SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D3.5

Final Business Platform Integration Report

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	12.5.2021
Author(s)	Filippo Vimini (LMF), Giuseppe Raveduto (ENG), Mait Märdin (GT), Antonio Antonino (LMF), Ahsan Manzoor (ROVIO), Yannis Oikonomidis (SYN)
	Filippo Vimini (LMF), filippo.vimini@ericsson.com
Target Dissemination Level	Public
Status of the Document	Completed
Version	1.20
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.

Summary of changes

Location	Description
Section 1	Described D2.7 and D5.4 defining test cases
Section 2.2	Describe deployment models and provide links to details
Section 3.1.2.2	Elaborated on benefits of Interledger to SMAUG
Section 3.5.2.1	Updated second paragraph

Table of Contents

List of abbreviations

API	Application Programming Interface
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 [component]
IL	Interledger [component]
IoT	Internet of Things
KSI	Keyless Signing Infrastructure (GuardTime)
MP	Marketplace [component]
P&D	Provisioning and Discovery [component]
PDS	Privacy and Data Sovereignty [component]
RFID	Radio Frequency IDentification
SR	Semantic Representation [component]
TD	Things Descriptor
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 systems in an open and secure manner. The main goal of this project is to enable communication among diversified applications from various areas. With SOFIE these applications are able to utilise heterogeneous IoT platforms and autonomous things across technological, organisational and administrative borders in an open and secure manner, making reuse of existing infrastructure and data easy.

The SOFIE architecture is described in SOFIE Deliverable '2.6 - Federation Architecture, final version' [D2.6]. SOFIE enables communication and interoperability by *federating the actions between different IoT systems using interledger technologies*. The Architecture consists of components that enable the main functionalities of the system and of federation adapters used as connection points for external IoT systems with a SOFIE enabled system *without requiring changes to the IoT devices*. SOFIE Deliverable '2.7 - Federation Framework, final version' [D2.7] describes the implementation of the aforementioned components.

In the SOFIE project, IoT business platforms have been developed (WP5), based on the framework components of the SOFIE architecture (WP2). These business platforms showcase the potential of the SOFIE architecture and the framework components to address relevant requirements for different business domains.

This document presents the final integration of the SOFIE components in the business platforms. It describes how all the business platforms implement the components, describing their overall architecture and the steps required to configure the baseline components to work in the specific business platform. The validation and test cases the platforms and components are subjected to are described in D2.7 for components [D2.7], and in D5.4 for business platforms [D5.4].

The rest of the document is organised as follows: Section 2 presents an overview of the SOFIE business platform. Then, Sections 3 detail the business platform architecture and components implementation. Section 4 summarises the document.

2 SOFIE business platform overview

2.1 Platform component integration

This chapter provides an overview of the business platforms and the framework components that have been used on each platform. These framework components are the *Marketplace* (MP), the Interledger (IL), Identity Authentication Authorization (IAA), Privacy and Data Sovereignty (PDS), Provisioning and Discovery (P&D) and Semantic Representation (SR). The framework components are detailed in the deliverable '2.7 - Federation Framework' [D2.7]. Table 1 shows the component that each platform implements. The business platforms use only the components needed to enable the main functionalities for each business case, as described in Deliverable '4.5 - Final Architecture, System, and Pilots Evaluation Report' [D4.5]. SMAUG acts as a reference platform and integrates all the components of the SOFIE framework.

Table 1: Components integrated into pilots.

Platform	Components
SMAUG	MP / IL / IAA / PDS / P&D / SR
Food Supply Chain	IL / IAA / PDS / SR
Decentralized Energy Data Exchange	IL / IAA / PDS
Decentralized Energy Flexibility Marketplace	MP / IL / SR
Context-Aware Mobile Gaming Pilot	IL / P&D / SR / MP

2.2 Deployment models

SOFIE platforms utilize Docker containers to provide repeatable builds and easy deployment of the individual platforms. The same applies to individual components where applicable. As some of the platforms and components interface with 3rd party services, the specific required configuration required for a test or production deployment vary. The detailed deployment guidance is provided in the README files of the individual platforms and components as shown in the table below. Roughly categorized, the different platforms and components fall into three different deployment models: standalone, configurable, and integrable. The standalone deployments are self-contained in the sense that they require minimal setup for a test setup, although a deployment tailored for specific, non-default needs would require custom configuration. Configurable ones require configuration due to external dependencies or their design to be used, but with proper configuration are deployable as docker containers. Finally, integrable need to be tailored for specific needs, although sandbox demonstrations are usually deployable as a standalone model.

Table 2: Deployment models and links to further details of platforms and components.

Platform	Model	Links
SMAUG	Standalone	README
Food Supply Chain	Configurable[1]	README

[1] FSC platform is proprietary; however, the Transportation FA is open-source and available.

Decentralized Energy Data Exchange	Configurable	Is not available as open source.[3]
Decentralized Energy Flexibility Marketplace	Configurable[2]	README
Context-Aware Mobile Gaming Pilot	Standalone	README
Component		
Marketplace (MP)	Integrable	README
Interledger (IL)	Configurable	README
Identity, Authentication, and Authorization (IAA)	Configurable	README (see also a tutorial)
Privacy and Data Sovereignty (PDS)	Configurable	README (see also a tutorial)
Provisioning and Discovery (P&D)	Integrable	README
Semantic Representation (SR)	Integrable	README

[2] DEFM platform also contains proprietary modules; however the FA is open-source and available.
[3] More information is provided in D5.4 chapter “4.4.1 Replication guidelines”

3 SOFIE platform integration

This chapter describes the business platform and the integration of the framework components. First, for each platform we define the overall architecture, which shows the position of each component. Second, for each component used in the platform, we discuss how it has been integrated.

