoperability, TDs may make use of domain-specific vocabulary, for which explicit extension points are provided.” [Kov2019]

Thing Description
The WoT Thing Description (TD) is a central building block in the W3C Web of Things (WoT) and can be considered as the entry point of a Thing (much like the index.html of a Web site).

A TD instance has four main components:

1. *Textual metadata* about the Thing itself,
2. *Interaction Affordances* that indicate how the Thing can be used and how actors can interact with the Thing.
3. *Schemas* for the machine-readable data exchanged with the Thing, describing the data format used.
4. *Links* that express any formal or informal relation with other Things or documents on the Web.

WoT standards also enable Thing Description extensions through the TD Context Extension. For such TD Context Extensions, the Thing Descriptions use the @context mechanism known from JSON-LD. When using TD Context Extensions, the value of @context of the Class Thing is an Array with additional elements.

As an example, Figure 6.1 presents a TD from the FSC pilot and highlights the different components and extensions.

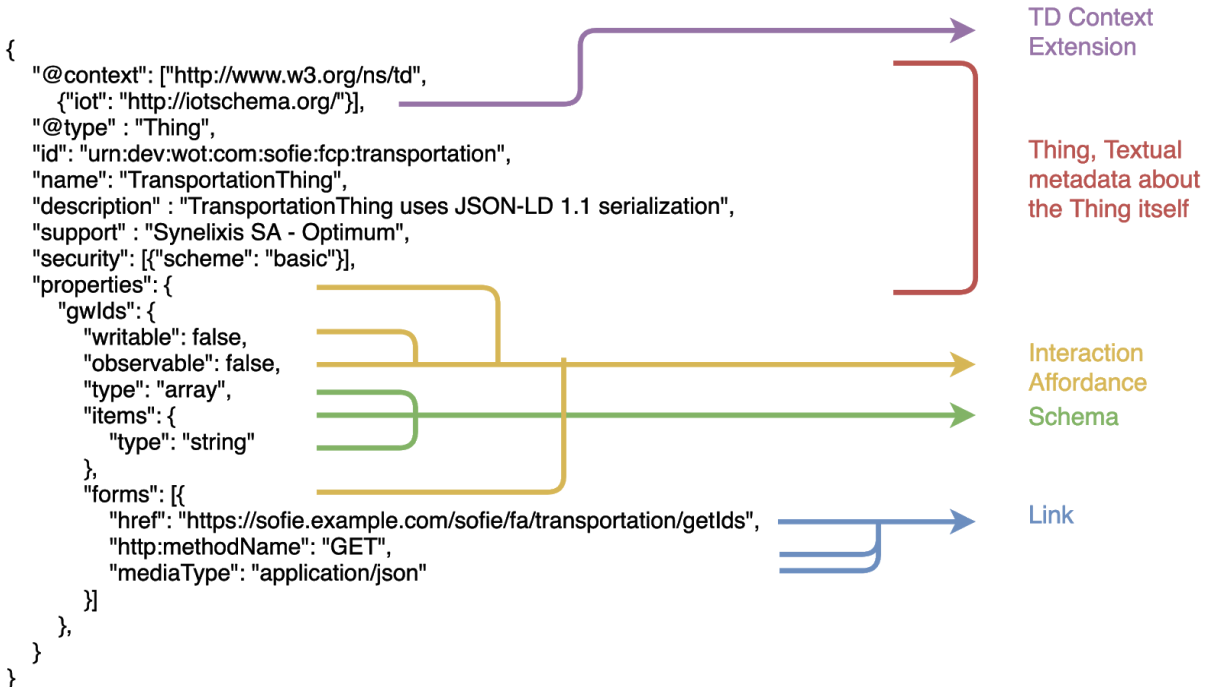

Figure 6.1: Things description (TD) of the food supply chain pilot

6.3 Scenario walkthrough

The following table describes how the Semantic Representation component satisfies the scenarios presented in section 6.1

Table 6.4. Scenario walkthrough

ID	Scenario validation	
W01	Description	A new participant joins the food supply chain. [SC01]
	Walkthrough	The new participant implements the Federation Adapter (FA) following the SR logic. The FA translates the data model of the new participant's IoT silos to a data model compatible with the WoT TD, enabling interoperability between the new IoT silos and the existing systems.
W02	Description	The player's smartphone discovers new beacons. [SC02]
	Walkthrough	Rovio shares the WoT TD-based SR required for all beacons to be included in the game. The partner creates a file according to the Rovio's TD, and the file is deployed in the BLE beacon. When the smartphone connects to a new beacon, the file is retrieved by the Provisioning and Discovery component. The player's smartphone then forwards the TD file to the gaming company's server, which reads the TD and check the license and devices fields. If the TD passes the checks, the BLE beacon device is added to the game.

W03	Description	A partner of the FSC updates her IoT system with new functionality. [SC03]
	Walkthrough	The partner creates an extension of the TD through the TD Context Extension mechanism. The extension supports the new functionalities of the partner's IoT system. The partner updates the WoT TD of the IoT system linking the new TD extension in the @context field. The participants of the SOFIE system can then manage the new functionalities of this partner IoT system by reading the context field of the TD and retrieving the extension to the TD.

7 Marketplace Component

The goal of the SOFIE marketplace component is to enable the trade of different types of assets (e.g. electricity for charging a vehicle) in an automated, decentralised, and flexible way. In this context, a decentralised marketplace has the capability of operating without a single entity owning or managing it, which in turn increases competition and enhances its security, resiliency, transparency, and traceability. The decentralised marketplace can be either partially decentralised, when, e.g. a group of independent agriculture producers and retailers are managing it, or fully decentralised, where anyone can join and use the marketplace.

The actors (buyers and sellers) on the marketplace should be able to agree on the terms of the trade, perform payments, and verify that the trade has been carried out successfully with as little user interaction as possible. The marketplace must also provide auditability to help with potential dispute resolutions.

Figure 7.1 shows a few examples of the marketplace component interacting with other components and ledgers. The marketplace interacts with the ledgers directly, unless it has to perform atomic operations involving multiple ledgers, in which case it uses the Interledger component. For instance, in the location based game pilot the marketplace component directly interacts with both the trade and asset ledgers to get information, and uses the interledger component to execute the trade. In the energy flexibility pilot the marketplace component again interacts with the asset and payment ledgers, and relies on an Oracle to report on the charging activity as explained in Section 3.1.

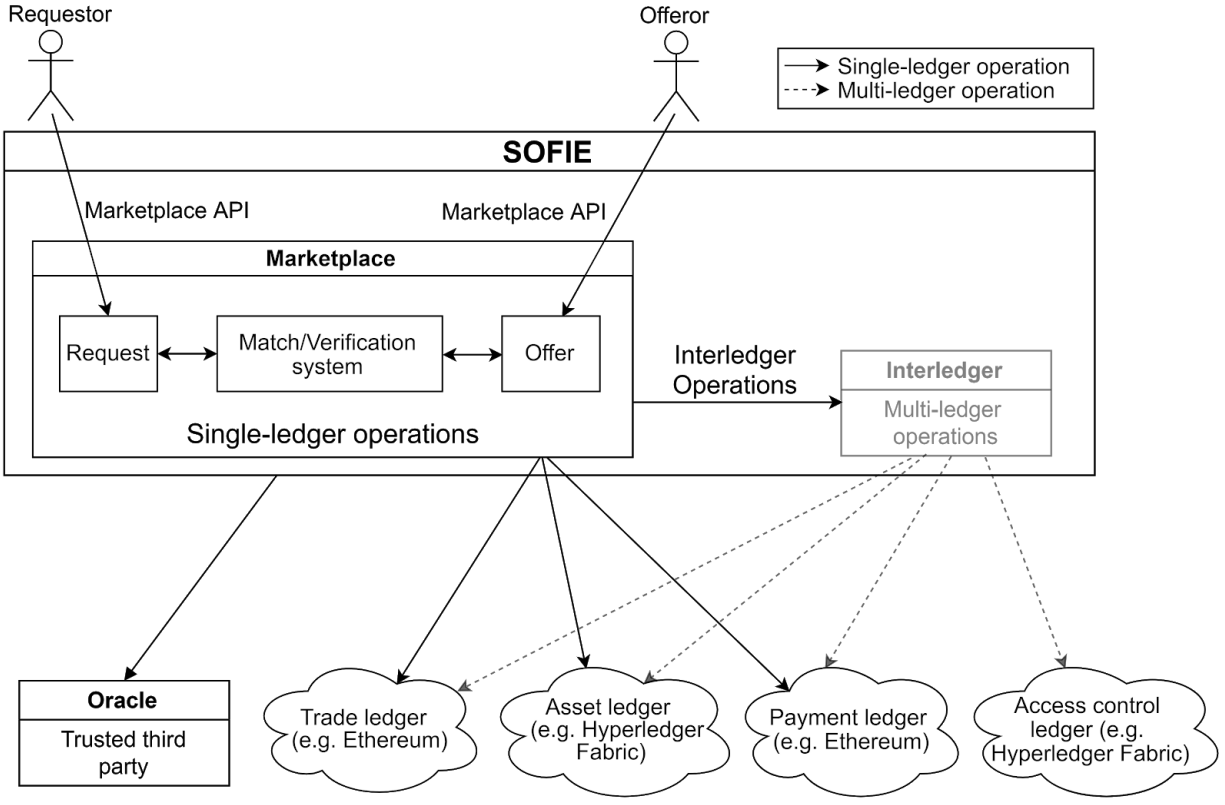

Figure 7.1: The marketplace component and its communication with other components and ledgers.

7.1 Requirements and Scenarios

Table 7.1 summarises the requirements for this component (from SOFIE deliverable D2.4).