3.1 SMAUG

The **Secure Marketplace for Access to Ubiquitous Goods**, or SMAUG, is a decentralised and open marketplace where smart locker owners can put their smart lockers for rent, and potential smart locker renters can place bids to get the authorisation to use them. Smart locker owners publish the availability of smart lockers on the marketplace by creating a request, i.e., a request for offers. The bids that smart locker renters place for the published requests are called offers.

SMAUG is intended as a reference implementation, to show how all the different SOFIE components can be used together to develop a system that benefits from all the features that the SOFIE framework provides. Furthermore, SMAUG is has been developed by LMF as a WP3 leader, and this means that an important target for SMAUG is to provide high-quality feedback to SOFIE component developers about the set of features the components offer, their level of reusability and extensibility, and their quality relating to how easily they can be integrated into systems other than the four pilots. This is achieved by following a “learn by doing” approach and testing the components via direct integration into a system developed from scratch during the last year of the project.

3.1.1 Pilot architecture

As previously presented, in a typical marketplace deployment three main entities are interacting with each other: the **marketplace owner** (MPO) manages the marketplace platform and enables smart locker owners (SLO) and **smart locker renters** (SLR) to interact via request and offer creations. The three main roles are also reflected in the architecture of the resulting system, as shown in Figure 3.1.a.

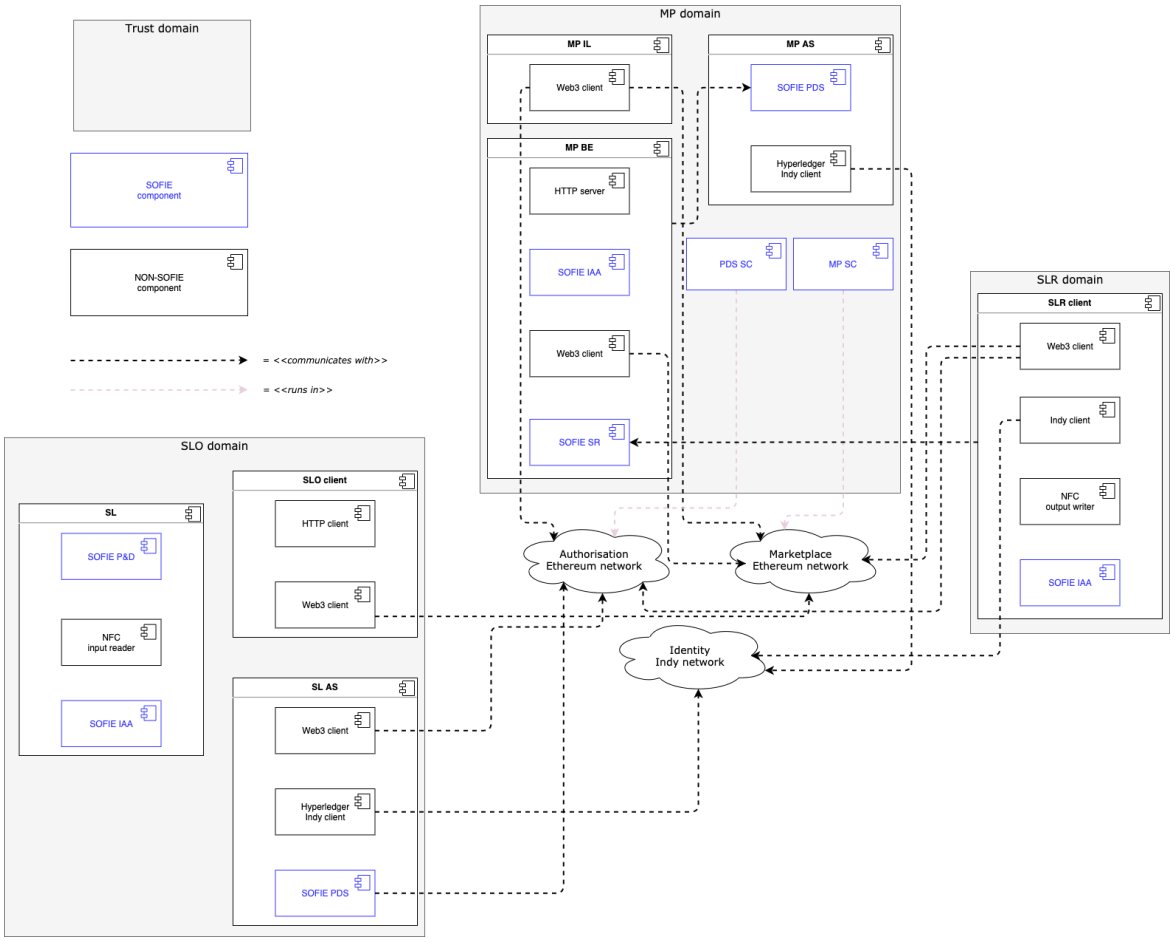

Figure 3.1.a: SMAUG architecture. The system is composed of three trust domains, i.e., the system components that each party trusts and/or manages directly. At the left is the smart locker owner (SLO) trust domain. At the centre-top is the marketplace (MP) trust domain. At the right is the smart locker renter (SLR) domain. At the centre, the three clouds represent the three blockchains that the system relies upon to provide its services: an Ethereum blockchain to run the marketplace, an Ethereum blockchain to manage authorisation-related information, and an Hyperledger Indy blockchain to manage the identities of SLRs, SLOs, and MPO.

MP domain

The marketplace (MP) domain includes components that are directly run and managed by the marketplace owner (MPO), or that are trusted by the MPO that relies on them to achieve some tasks. Specifically, the MP domain is composed of:

- **Backend (MP BE):** the backend is used by smart locker owners (SLO) to manage their smart lockers (SL). Specifically, through the backend, the SLOs can register new SLs and can manage their status (e.g. publishing requests on the marketplace, or deciding the winning offers for a given request).
- **Authorisation Server (MP AS)**: the authorisation server manages access to the marketplace platform. The MP BE relies on the MP AS to authenticate users and grant them access to the platform. Authenticating users allows the MPO to track usage of the marketplace by SLO and SLR(e.g. how many requests SLOs have created, or how much money they have obtained from marketplace transactions).
- **Interledger (MP IL)**: the interledger bridges the communication between the marketplace blockchain and the authorisation blockchain. Specifically, the interledger will notify the authorisation blockchain whenever a request is closed and the winning offers decided. Similarly, parties operating on the authorisation blockchain, after

performing some actions in response to the event received from the marketplace blockchain, can interact with the interledger to send response data back to the marketplace blockchain.

SLO domain

The smart locker owner (SLO) domain includes components that are either owned and directly controlled by the SLO, or that are trusted by the SLO to provide the agreed services. Specifically, the SLO domain is composed of:

- **Smart Locker (SL)**: the physical resource being rented and purchased on the marketplace. They offer a storage space service to authorised users for the duration they have purchased. The presence of an SL is advertised via Bluetooth Low Energy (BLE), such that interested potential renters (SLRs) can discover them using a BLE-capable mobile phone running a compatible application. All the communication between the SL and the SLR's mobile device takes place via NFC technology. At the time when SLRs wish to use an SL, they need to prove their authorisation by presenting a valid attestation that the SL can verify and validate.
- **Web client (SLO client)**: runs on the browser of the SLO's device, and allows the SLO to perform SL management operations. To access the management interface, SLOs must authenticate themselves and must be authorised to perform the required operation. Furthermore, the SLO client allows the SLO to directly interact with the marketplace blockchain to perform SL management operations.
- **Authorisation Server (SL AS):** this authorisation server manages access to one or more smart lockers (SLs). It includes an agent listening on the authorisation blockchain for interledger events, and in response to those events logs an access token that the winning users of that specific request can use to access the smart locker they have purchased access for. The SL AS does not have to be directly managed by the SLOs (although nothing prevents them from doing so), but can also be used following an as-a-service model, where the SLO delegates the management of one or more SLs to the SL AS.

SLR domain

The smart locker renter (SLR) domain includes only the mobile device (defined as SLR client in Figure 3.1.a) that a potential SLR uses to discover nearby smart lockers (SL). The mobile device purchases access for a specific time frame and interacts with the SL to access its enclosing storage space. The **SLR client**, therefore, allows SLRs to discover nearby SMAUG-compliant SLs using BLE, interact with the Ethereum marketplace where access to the SL can be purchased, and interact with them using NFC.

3.1.2 Usage of SOFIE components

Following is a description of how the different SOFIE components are used within SMAUG, and what benefits they bring.

3.1.2.1 Marketplace

The SMAUG marketplace extends the functionalities of the SOFIE Marketplace component by adding support for instant-rent offers and Interledger sender/receiver support. The instant-rent support makes it possible for the smart contract to automatically close and decide requests whenever a valid instant-rent offer is presented, thus discarding all the auction offers and triggering the interledger process for the winning offer. The entire process is performed automatically and does not require nor involve the smart locker owner.

From a technical point of view, extending the features of the SOFIE Marketplace component implies subclassing some of the SOFIE Marketplace contracts to contain the additional SMAUG-specific logic. The SMAUG marketplace backend has been developed on top of the

SMAUG smart contract to implement a layer of access control, as the creation of new requests on the marketplace requires the request creator, the smart locker owner, to also specify a token that must be signed by one of the manager accounts of the marketplace. This represents the second key difference of the SMAUG marketplace smart contract compared to the SOFIE Marketplace component, which has no knowledge of access control beyond what is enforced by the Ethereum network. A high-level representation of the flow is given in Figure 3.1.b.

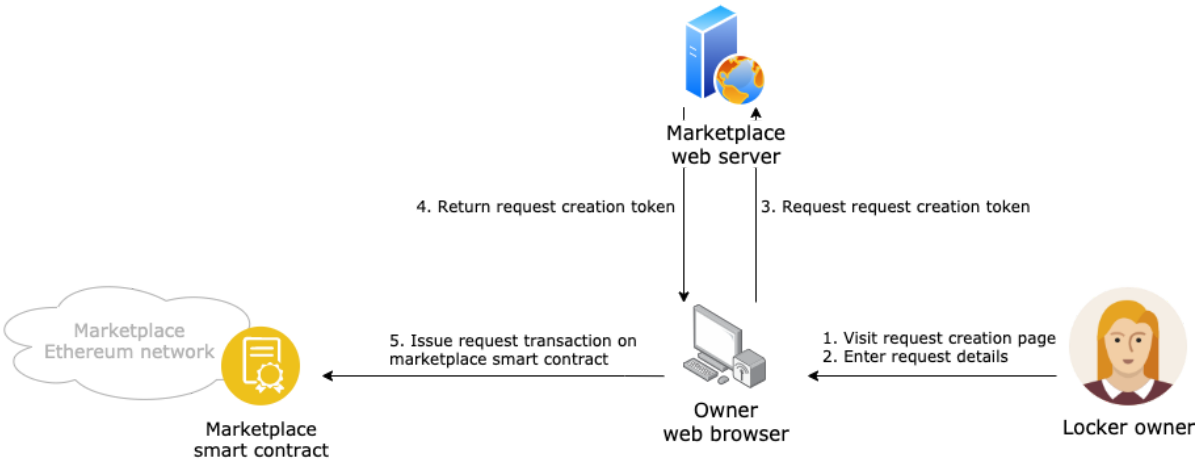

Figure 3.1.b: the flow to create a new request on the marketplace. Notice that smart locker owners are required to retrieve a new access token from the marketplace backend before being able to create a new request on the smart contract.