Table 7.1. Requirements for the Marketplace component

Req. ID	Requirement Description	Priority	Category
RF20	The marketplace must log the configuration of all trading actions (including offers, bids, parameters of resources, transactions, etc.).	MUST	QUALITY
RF21	The marketplace must provide actors the capability to post/claim offers and sell/negotiate/exchange/buy resources and digital objects.	MUST	INTEROPE-RABILITY
RF22	The marketplace must support transparent trading of resources, i.e. the bid/offer matching process and the payments must be transparent.	MUST	OPERA-TIONAL
RF23	The marketplace must provide evidence once trades have been completed and resources have been properly delivered to the buyers.	MUST	SECURITY
RF24	The marketplace should allow integration of payment technologies.	SHOULD	OPERA-TIONAL

Table 7.2 presents example scenarios derived from use cases in SOFIE deliverable D5.2 to cover all requirements from the previous subsection.

Table 7.2. Scenarios derived from the pilots use cases

ID	Scenario Content	
SC01	Description	To balance the load on the electricity grid, the DSO offers incentives for EV users to charge their vehicles at specific times and locations. The DSO publishes these flexibility requests on the marketplace, stating that a certain amount of energy needs to be used at a certain time and location. EV users can offer bids stating that they can fulfill the request for a specified amount of incentives. Trading agreements (matching bids and offers), verification of trade (both parties have carried out their obligations), as well as compensation after the trade has been completed should all happen as automatically as possible. Finally, for accountability and auditability purposes, all trading-related actions should be logged.
	Stimulus	A party that wants to trade assets (either buyer or seller).

	Response	Bids and offers have been matched, and after the charging has taken place, the EV user is properly compensated. All above mentioned actions have been logged.
	Derived from use case	DEFM_UC1, DEFM_UC2, DEFM_UC3, DEFM_UC8, DEFM_UC9, MRMG_UC4, MRMG_UC7
	Covers requirements	RF20, RF21, RF22, RF23, RF24

7.2 Services and Interfaces

Figure 7.2 presents a marketplace that communicates with smart contracts and stores the data both in the database and on the blockchain. Customers can use this marketplace with a known URL or with a web application.

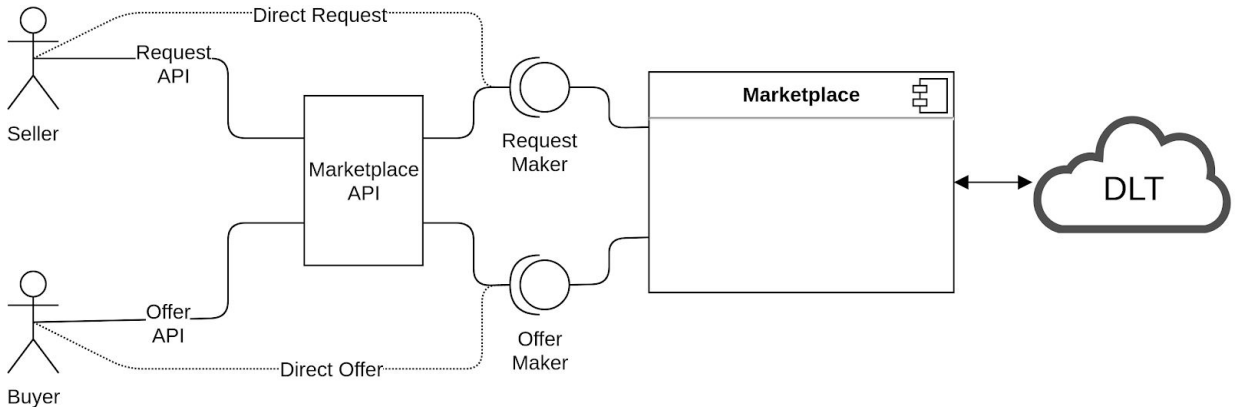

Figure 7.2: The Marketplace component and its interfaces

The Marketplace component offers two interfaces: Request Maker for sellers to create, manage and conclude auctions, and Offer Maker for buyers to participate and bid in auctions. The interfaces are described in Table 7.3.

Table 7.3. Interfaces of the component

ID	Interface Content	
IF01	Name	**Request maker**
	Description	This interface has several functions for creating and submitting requests (to sell assets), for the matching process (of requests and offers), and for getting information about a request and its offers.
	Key inputs	Request ID
	Response	It creates requests, returns their information, matches them with offers automatically, and logs the actions.
IF02	Name	**Offer maker**
	Description	This interface has several functions for creating and submitting offers (to buy assets) and getting information about an offer.
	Key inputs	Request ID, Offer ID
	Response	It creates offers, returns their information, and logs the actions.

7.3 The internal structure

The marketplace is used for trading assets (e.g. energy, digital assets, resource access, etc.). Figure 7.3 presents the internal structure of the marketplace component. Marketplace module includes functionality to communicate with marketplace smart contracts (which are shown in dotted line). Marketplace interface smart contract includes offer maker and request maker interfaces. Marketplace base includes all of base functionalities for the marketplace component, while Ethereum standards includes the standard Ethereum tokens like ERC20[7].

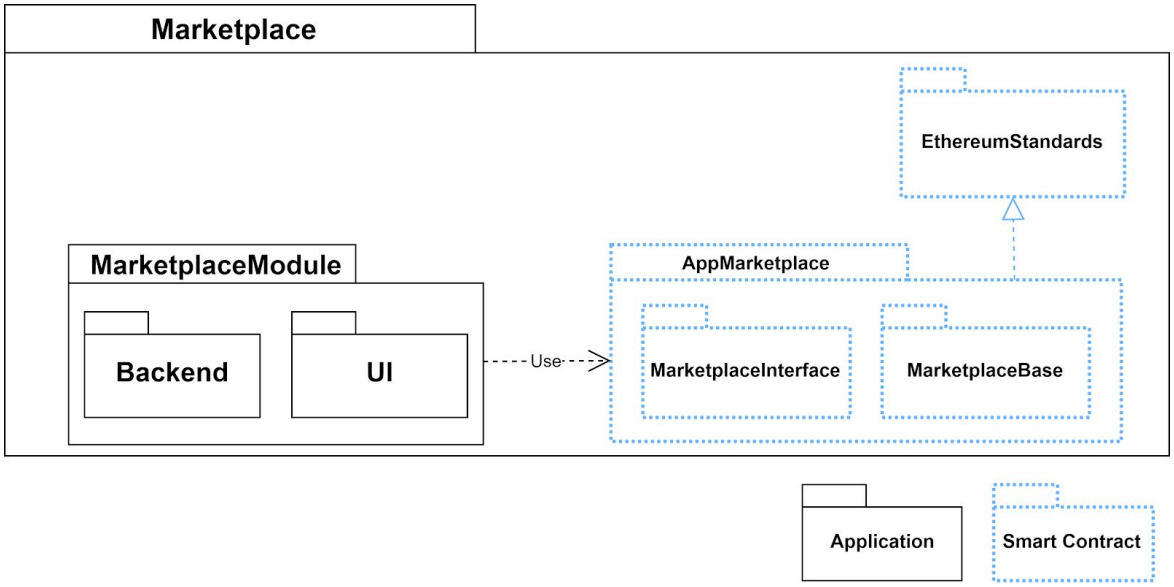

Figure 7.3: The package class diagram of the marketplace component

[7] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

Three key functionalities of the marketplace component, Request creation, Offer creation, and Decision process are described next.

Request creation

For trading assets, request creation, depicted in Figure 7.4, is the first step. The manager of an asset creates a request to sell his assets, and customers can then make offers based on his request.

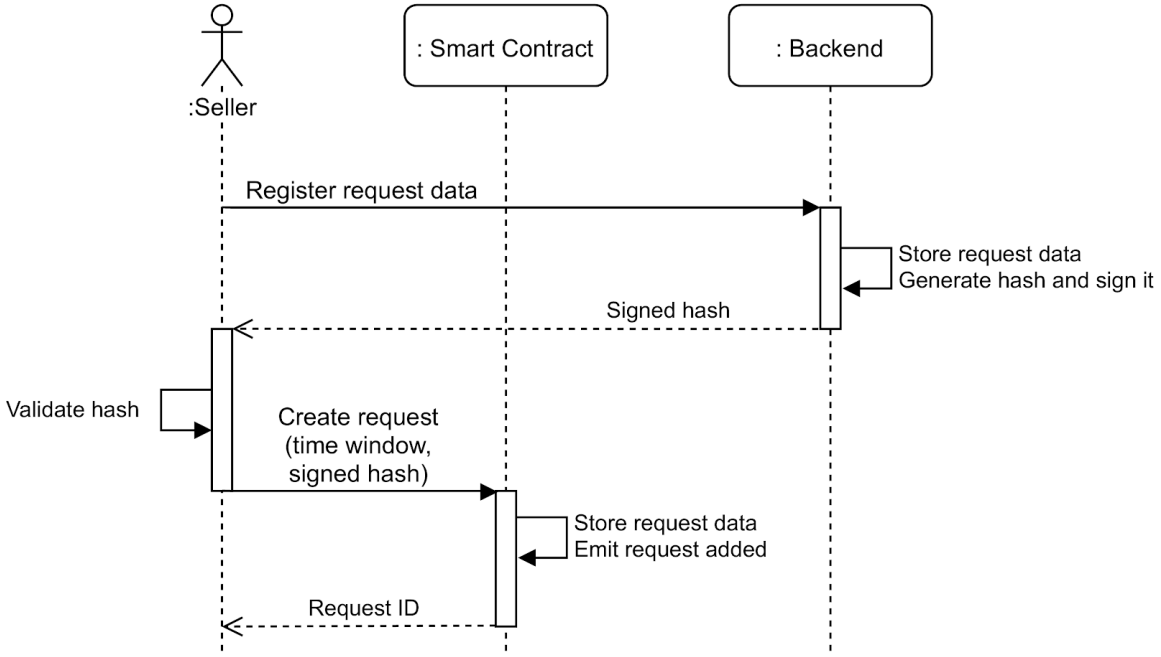

Figure 7.4: Request creation sequence diagram

Offer creation

Once a request has been created, customers can then submit offers on the request. Figure 7.5 shows how an offer can be created.