3.1.2.2 Interledger

The SOFIE Interledger component has been integrated without extension of its functionalities. It has only been parameterized for the SMAUG use case, including the addresses of the smart contracts on the marketplace and authorization blockchains, and the accounts used to issue the needed transactions to perform the Interledger procedure. Figure 3.1.c shows the flow of the Interledger protocol that is triggered when a request on the marketplace is closed and decided (i.e., when a winning offert is selected).

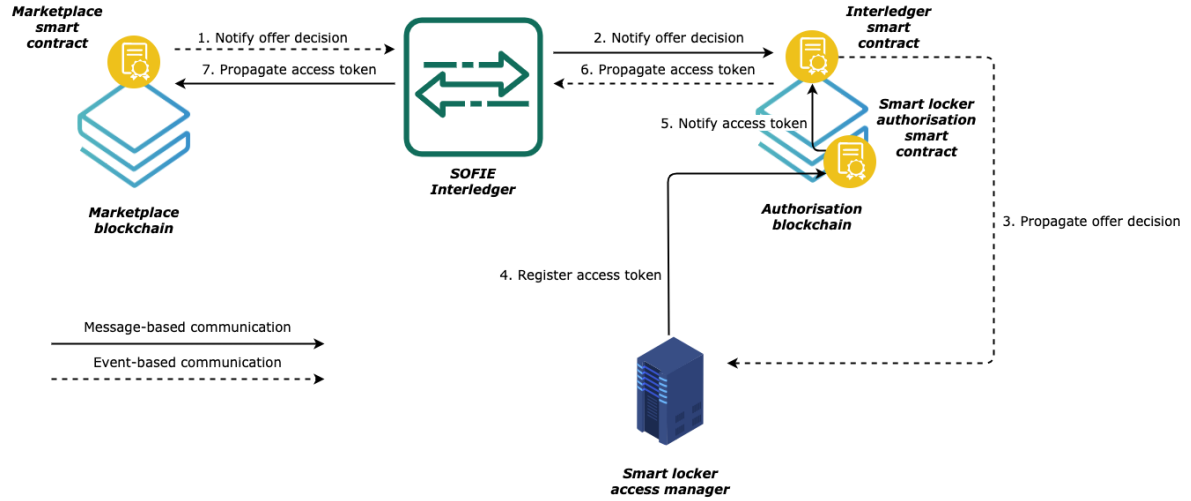

Figure 3.1.c: Interledger flow for a typical marketplace transaction. First, when an offer is decided, an event is emitted to start the Interledger procedure (step 1) which is captured by the Interledger agent. Then, the agent calls a smart contract on the authorisation blockchain (step 2) which, in turns, emits an

event to notify potential listeners about the Interledger operation and its associated data (step 3). Interested listeners can then perform custom actions, such as registering access token (step 4) and, if they need to propagate the result of the action back to the marketplace, interact with the Interledger smart contract (step 5), which then propagates the information (step 6), which is captured by the Interledger agent and forwarded to the marketplace smart contract (step 7).

3.1.2.3 Identity, Authentication and Authorisation

The SOFIE IAA component is used for access control by the marketplace backend and the smart locker. In the first case, a Docker container running IAA is accessible only by the marketplace backend container, which delegates verification of JWTs that smart locker owners present when accessing the marketplace APIs. In the second case, the IAA Python executable is run directly on the smart locker platform and allows the smart locker controller software to delegate validation of the access token presented by customers willing to physically access the smart locker space.

3.1.2.4 Privacy and Data Sovereignty

The SOFIE PDS component has a dual function. On the one hand, it enables marketplace backend support for DID-based authentication of smart locker owners. Specifically, when smart locker owners need to renew their JWTs, which are verified by IAA, they need to authenticate using a DID previously registered on the marketplace. Hence, the marketplace backend delegates to PDS, in the form of a Docker container accessible only by the backend, the generation and verification of challenges as well as the generation of the JWTs the users need to use the marketplace platform. The second function of PDS is to generate access tokens for winning offers on the marketplace. To do this, a PDS smart contract has been deployed on the authorization blockchain, and a PDS Docker instance has been deployed as part of the smart locker access manager component. Specifically, the access manager listens for interledger events broadcasted by the Interledger component on the authorization blockchain, and then delegates to PDS the generation of the access token. The PDS, in turn, logs the generated access tokens on the PDS smart contract. When all access tokens have been generated and logged, the smart locker access manager triggers the interledger process back to the marketplace blockchain with the information about the newly generated access tokens. The process is triggered by the PDS smart contract, which in turn interacts with the IL smart contract deployed on the authorization blockchain. A graphical representation of the process is given in Figure 3.1.d.

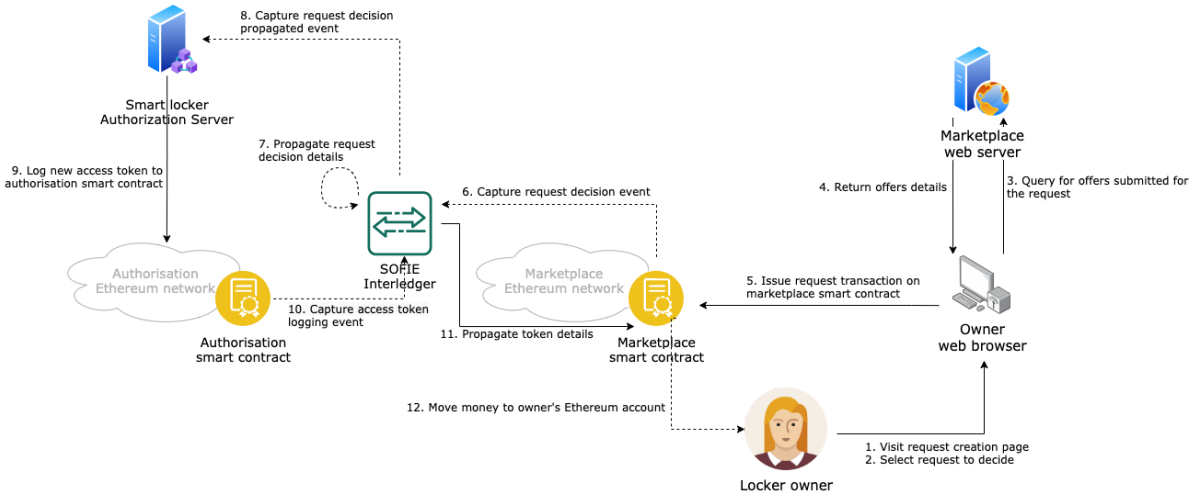

Figure 3.1.d: The complete flow triggered upon request decision. For simplicity purposes, the PDS smart contract has not been included in the diagram, only the IL smart contract that is deployed on the authorization blockchain (and that IL interacts with at step 10).

3.1.2.5 Provisioning and Discovery

SMAUG utilizes only the *discovery* functionality provided by the Provisioning and Discovery (P&D) component. SMAUG-compliant smart lockers will use this component to advertise their presence to nearby users using Bluetooth Low Energy (BLE) as the communication medium. The P&D component has been integrated into the smart locker, configured to fit SMAUG's needs, and run as a Python executable.

3.1.2.6 Semantic Representation

The Semantic Representation (SR) is the key component that makes SMAUG open and interoperable with external systems. The data that each smart locker advertises include its physical properties (e.g., capacity, identification number) and information about the marketplace that potential renters will interact with. The SR is deployed as a Docker instance accessible only by the marketplace backend, which delegates to the SR validation of smart locker related information, based on the SMAUG semantic representation description file which follows W3C WoT TD standard.

3.2 Food Supply Chain Pilot

In the Food Supply Chain (FSC) pilot, three different IoT platforms are federated to establish a distributed and immutable data management layer that provides traceability and quality control services for transported products. This makes traversing the path from field to fork more robust, reliable, and time-efficient for all parties involved in the food supply chain.

3.2.1 Pilot architecture

In the Food Supply Chain (FSC) pilot SOFIE's Framework Interledger (IL), Semantic Representation (SR), Privacy and Data Sovereignty (PDS), Identity, Authentication, Authorisation (IAA) components have been used, as well as Federation Adapters (one has been developed for each IoT platform). Figure 3.2.a below depicts how these components are connected with the pilot platform.