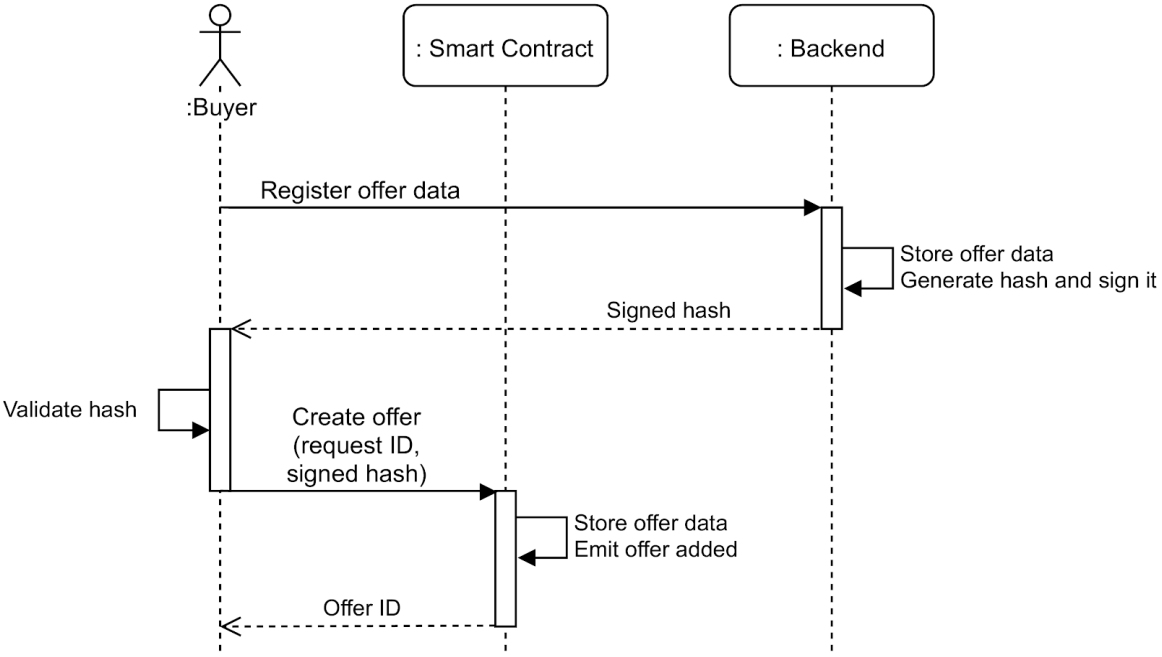

Figure 7.5: Offer creation sequence diagram

Decision process

When the deadline for offers has passed or the requestor wants to determine the result, the decision process should be run to choose the winning offer. Figure 7.6 shows how the decision process happens.

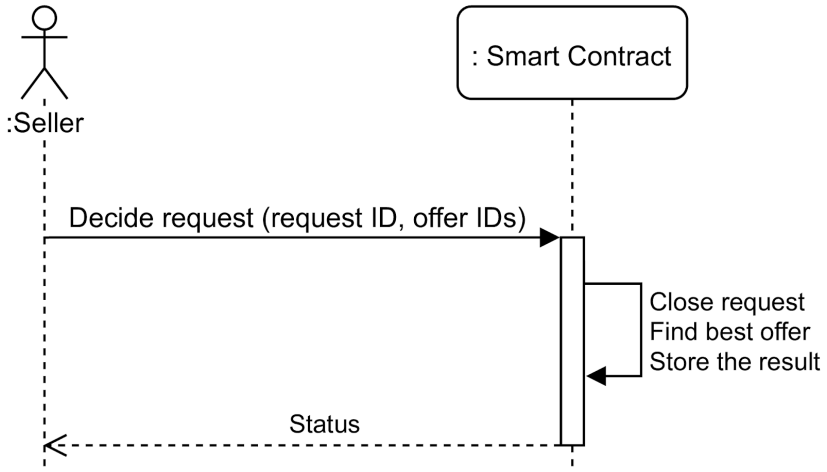

Figure 7.6: Decision process sequence diagram

The current version of the smart contract implements an auction mechanism, in which the best offer is selected following the "lowest bidder" rule. In the future, the smart contract may be upgraded to consider alternative ways to select the winning offer, e.g. in addition to the price also taking other features of the offer into account. In the case of balancing the load on the electricity grid, these can include, e.g. more fine-grained promises to use energy during a specific time.

7.4 Scenario walkthrough

The following table describes how the Semantic Representation component satisfies the scenarios presented in section 7.1.

Table 7.4. Scenario walkthrough

ID	Scenario Validation	
W01	Description	Trading assets is carried out by the bidding system. [SC01]
	Walkthrough	To start trading, the owner initiates a request for assets that he wants to sell and submits it by calling the request creation and submission function from IF01. The function emits an event on the blockchain, allowing interested parties to notice the request. Buyers initiate offers for the request and submit them by calling the offer creation and submission function from IF02. The request is closed automatically after the deadline has passed or manually by the seller. The matching process function is called and the accepted offer for the request is determined. Then, the seller and the winning buyer carry out their obligations. Finally, all information is recorded on the blockchain.

8 Provisioning and Discovery Component

The goal of the provisioning and discovery component is to enable the discovery of new IoT resources and their related metadata. Using this functionality, it is possible to decentralise the process of making new resources available to systems utilising the SOFIE framework and to automate the negotiations for the terms of use and the compensation for the use of these resources. As an example, a location-based game can discover new IoT devices usable for expanding the game world and automatically add them to the resource database, if the resources are accompanied with the necessary metadata including the licence for using the device and the terms of compensation.

8.1 Scenarios

Table 8.1 summarises the requirements for provisioning and discovery component from SOFIE deliverable D2.4.

Table 8.1. Requirements related to the provisioning and discovery component

Req. ID	Requirement Description	Priority	Category
RF18	SOFIE must provide service discovery and resources selection processes based on multiple-criteria over the features, associations and interaction patterns of integrated resources.	MUST	INTEROPER-ABILITY

Table 8.2 presents example scenarios derived from use cases in SOFIE deliverable D5.2.

Table 8.2: Scenarios derived from the pilot use cases

ID	Scenario Content	
SC01	Description	The scavenger hunt location-based game uses IoT beacons to provide the proximity location of the players when they visit Points of Interest (PoIs). These beacons communicate with the player's smartphone using Bluetooth Low Energy (BLE). Upon arriving to a PoI, the mobile application detects the BLE beacon and notifies the player with the relevant task. However, the mobile also detects any new beacons present in the area, and notifies the game server so they can be added to the repository (if they meet the necessary requirements) and later used as PoIs in the game. A list of known and previously rejected beacons prevents the re-submissions of the same beacon.
	Derived from use case	MRMG_UC1 (Play challenges / tasks), MRMG _UC5 (Design new challenges).
	Stimulus	Player's mobile device detects a new beacon with suitable metadata.
	Response	Game server has verified the beacon's metadata and added the beacon to the repository.
	Covers requirements	RF18

8.2 Services and Interfaces

As shown in Figure 8.1, the provisioning and discovery component provides the following functionalities: service discovery using Bluetooth and DNS, and device and licence provisioning.

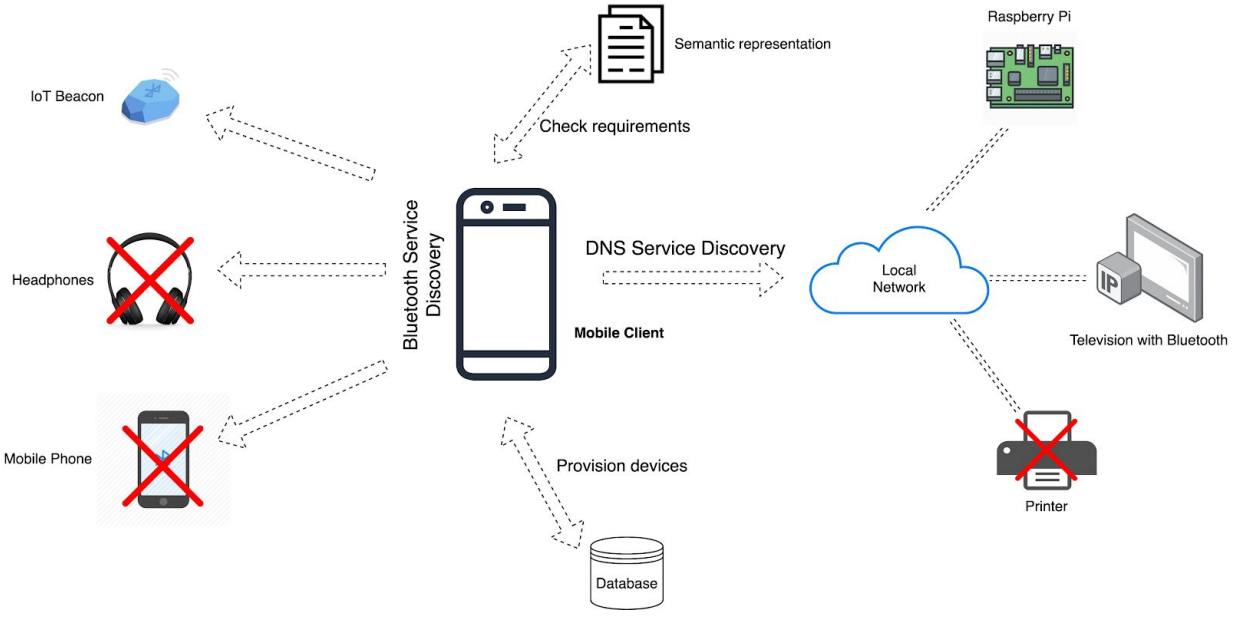

Figure 8.1: High-level overview of Provisioning and Discovery component and its interfaces

The figure above shows the high-level architecture of the component. The mobile client uses Bluetooth and DNS service discovery protocols to search for new IoT devices and later, checks wherever the semantic description of the device matches the predefined requirements. If requirements are fulfilled, the devices (in this example IoT Beacon, Raspberry Pi, and television) are provisioned to the given database.