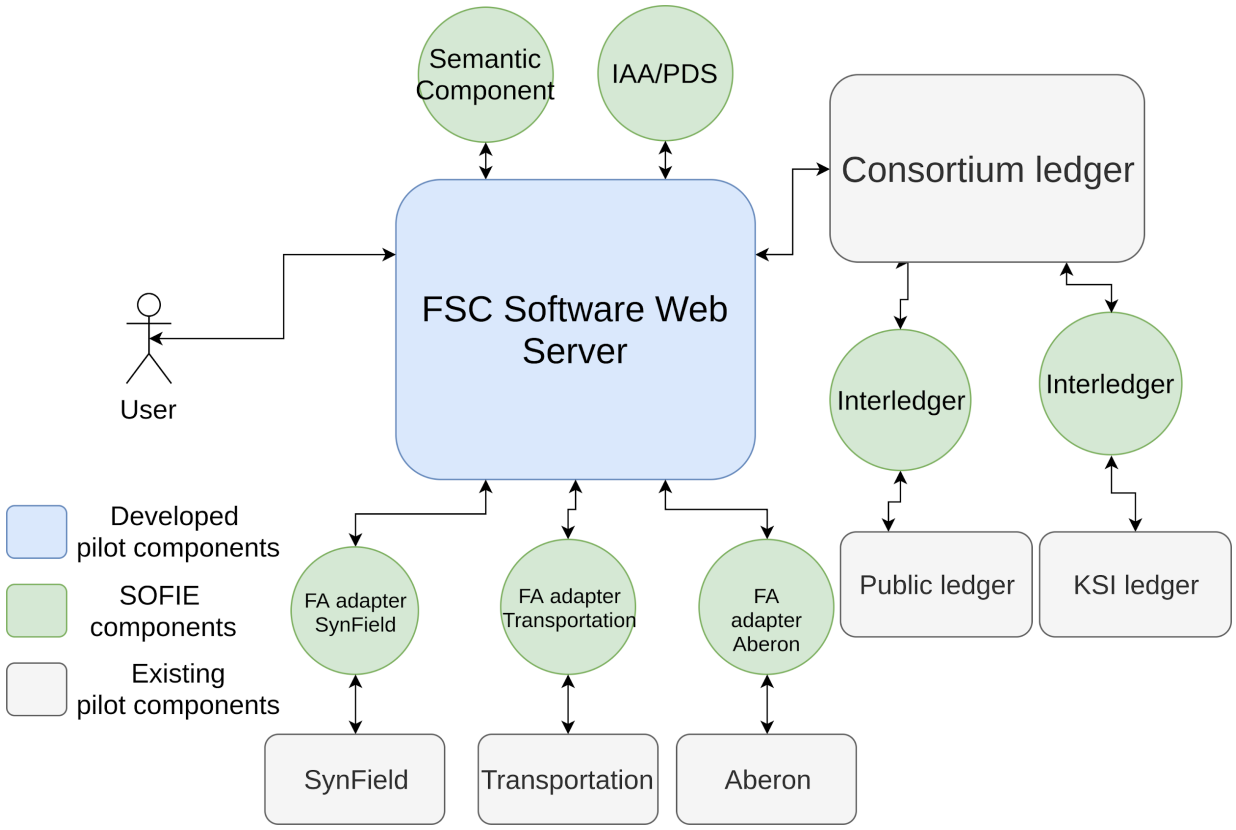

Figure 3.2.a: Food Supply Chain pilot platform architecture and SOFIE components' relation.

3.2.2 Usage of SOFIE components

The following sections provide a description of how the different SOFIE components are used within the FSC pilot, and what benefits they bring.

3.2.2.1 Interledger

The Interledger component of the SOFIE Framework is of significant importance to the FSC pilot. Its interaction with the rest of the pilot platform is described in the following subsections.

3.2.2.1.1 Ethereum to Ethereum integration

The IL component is used for Ethereum to Ethereum (eth-eth) integration, where the source chain is the Consortium (private) blockchain, and the destination one is the public testnet Ethereum network (Ropsten).

The IL component uses two adapter smart contracts (one for the source and one for the destination in order to facilitate functionality). The adapter smart contract for the Consortium blockchain can be found in SOFIE's repository [FSC private]. The adapter smart contract for the public blockchain can be found in SOFIE's repository [FSC public]. The interledger payload (Figure 3.2.b), transported from the private blockchain to the public one, is the set of the hashes of every step in the FSC along with the box & session id.

{
'box_id': 'E280113020002894DC0A00AB',
'session_id': '0x711789efa2b7d2c83f59dd15d073afb8e30b7cda9b1043f825a939e62ee0b256',

```
 'signatures': [
        '0xc90996e6355f2579b7db3efb8c2556896f4215e1f6b28d76ffe6e5d2a44db5ce',
        '0xadab9156376065aabc73b88b71de77c0e2a47349164a4c1d1180eb1645072aae',
        '0x5c834af27871086bf6d30c6bf580434e1de1bcfba5387fcc32b56a589c40b661',
        '0x4f0b59e66e3647c0119a72823e7e9c0d2d526715672e69703f822ec5859c54b6',
        '0xc5c9ee36c93fbb42a60e4cf2176f00bed4e1fb30a6bf03fb576b5e8c59ab9e6f',
        '0x95c1d346a6dd06c493f78ec871888c347d0634076cf433e8f4f4a13b29f955f4',
        '0x4a0b14e1e1124a8a25a6adbd0d1678067f1a73d0e174496563086f9a7d75cefc',
        '0x530165dbc3f9f2e4eb9ef726a63247ce7f28db415957707160fdeb5214466ddf']
}
```

Figure 3.2.b: Interledger Payload.

Figure 3.2.c shows the interledger eth-eth configuration:

```
[service]
direction=left-to-right
left=private
right=infura

[private]
type=ethereum
url=http://192.168.1.117
password=pwdnode
port=7545
poa=True
minter=0xa4DFb027FA681d0c6ef3ab46DaBc73bB7C2DF48E
contract=0xae18bc02e1F9821620BECeb8715962aDf2A105C7
contract_abi=fsc/FoodChainAdapter.abi.json

[infura]
type=ethereum
url=wss://ropsten.infura.io/ws/v3/c96300b21ee04e0795403a181a08a73f
private_key=0x0e98b5148c9a1028e530d87ce942300c3820f8a7761f520d295058d21ba8c7cb
minter=0xE2C676d11dD3af0404991b71781c599D89356B30
contract=0xac8af7df94EeDc1b050663844c1e2be37B060036
contract_abi=fsc/SofieFSCAdapter.abi.json
```

Figure 3.2.c: Eth-eth configuration.

In the scope of the consortium ledger, every step of the FSC is recorded along with valuable information about each step of the chain.