Table 8.2. Interfaces of the component

ID	Interface Content	
IF01	Name	**Bluetooth discovery**
	Description	This interface provides operations to perform a Bluetooth scan and discover open Bluetooth devices
	Key inputs	Bluetooth scanning interval and timeout.
	Response	It lists all the Bluetooth devices and downloads the device description file using the URL provided by the Bluetooth service.
IF02	Name	**Local network discovery**
	Description	This interface provides operations to perform a Multicast-DNS scan to find beacons services published on the local (WLAN, etc.) network.

	Key inputs	Service type.
	Response	It lists all the devices using the webthing service and downloads the device description file using the URL provided by mDNS discovery.
IF03	Name	**Device provisioning**
	Description	This interface goes through the semantic representations and checks them against the requirement for provisioning of the device.
	Key inputs	Semantics representation file of discovered beacons.
	Response	It either adds the device to the client database or rejects it.
IF04	Name	**Licensing provisioning**
	Description	The interface checks for the license in the semantic representation file and automatically makes a contract for usage of the device.
	Key inputs	Semantics representation file and information for device usage.
	Response	It deploys a contract and compensates for the usage of the provisioned device.

8.3 The internal structure

Discovery of the IoT devices is implemented using the following two existing protocols: Bluetooth Service Discovery protocol and DNS Service Discovery with multicast.

Bluetooth Service Discovery protocol
In Bluetooth environments, services can be discovered using the Service Discovery Protocol (SDP) and they can then be accessed using other protocols defined by Bluetooth. The SDP provides a means for applications to discover which services are available and to determine the characteristics of those available services. It is a peer-to-peer protocol. The Bluetooth emitter will therefore use this protocol for the discovery of devices within its proximity, and then go further to search for services found on these devices. SDP focuses primarily on discovering services available from or through Bluetooth devices.

DNS Service Discovery with multicast
DNS service discovery (DNS-SD) allows clients to discover a named list of service instances, given a service type, and to resolve those services to hostnames using standard DNS queries. It discovers devices and services on a local area network using IP protocols, without requiring the user to configure them manually. DNS service discovery requests can also be sent over a multicast link, and it can be combined with multicast-DNS (mDNS) to yield zero-configuration DNS-SD.

Figure 8.2 presents the internal structure of the Discovery and Provisioning component.

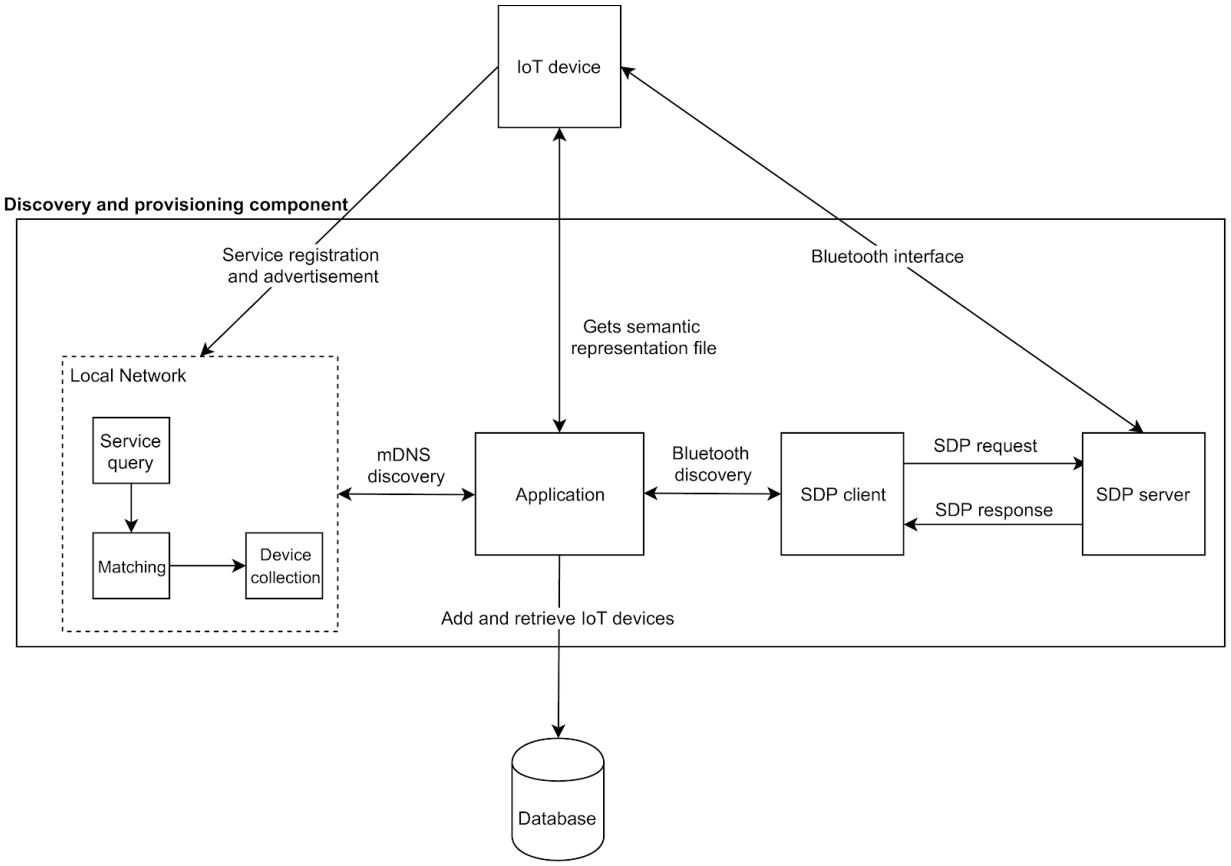

Figure 8.2: Internal structure of Provisioning and Discovery

In more detail, the main entities shown in Figure 8.2 are:

Application: E.g. a mobile application to search for the new IoT devices on the WLAN or using the Bluetooth interface.

SDP Server: An SDP server is any Bluetooth device that offers a service or services to other Bluetooth devices. Information about the services is maintained in SDP databases. Each SDP server has its own database; there is no central database.

SDP Client: SDP clients use the services provided by servers. To allow them to do this, servers and clients exchange information about services using service records

Database: Stores the information about available IoT devices. The application connects to database by the provided URL and sends the IoT device information to the database to provision devices. The database is run by the party utilising the component.

DNS discovery: Service query is used to find all the particular type of service on the network e.g. "_webthing._tcp". Multicast-DNS is used to return the name of the services found and add them to the cached list on the device. When an application wants to use a service, multicast-DNS resolves the chosen service name to an IP address and port.

Figure 8.3 presents the information flow in the discovery and registration process. Here, the resources types, provided services, licences and compensation are all described using WoT TD semantic representations.

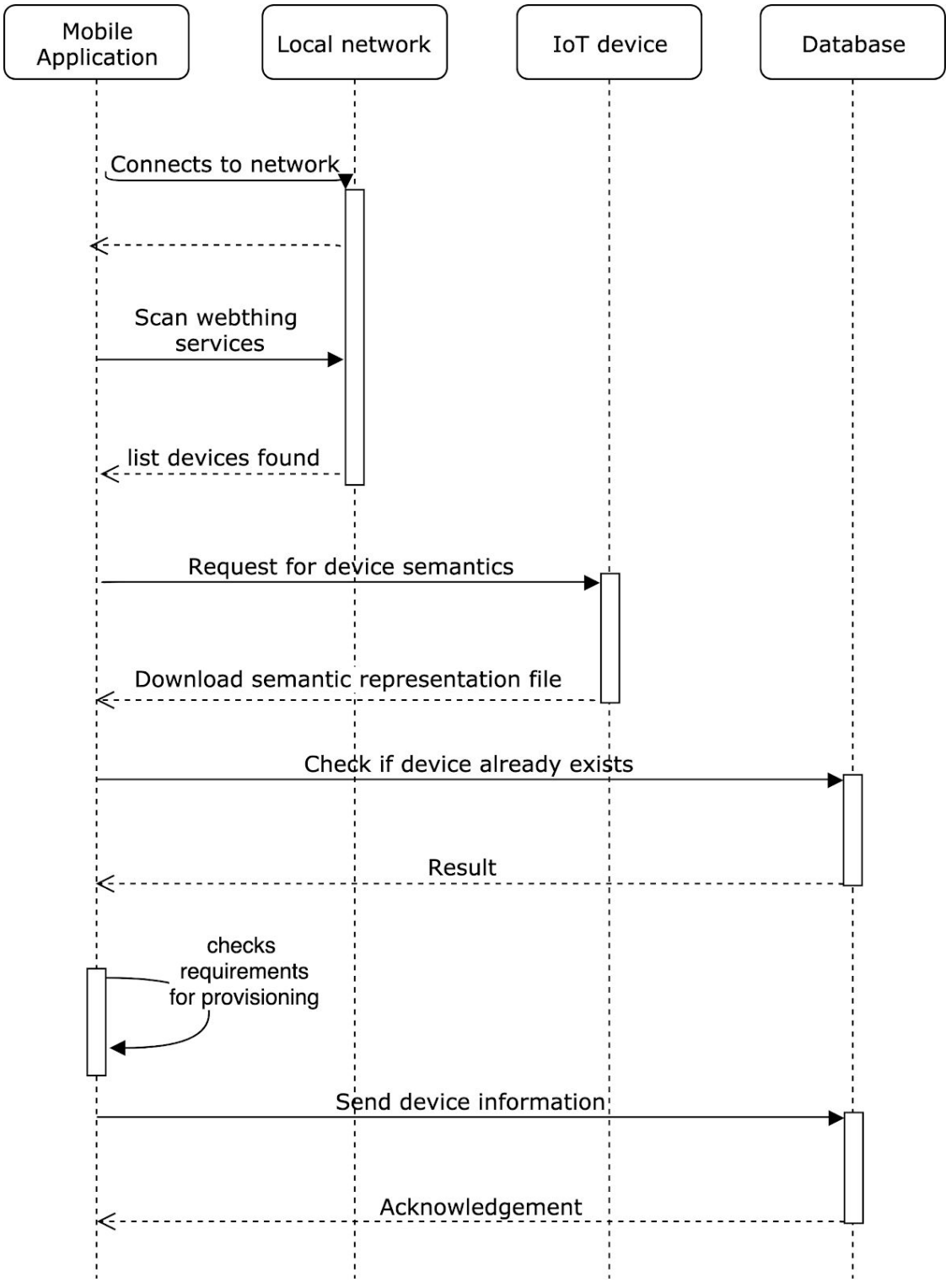

Figure 8.3: The process of discovering new devices on a local network

After the mobile application connects to the local network, it starts searching for the particular type of service on the network. After getting the list, the application request for the semantic description of the devices and check them again the requirements. In the end, it sends new devices information to the database.

8.4 Scenario walkthrough

Table 8.4 describes how the Discovery and provisioning component satisfies the scenario presented in section 8.1.

Table 8.4 Scenario walkthrough

ID	Scenario Validation	
W01	Description	Discovering and provisioning of devices using service discovery
	Walkthrough	The player's game application on the mobile phone starts searching for the devices in the background using the WLAN. Whenever the mobile connects to a network, it uses the IF02 interface to discover the devices available over the local area network and requests their semantic description to look for the services provided. Once the client checks that the requirements are met, it calls the IF03 interface to automatically provision the device to database along with its details. The IF04 interface will be used to automatically create a contract for the usage of the discovered device
W02	Description	Discovering and provisioning of devices using Bluetooth
	Walkthrough	For Bluetooth discovery, the client calls the IF01 interface to scan for Bluetooth devices nearby and then, using IF03 interface, adds all devices to the database after checking them against the requirements. The IF04 interface will be used to automatically create a contract for the usage of the discovered device.

9 How Components are used in the SOFIE Pilots

This section details how the SOFIE pilots are utilising the framework components to realise some of the key functionalities of their use cases and how the functionalities have been implemented in the Federation Adapter and Application API components.

9.1 Food Supply Chain Pilot

The Food Supply Chain (FSC) pilot makes use of a consortium DLT to federate otherwise siloed IoT platforms and to establish a distributed and immutable data management layer that makes traceability and quality control of transported products more robust, reliable and time efficient for all parties of the food supply chain process. The business process of the pilot is shown in Figure 9.1.

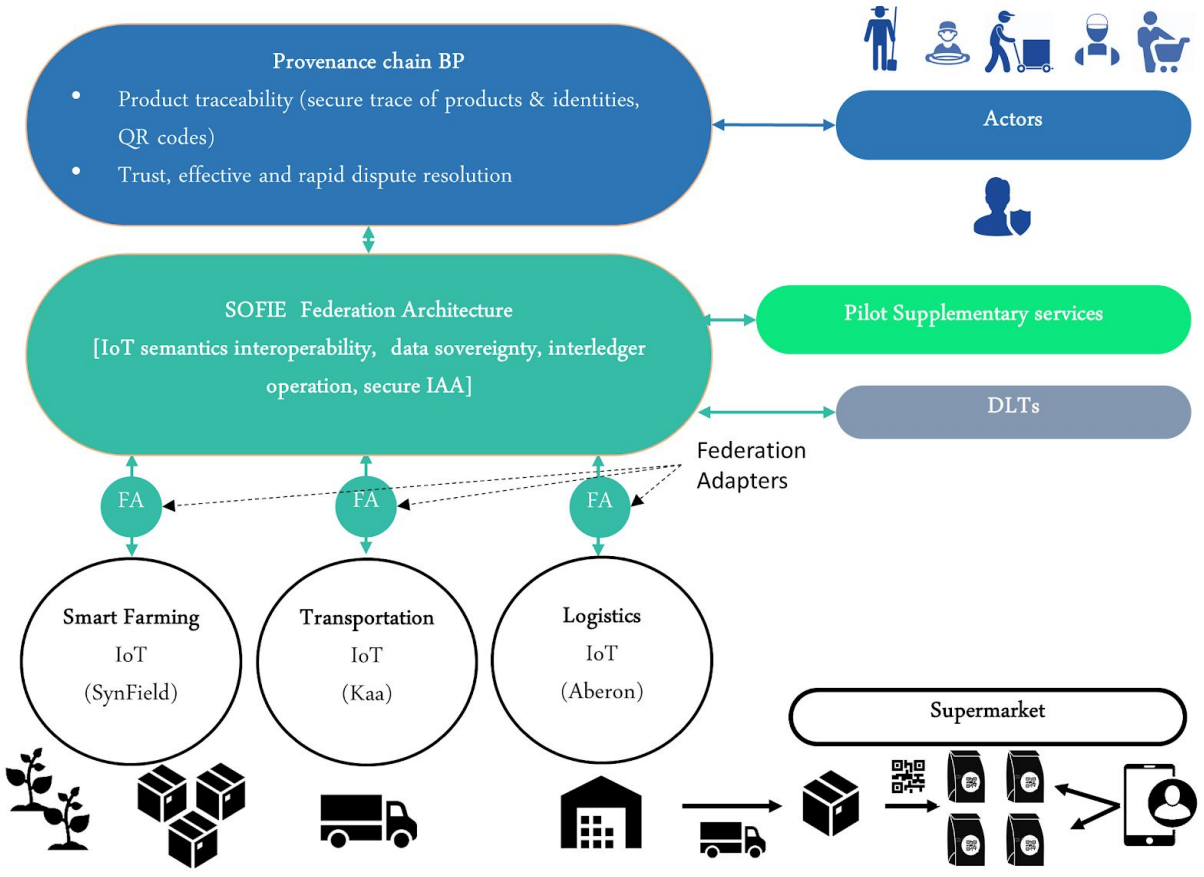

Figure 9.1: Overview of the food supply chain pilot

The pilot federates three IoT platforms and operates data related to trackable assets (i.e. smart boxes that carry produce over the chain) as those are handled and handed off by the corresponding business segments before finally reaching the selling point. The system architecture makes use of SOFIE framework components to guarantee data integrity, interoperability and privacy towards improving efficiency of traceability, food safety and quality control services. In particular, the pilot aims to demonstrate two enabling services: i) a QR code-based service for retrieval of product history by the consumers, and ii) an audit service

for involved business parties to identify and verify points of failure affecting produce quality as produce is transported from the farm to the selling point.

The key benefits of applying SOFIE technology in the FSC are summarised as:

- The development of a decentralised and immutable data management and storage framework that leverages DLTs to enable automated, reliable and flexible operation of all critical data used in the food supply chain. This aims to ease the development of further enabling services, e.g. by using add-on modules, processes or smart contracts to enable automated payments, trust evaluation, etc., among the involved parties.
- The transparent federation of heterogeneous IoT silos used by the involved parties in a technology-agnostic way that enables a 'cross platform access' pattern of interoperability. As a result, applications or services to be developed can discover resources from different IoT platforms through the same interfaces and by using the same formats to communicate data.
- The provision of secure data access and retrieval services in the sense of data confidentiality and integrity for all involved parties based on their role in the supply chain (e.g. viewing where the produce comes from, which steps it passes through, which other produce may be affected in the case of quality issues, etc.).

The following section presentes how the various SOFIE framework components are used in the FSC pilot to reach the above mentioned benefits.

Interledger component

The FSC pilot uses the following ledgers:

1. A private consortium ledger (Ethereum) to store IoT data which is collected from the three business companies operating the farming, transportation and warehouse segments, respectively. This data captures both the conditions and the handoffs of the trackable boxes as they move along the supply chain.
2. A KSI blockchain to create a unique signature (anchor) per box by hashing the series of data which refers to that specific box and has been stored in the consortium ledger.
3. A public ledger (Ethereum) where the above mentioned signatures are stored. These signatures are used by external entities (i.e. entities who do not have direct access to the consortium ledger, such as the supermarket organisation and the customers) to verify the authenticity of data requested from the consortium ledger.

The integration between the interledger component and the data management layer of the pilot is under development. In particular, the interledger will be used in the following ways:

- for data transfer, a multi-ledger operation that implements a one-way transfer of data (signatures) from the consortium ledger through the KSI to the public ledger. This involves running smart contracts in both Ethereum ledgers to control transactions and to exchange information between them, as well as using the interface of the KSI blockchain to produce signatures.
- for a proof of integrity operation to cross check data on the consortium ledger against the signatures stored in the public ledger, thus validating the integrity of the data in the case of audits on behalf of external entities.

Identity, Authentication and Authorisation component

In the FSC pilot, two IAA mechanisms are implemented for the actors and the IoT platforms, respectively. More specifically:

1. Authentication and authorisation of actors: The username/password method of the IAA component is used to authenticate the actors of the various business segments. The AA API of the Oauth2 server (keycloak) is used by the FSC web application which implements the OAuth2 client. The Oauth2 server provides the client with authorisation tokens, so actors are able to access the endpoints of the Supervisor Web Server (SWS) based on their role.
2. IoT platforms are authenticated in the consortium ledger by applying a simplified version of Model 2 of the IAA component (i.e. a smart contract that handles authorisation requests), since no payments are considered in the pilot case. Once an IoT platform is registered in the consortium ledger (i.e. its wallet address is recorded), the smart contract acts as an authorisation contract to confirm whether it has the authority to perform certain transactions or not.

Privacy and Data Sovereignty

In the FSC pilot, the privacy and data sovereignty component provides guidelines for how to address data privacy in the implemented use cases and in the actors' activities. In particular, the following policies have been implemented:

- The SWS is a full node of the consortium ledger which is also responsible for providing wallet functionality to the federated IoT platforms. These platforms are registered directly to the consortium ledger
- Every transaction that is fired by an IoT platform to the consortium ledger is digitally signed by using the platforms' private key.
- Upon registration of an actor into the FSC web application, a unique ID and role are assigned to it. This information is the only one related to the actor profile which is included in the transactions written in the consortium ledger. The mapping between actors' profiles, IDs and roles is accessible by the SOFIE system administrator only (referred to as the consortium certifier organisation).
- In the preparation of data that relates to multiple actors (possibly belonging to different segments) which should be combined together, actor IDs are used to retrieve and chain information from the consortium ledger.