3.2.2.1.2 Ethereum to KSI integration

The IL component is also used for Ethereum to KSI integration, where the source chain is the Consortium (private) blockchain, and the destination one is the KSI public ledger. The KSI FSC Pilot integration happens at the final stage of the supply chain. When a box reaches its destination (the supermarket), the details of the box supply chain history are hashed and passed to the public Ethereum ledger (Ropsten). The FSC smart contract in the public ledger collects the hashes that are sent via the Interledger Component to create a master hash value that combines (merges) the hashes of all stages of the food supply chain. When the master hash is generated (e.g. the transaction is placed in a block inside the public blockchain) the supervisor component collects the generated event and makes a KSI transaction containing the generated hash. The KSI transaction is performed with another instance of the Inteledger Component, configured between the Consortium blockchain and the KSI public ledger. When the KSI transaction is complete, the Inteledger component performs a transaction inside the consortium ledger via the FoodChainKSIAdapter smart contract. A

LogInterledgerEventCommitted (as per the interledger component spec) event is created containing the generated KSI uuid. The supervisor then stores the KSI uuid for further reference during product quality evaluation. An example KSI entry is shown in Figure 3.2.d:

```
curl -X GET --header 'Accept: application/json' --header 'Authorization: Basic {code}' 'https://tryout-catena-db.guardtime.net/api/v1/signatures/3841e0ad-86af-4ecc-afec-30434e59ddc8'

Response:
{
  "id": "3841e0ad-86af-4ecc-afec-30434e59ddc8",
  "signature":
"iAAH2ogBAFMCBF9GYg0DAQsDAgGdAwELAwERAwEDAwEDBSEB07pvSJKPYIL+Ix91m674+iY9
gQ/
i5d9alcXBoVJv7D0GAQEHEgQQfgIBAQEKb3QudmRjdXRyAIgBAGYCBF9GYg0DAQsDAgGdAwEL
AwERAwEDBSEBBkckDA6fzq+AwYBrhe5K9YWdhlLsWUQIualiE8kfL3sGAQEHKAQmfgI
BAWEGa2F0amEAYg10cnktY2F0ZW5hOjEAYwBkBwWtx7WaQQeIAQDOAgRfRmINAwELAwIBnQ
MBCwMBEQUhAbsm6hmZr0WR/CbpK8P8LFJXQVkTz/
pLWh0FiLXoYgJsBgEBByQEIn4CAQFhA0dUAGILQUxlMi0xLTI6NwBjAQ5kBwWtx7WfCNQIIwIhAaI
Wuu0Fd2LqmTI9pH1gjkGLe8nwPBppX599QSJx/
f7aCCMCIQEufMdK6eIJ53CvtSlMoBbzPlBYZ/RQXCfhoWGggsujRwgjAiEBQjTqPnQLu3sgQmMjRy1
vKmQA3gR7aa6Z5P+zxLF/
36CIAQCnAgRfRmINAwELAwIBnQMBCwUhAXFmjf5+uYmr7qIangXve4i4Cz+yI33ENNtDXpHhzNr/
BgEBByIEIH4CAQFhA0dUAGIJQVNlMi0wOjEAYwEBZAcFrce1pPTkByYBAQECIQHuUnjFbMzRR6i
anVqcbSf8G5mDRprfy3Zm3VcgaDTj0QgjAiEBkqTemUkIRtYKwr+BoUOTDtzmQsGB
mQpLmJia/
77UKMWIAQFbAgRfRmINAwELAwIBnQUhATA04uIWex9wiwJXY3upz1TBR3XPMC2ZpxrgFsDbJb+
QBgEBByAEHn4CAQFhA0dUAGIHQU5lMjowAGMBBmQHBa3HtaU8WwgjAiEB7UnSMQshB
UbNke5YRsk+EIRjUZMeHsiPXpRrCVCeQxQHIwIhAZPsOQmOvnMyZ1p6s1c56ZF6nisycskECuNty
+FtNhwKByMCIQHZZ5RVwa9/iG50eXD1WWH2RtjedLAXKI951X/
AiQHnJwcjAiEBYCK2FYmHuEWCSZtowd3oMkgz6c1XjAXAypYuEc/dD84IIwIhAfuhOPeft7qpADSYQ
yNJmI0XSSDYBV/
a99h4vfYKACMICCMCIQHbTuI1k5r4K+JNnK7GPbc98TEcAFVLDSM51jsFHqNuTAcmAQEBAiEBN
FbFDIKSNbSKRLqWfWVqpb2fOIiLuQbHwnydA85jjAmIAQCkAgRfRmINAwELBSEB1kD
0b9NDCin4vapGx64pBnqKYrRCcfAirYfq3+EUI7IGAQEHJgEBPwIhAQAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAByYBASwCIQHkiubCUX2uuCvgR8oAIhNPtr/
XwVt0XBAR4rrRIHRXjAgjAiEBkCYSYaoJj5PZJ5woc2JDig+T8HA2j52OJstJbAV0E4qIAgI8AQRfRmI
NAgRfRmINBSEBcZQA5e/2VAOHiaayFkwZoiz7kMs2QpSC/hLbqpGj/
EcIIQGw/hhtRV8j7XebAyWmPMEQpQKn6luBbXZ/GwhcWYeZjwghAR3nAXgk5yS1jVeQFWaIzpYpM
8sKfogXYuhFi11NvoP0CCEBB6XzTWnOv9ulIE8NFpbtkp0vjc/h0JDQO6E/
PVpyKAoIIQHS9B8xONz4b8g/plYjSC+Z8ECSZOU8UQdmiekL4/vITAghAYJHtvGM2tnwc1Frs2kT2cd
K1kOQna/swOo8by85UknACCEBU/
zq8+ecX8zAoqN4C59WiVSvOwQdo5MqD09/8dkBh4gIIQFV3SL5K6veWVOfEPZbVpbkUtE03Ojpz9
T9yr65busj3QghAYVTvOLI/
UWnEqoyz5xE8WITUIf1ehMnOJ+ezw5bPWaTCCEBHazb165CHNaZC5c/PzzQcHpqAuRbiHf+Eet0
SXmo5AUIIQGHQy9MH2LCiBI+qg/7XRTg3WsYP3xHQSL5bAxQmMrw7QghAas/
GScAfb6ffBiauw8OPdwdAmV49wu2+5gOHbh698gVCCEBVre3MhAq26G12NxCAkvHKKygFC81aLn
rZUdQeYa2UNcIIQHrw6sdhmQVgRMKw8cHe3G2e7pMkVUw6f5JuYdp/
I3K6wghAUIvwBINhU51NLmSqzLsMEWyDUvuG/vkVk/Qks6voItyCCEBu0T9NqXzze57XG3zpgmKC
eNTM1tgKfFHdQJYin43vgCIBQFRMCkCBF9GYg0EIQGtvB79k34CMpWn10/e1/
wLfRILF7rPBQZJTC9p3gn9NIALASIBFjEuMi44NDAuMTEzNTQ5LjEuMS4xMQCAAgEALsQEom7Xr
A0f8lI55jfyRtEu03+Q+25wYy0o34XIkbRoNbv4QFgo1y689fcJOvzFL6tfGdS5KT
5bA7QIChe8EqrrOZswAgTdq5wOZxJB0ymlwXOHoq3MdYCJS/K0zLx8k5DawPxckzd9XMCDKsa6g
Tr1BB/
zZOyRZxxYjcNcej5Rl8tBR97fX2ag19ksI91FV8OZvhWGGha4IzPlJgz1ab5BMNNou7y+hAlWikeKN5
NSH9VQsbxv8lUvxb/
k5ddwNOauRM92T8lfAsqQUBn5JJhkED5bOwmC1/7ruri7W31z6uApdR9qrcprd+tndjICbmfCLW8xJn
cJLzojAKv1VAMEBaG2MA==",
```

```
  "details": {
    "dataHash": {
      "algorithm": "SHA-256",
      "value": "07pvSJKPYIL+lx91m674+iY9gQ/i5d9alcXBoVJv7D0="
    },
    "aggregationTime": 1598448141000,
    "identity": "GT :: GT :: GT :: katja :: ot.vdcutr"
  },
  "metadata": {},
  "createdBy": "ot.vdcutr",
  "createdAt": 1598448142069,
  "verificationResult": {
    "status": "OK",
    "policyResults": [
      {
        "policyResultCode": "OK",
        "policy": "KEY_BASED_POLICY"
      }]
    }
  }
}
```