Semantic Representation

In the FSC pilot, three IoT platforms are federated into the system architecture. For each platform, an adaptation layer is implemented to interface the SWS and to expose platform's functionality and things' services according to the same semantics, thus enabling cross platform access and interoperability.

The information model that is implemented in the FSC system architecture is shown in Figure 9.2.

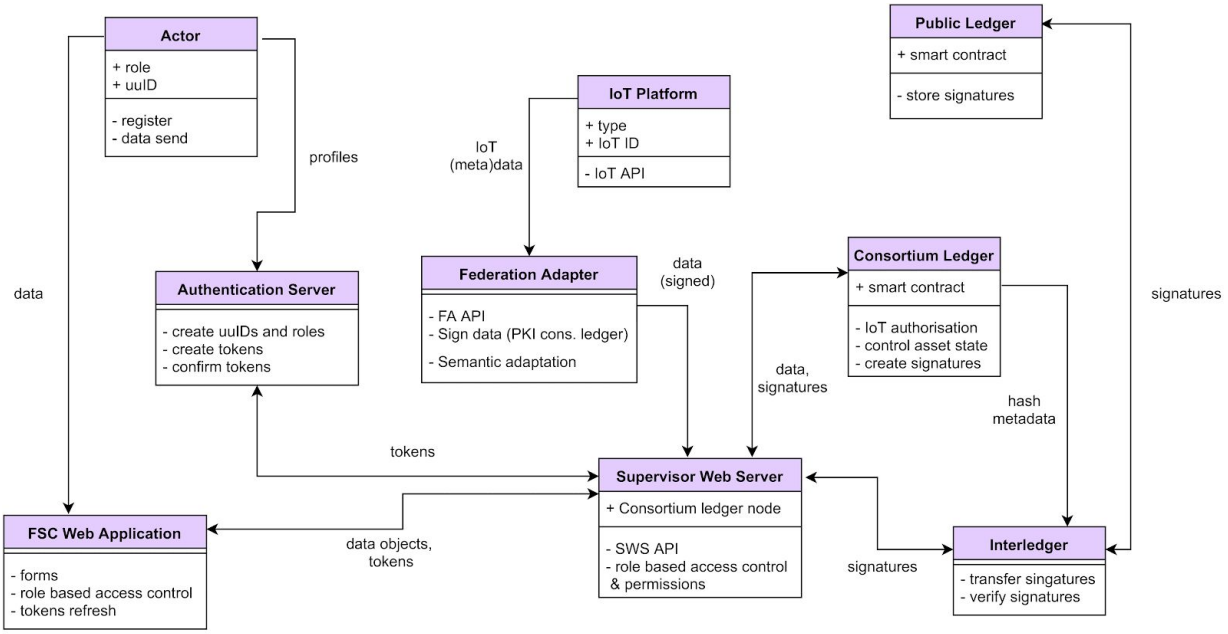

Figure 9.2: Conceptual data model of the food supply chain pilot

The model defines the following main entities:

- Actors: actors register to the FSC architecture by defining profile details (e.g. their role, business segment, username/password, etc.) and also provide (meta)data about how they handle assets (boxes) during their activities.
- Authentication server: it is responsible for creating (unique) IDs for actors as well as for creating and verifying role-based JWT tokens, which are used by the FSC web application and the SWS to control how actors access the provided services.
- FSC web application: it provides a set of forms to be used by the actors. It performs role-based access control and initiates token refresh by requesting a new access *token* whenever needed.
- Supervisor Web Server (SWS): it implements all the backend logic of the system about how to collect, filter and manage data objects, tokens and resources. It is a full node of the consortium ledger that prepares and executes all the transactions performed by the various actors. It exposes a number of Restful API endpoints to share data objects, tokens and signatures to other modules of the model.
- Consortium ledger: it executes smart contracts to maintain the status of the pilot, e.g. status and ownership of boxes, registered actors and IoT platforms, etc.
- Public ledger: a public ledger where signatures of the consortium ledger data are stored.
- Interledger: It is responsible for transferring signatures between the consortium ledger and the public ledger, as well as for verifying the binding between these signatures and the corresponding data of the consortium ledger.
- Federation adapter: it implements syntactic and semantic interoperability between the corresponding IoT platform and the pilot information model. It exposes a RESTful API to provide data and things services to the SWS and also uses the platform's PKI to digitally sign every data object that is sent to the SWS.
- IoT platform: each federated IoT platform exposes a northbound API to provide its services and data.

Marketplace

This component is not used in the FSC pilot.

Provisioning and Discovery

The provisioning and discovery component will enable the FSC pilot application to find and use resources across platforms in a dynamic way through uniform interfaces. This component will be integrated into the second release of the FSC pilot architecture.

Federation Adapters

Each federated IoT platform is connected to the pilot system architecture through a federation adapter which is responsible to:

- Provide an adaptation layer for data and resources, so as to enable unified syntactic and semantic interoperability and secure usage of platform resources, services and data.
- Implement a domain-specific API to communicate with SWS and allow rapid cross-platform access and application development.
- Create wallet addresses to register the IoT platform in the consortium ledger and digitally sign data objects which are sent to the SWS.

9.2 Decentralised Energy Flexibility Marketplace Pilot

Most of the electricity produced from renewable sources (e.g. solar cells) will normally be consumed by energy customers adjacent to the generation plant, however, an excess of the generated power will create reverse power flows through the substation of the Low Voltage (LV) distribution network. One of the objectives of the Distribution System Operator (DSO) managing the distribution network is to reduce this reverse power flow, since it can cause malfunctions and reduce equipment lifetime.

To achieve this objective, the DSO will create Demand-Response (DR) campaigns to consume the surplus of energy. DR campaigns will be directed to Electric Vehicle (EV) Fleet Managers due to their ability to consume large amounts of energy by recharging electric vehicle batteries. Fleet Managers are incentivised to participate in DR campaigns thanks to the economic bonus granted for the flexibility provided to the DSO.

To satisfy the flexibility request, the Fleet Manager will create an auction to request lower priced electricity supply from the Energy Retailers. In this scenario, the DSO has avoided reverse power flows and solved the problems of the electricity network, the Fleet Manager has charged its fleet of electric vehicles at an advantageous price, and the Energy Retailer has achieved its daily energy buying and selling goals.

Considering the current trend in distribution networks, in which the amount of distributed generators is increasing, the DSO is assuming increasing responsibility as coordinator of distributed local resources. The reliable and secure observability of the network is crucial for enabling market participation of distributed generators and will allow the implementation of a flexibility market and peer-to-peer transactions. In addition, real time measurements are a pillar for the deployment of real time management of the distributed resources carried out by the DSO or other stakeholders (e.g. Aggregator, RESCO companies). In this environment, the SOFIE framework enables the DSO to provide the aforementioned services for managing real-time data in a secure and open way and potentially also enables new business scenarios for the fleet managers and energy retailers.

SOFIE's role

The SOFIE framework makes it possible to have

- a clear identification of the actors involved and their role in the marketplace
- the standardisation of different IoT environments such as the DSO metering infrastructure or the Fleet Manager equipment
- the interaction of different platforms with different blockchain technologies for data
- process and market management as well as micro-contracts and micro-payments management in a quick and user-friendly way
- interoperability, security, transparency, and auditability of the marketplace actions.

Figure 9.3 provides an overview of the Energy Flexibility Marketplace pilot's implementation.

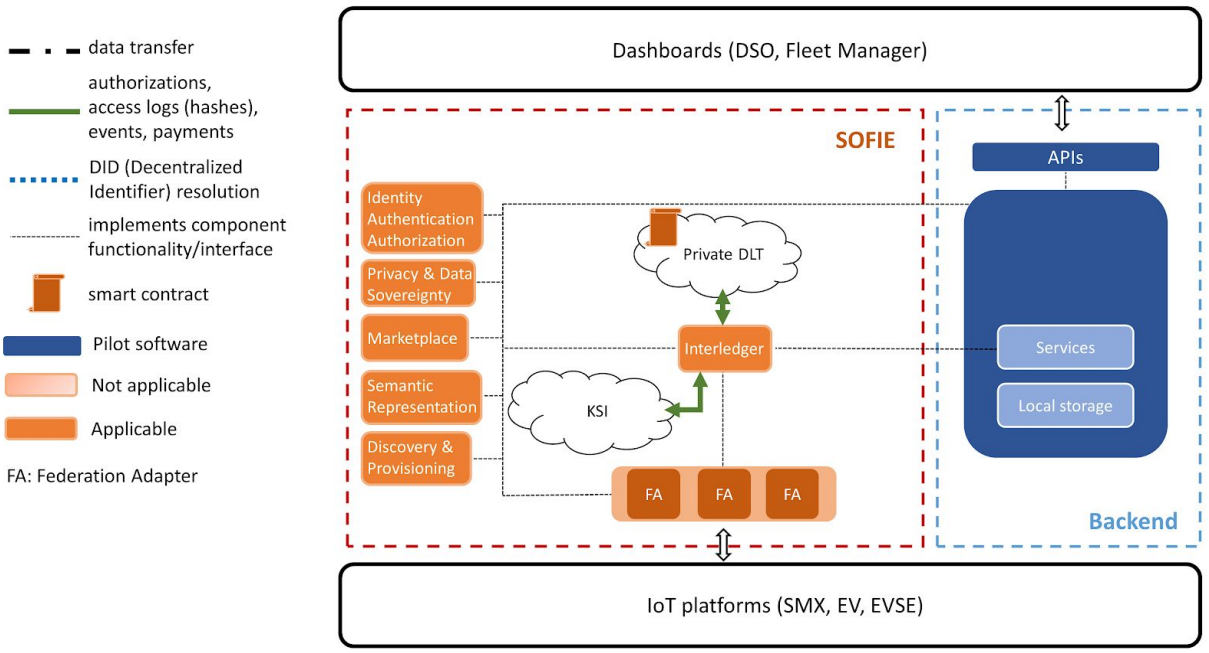

Figure 9.3: The Energy Flexibility Marketplace pilot's implementation.

Semantic Representation, Discovery and Provisioning

Thanks to SOFIE *Semantic Representation* and *Provisioning and Discovery* components, it is possible for client applications to use the services and devices available on the network and to know the models describing IoT Things and services on top of federated IoT environments. In this way, for example, the DSO can know in advance how many Fleet Managers operate under its network, what kind of flexibility service they provide (e.g. day-ahead or intra-day market) and *how* to communicate with them. In the same way, a fleet manager can query for the services requested by the DSOs managing the networks in the zones it operates in and can *understand* such requests. In turn, the Fleet Manager can query for the market offers made by the energy retailers in advance and select the offer that best suits its needs.

Marketplace

The SOFIE *Marketplace* component is the key component of this scenario. It enables the actual trade of resources in an automated, trusted, and decentralised way. The blockchain powering the marketplace grants it security, resilience, transparency, and traceability, with the effect of increasing a healthy competition among the actors participating. Trades are negotiated and verified, and payments are performed seamlessly on behalf of the actors with the least user interaction possible, but always allowing for transparent verification.

Interledger

Thanks to the Interledger component, it is possible to enrich the scenario with different features. The first and most obvious way to use it, is to build an additional layer on top of the marketplace, that further improves its security and transparency. In the scenario, the marketplace is meant to be run on a private distributed ledger. This choice maximizes the privacy and keeps the network control secure from external entities, but sacrifices the redundancy and auditability granted by a public ledger. Using a second layer public ledger as a trust anchor helps to keep the benefits of the private ledger together with the benefits of the public one.

Another interesting use case enabled by the interledger component, is the usage of a smart contract capable blockchain (e.g. Ethereum) together with another blockchain dedicated to the micro-payments (e.g. Lightning Network payments over Bitcoin blockchain).

Finally, the interledger component enables the capability to federate together in the same scenario services and actors operating natively on different blockchains.

Federation Adapters

SOFIE *Federation Adapters* federate the IoT systems constituting the infrastructure used by the smart meters, the electrical vehicles and the supply equipment, providing data and services representation according to the SOFIE semantic representation and supporting authentication and authorisation.

Pilot implementation

Figure 9.4: The DSO uses a dashboard to monitor the network.

The business logic of each actor is enabled by a set of specific APIs. The client applications will use the SOFIE services to provide the “common” features, while the specific APIs are being called to provide specific functionalities. For instance, the SOFIE federation adapters enable access to the data provided by the IoT smart meters, and the DSO is able to visualise them on its dedicated dashboard, together with the forecast obtained by invoking its own production/consumption forecasting service via the API. The resulting forecast will then be used by the operator as a decision support tool before creating a new day-ahead DR campaign using the SOFIE marketplace component.

9.3 Location-based Mobile Gaming Pilot

The gaming pilot utilises the SOFIE framework to explore how DLTs can be leveraged to provide new gaming features for players as well as on validating the potential of the location-based IoT use cases.

Multiple use cases will be studied throughout the pilot. The first use case utilises a prototype game that enables players to collect and trade in-game content (e.g. characters, weapons, equipment, parts) either by swapping with or buying from other players. Here, DLTs are used to provide player ownership of the asset, transparency, and consistency of asset attributes and transactions.

In the second use case, a prototype scavenger hunt location-based game uses IoT beacons and an ecosystem backed by a DLT. The players need to solve the riddles using the clues they receive on their smartphones. Solving the riddle will reveal the location of the IoT beacon and the player has to physically visit those areas to collect the points. The competition is to find the beacon locations and solve the tasks until the last beacon has been reached. After performing a series of tasks and visiting multiple locations, the players are awarded points, which they can later redeem for rewards. Here, DLTs will be used to manage, e.g. player check-ins, points collection and rewards.

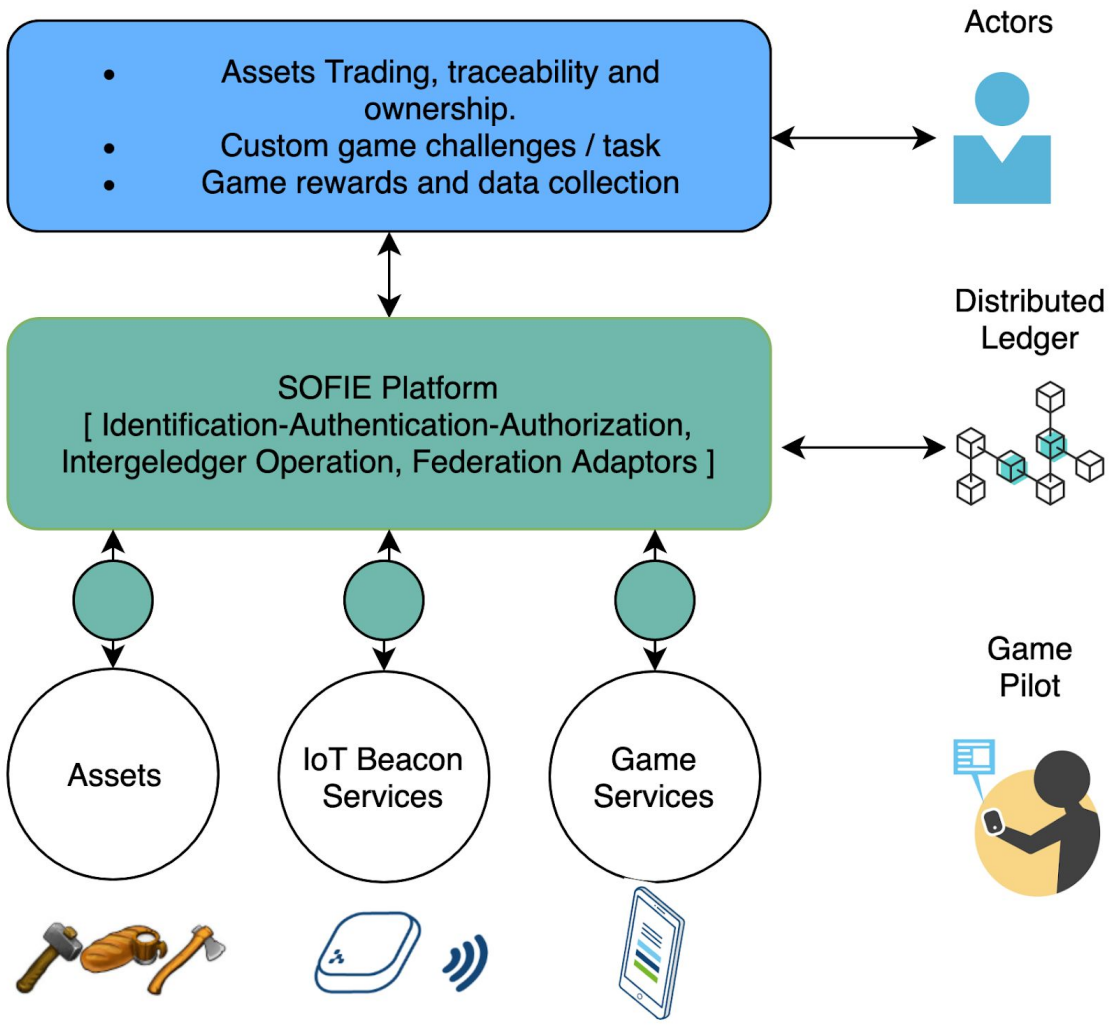

Figure 9.5: Overview of the location based mobile gaming pilot.

SOFIE's role

Multiple SOFIE framework components are used in the gaming pilot to address the above challenges.

Identity, Authentication and Authorisation (IAA)

In the gaming pilot, the IAA component is utilised to authenticate the actors of the various business segments. The IAA component ensures that only authorised players can participate in transactions on both ledgers. Decentralised identifiers (DIDs) are used to manage the identity of participants. For the authorised entities, a secure data exchange channel will be used. The gaming services can only be accessed using the custom credentials and session services will be used for authenticating players. The session and the associated user are accessible only from the device the session was created on.

Interledger

The gaming pilot uses the following ledgers:

1. A private consortium ledger (Hyperledger Fabric) to store game data which is collected from the gaming companies and the Points of Interest (PoIs). This ledger provides a contractual relationship between the PoI and the game. It is also used to register the User Generated Content (UGC) along with keeping track of their owners. Finally, it is also used to issue the rewards or vouchers and keep track of them.
2. A public ledger (Ethereum) that enables the trading of different types of assets. Users are able to freely negotiate the trades and the ledger also provides auditability to help with potential dispute resolutions.

The Interledger component will be used as a bridge between the Ethereum and Hyperlegder Fabric ledgers. The required asset information from the Fabric will be transferred to Ethereum using the data transfer interface of the interledger .The asset can than be traded on the Ethereum marketplace. The main role of the Interledger is to register callbacks to certain events and then react when such events are caught. This involves running smart contracts in both ledgers to control transactions and exchange information between them.

Provisioning and Discovery

The provisioning and discovery component will enable the gaming pilot application to find and use IoT beacons in a dynamic way. Already deployed working Bluetooth devices can be discovered and added to the repository, so that they can be later used as PoIs in the pilot. In this way, we can increase the number of IoT beacon devices used for the game without the additional cost of deploying and purchasing the beacons. This component will be used along with semantic representation that exposes the functionality of the devices and the methods of communication with them, thus enabling cross-platform access and interoperability.

The component also provides beacons with decision support, helping them make decisions in real time. This provisioning capability allows owners to control who can communicate with their devices and what they can do with them. The mobile application will automate decision support by uploading digital contracts for the devices discovered. These contracts stipulate requirements that must be met in order to gain access to the devices, under what conditions access can be given, for how long and how often. This allows new devices to be introduced and their owners to be compensated without human intervention if the terms of the contract posed on the device fall within the guidelines of the game company thus distributing and automating the required negotiations.

Marketplace

The SOFIE Marketplace component will be used for the trading of the in-game virtual assets. It enables the actual trade of resources in an automated, trusted, and decentralised way. Once digital assets have been stored on the ledger, the ownership and the item itself cannot be altered. DLTs also help maintain the scarcity of a virtual item in a secure and verified way. The DLT-based marketplace grants security, transparency, and traceability, with the effect of increasing a healthy competition among the players participating.

Semantic representation

SOFIE's Semantic Representation component will be used to provide a solution for device and service interoperability. This component will be used along with provisioning and discovering component. After discovering the beacons, the semantic representation will be used to unify different IoT devices used as beacons. Web of Things (WoT) standards are used for defining the description of the IoT beacons.

9.4 Decentralised Energy Data Exchange Pilot

The decentralised energy data exchange pilot is focused on providing a proof-of-concept for secure data exchange and access rights management between smart meter data owners and energy service providers (intermediaries, distributors, and brokers). The pilot utilises the capabilities of the SOFIE framework for the validation and demonstration of the defined scenarios and use cases.

The pilot uses the Estfeed data hub (connecting 700 000 smart meters in Estonia) as a main source for data exchange demonstration. The data that is used in the Pilot is anonymised and with the same structure and characteristics as is in production Estfeed data hub. The main concepts are described in the following figure, where the SOFIE approach and the added value are presented.

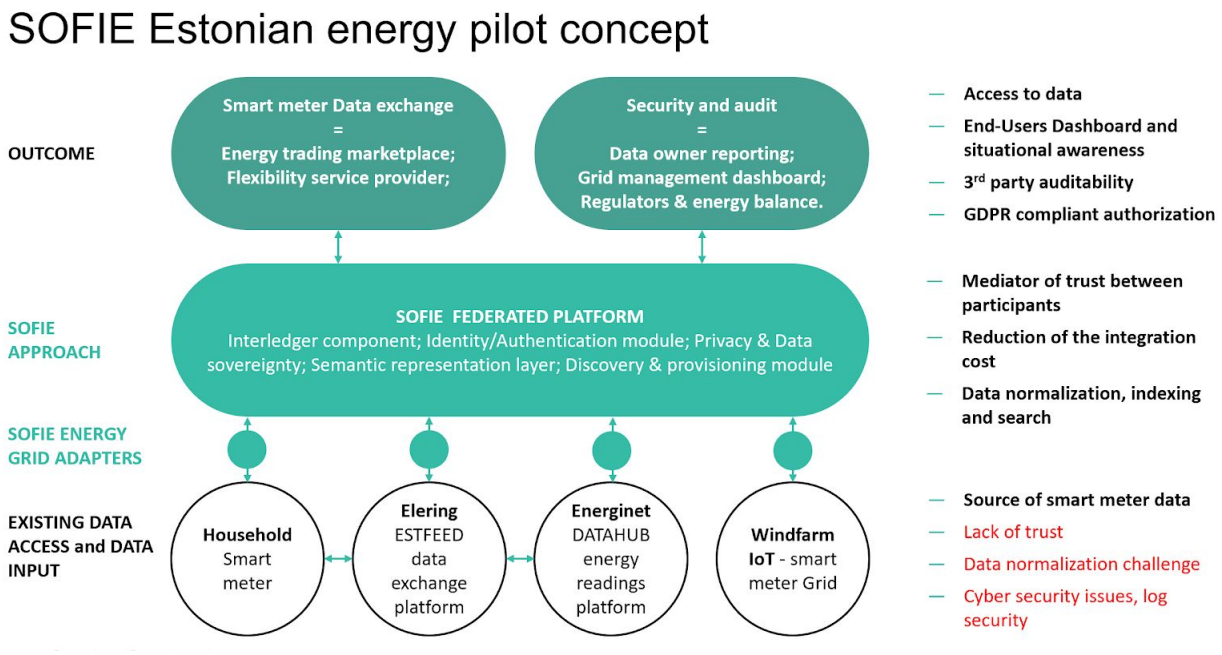

Figure 9.6: An overview of the Energy Data Exchange Pilot

SOFIE's role

The architecture of the Energy Data Exchange pilot has been depicted in Figure 9.7.

The SOFIE *Federation Adapters* will be used to enable data exchange with different smart meter systems:

- **National data hub** - existing information system having information about users and their consumption history.
- **Single metering point** - the adapter will enable requesting metering data from existing devices.
- **Wind farm network** - the adapter enables data exchange with a group of smart meter devices for consumption and production data.

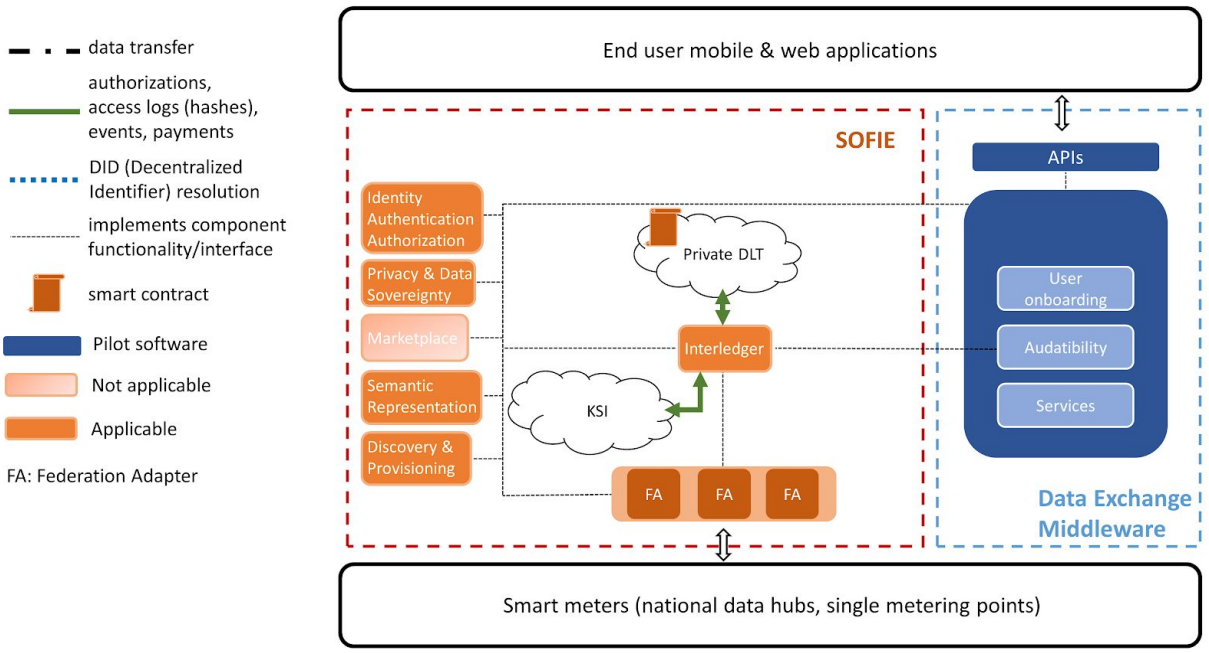

Figure 9.7: The architecture of the Energy Data Exchange Pilot

Existing smart meter data systems use different technologies and representation models, which makes integration more complex. The SOFIE federation adapter introduces a common data model to represent measuring data, so the details of existing smart meter data systems will be hidden and different participants will have no need to be aware of those. When new systems are added to the platform they only need to map their existing data model to the model provided by federation adapter.

Identity, Authentication and Authorisation

The SOFIE IAA (Identity, Authentication, Authorisation) component makes it possible to support the requirement that only authorised entities can participate in transactions. Decentralised identifiers (DIDs) are used to manage the identity of participants, so the actors are not dependent on external identity providers, which minimises 3rd party data leakages. To map real-life identities with DIDs some special issuers are needed to support the credentials. For the authorised entities, a secure data exchange channel will be used. A person, who is the owner of the metering device, should be able to grant and revoke access to its data. Decentralised identifiers give complete control to data owner over its data.

Semantic Representation and Provisioning and Discovery

The Semantic Representation and the Provisioning and Discovery components provide a solution for device and service interoperability. Measuring devices and measuring data have semantic representation to unify different devices used as smart meters. Web of Things (WoT) standards are used for the descriptions.

Federation Adapter

The central component for the pilot is the federation adapter. From the data owner's point of view it provides the possibilities to make the data available for other market participants in a secure and transparent way. The federation adapter also helps service providers to handle secure data exchange with data owners.

10 References

[Elo2019] T. Elo et al. “SOFIE Deliverable 2.4 - Federation Architecture, 2nd version”, June 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D2.4-Federation_Architecture_2nd_version_v1.00.pdf.

[Fot2018] N. Fotiou, V.A. Siris, G.C. Polyzos, "Interacting with the Internet of Things Using Smart Contracts and Blockchain Technologies", Proc. of Security, Privacy, and Anonymity in Computation, Communication, and Storage 2018 (SpaCCS 2018), Melbourne, Australia, 2018.

[Kov2019] M. Kovatsch et al. “Web of Things (WoT) Architecture”, retrieved August 2019, Available at: https://www.w3.org/TR/wot-architecture/.

[Lag2019] D. Lagutin, Y, Kortesniemi, N. Fotiou, V.A. Siris, "Enabling Decentralised Identifiers and Verifiable Credentials for Constrained Internet-of-Things Devices using OAuth-based Delegation," NDSS Workshop on Decentralized IoT Systems and Security (DISS), San Diego, CA, USA, 2019.

[Oik2019] I. Oikonomidis et al. “SOFIE Deliverable D5.2 - Initial Platform Validation”, June 2019. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D5.2-Initial_Platform_Validation.pdf.

[Paa2018] S. Paavolainen et al. “SOFIE Deliverable D2.3 - Federation Framework, 1st version”, October 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D2.3-Federation_Framework_1st_version_v1.00.pdf.

[Sir2019a] V.A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, G.C. Polyzos, “Interledger Approaches,” IEEE Access, July 2019.

[Sir2019b] V.A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, G.C. Polyzos, "OAuth 2.0 meets Blockchain for Authorization in Constrained IoT Environments," IEEE 5th World Forum on Internet of Things, Limerick, Ireland, 2019.