Figure 3.2.d: Example KSI entry.

3.2.2.2 Privacy and Data Sovereignty

In the FSC pilot, SOFIE's PDS component is used by the Federation Adapters, where each constructs a Decentralized IDentifier (DID) using their Ethereum wallet addresses:

```
wallet_address = '0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773'
wallet_handle = await wallet.open_wallet(client['wallet_config'], client['wallet_credentials'])
client_did, client_verkey =
  await did.create_and_store_my_did(wallet_handle, json.dumps({'seed': wallet_address[2:] }))
```

Figure 3.2.e: Wallet address configuration.

Upon an IoT platform registration, the Supervisor Web Server component configures the PDS with the DID of the IoT platform:

```
wallet_address = '0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773'
client_did = 'Q3f3yDjBJPkxZ8GQjdFK8s'
client_verkey = 'DZXLK6rjZUeq8aguq6wo89DAGdydxqTCunDnBosV246P'
now = datetime.utcnow()
nbf = time.mktime(now.timetuple()) # Not before
exp = time.mktime((now + timedelta(days=1)).timetuple()) # Expiration time
aud = 'sofie-iot.eu' # The domain name of the protected resource
payload = {
  'action': 'add',
  'did': client_did,
  'verkey': client_verkey,
  'metadata': json.dumps({'aud': aud, 'nbf': nbf, 'exp': exp, 'sub': wallet_address})}
response = requests.post("http://pds:9002/", data=payload).text
```

Figure 3.2.f: PDS configuration.

The FA adapter of each IoT platform completes the challenge - response interaction via an endpoint (http://pds:9001/gettoken) and it receives a JWT token:

Example token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiJzb2ZpZS1pb3QuZXUiLCJzdWIiOiIweGIwMDU3NzE2ZDU5MTdiYWRhZjkxMWIxOTNiMTJiOTEwODExYzE0OTdiNWJhZGE4ZDc3MTFmNzU4OTgxYzM3NzMiLCJleHAiOjE1OTU1NzM4MjAuMCwibmJmIjoxNTk1NDg3NDIwLjB9.rVOfnYVsl-dgQlopkv57obKERGakFlmnvDZTZhLMIt5tenfggSyW9LUpazrSrVu_C334WbuiUjKt2NYCOeW0KButyusRRpT6Ho3n8MSDU9wUoEUOie4nTdlyXYrnuRWmQMY9oDWjd9zvijf3Q98_-eZHoYFVM7TD_4qvNInUkEoSIlKqOG40Zf4PSTRw6Z6CWrIg3y2p707Ioznsy5FK179lyK4SF9DLXh1iNtSQ7wJjUDe0GwdJ8gSOa4fDksWFHmVWL9QVwdFJPJLXIk8c-gM2awIRlNzAfX9A86exKHBAWrsrp8RT5zHtuKDO2fW0veBXDodpmRk5PyaAUZN7Bw

Token decoded:
Header:
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload:
```
{
  "aud": "sofie-iot.eu",
  "sub":  "0xb0057716d5917badaf911b193b12b910811c1497b5bada8d7711f758981c3773",
  "exp": 1595573820,
  "nbf": 1595487420
}
```

Figure 3.2.g: PDS Token example.

When the Supervisor Web Server component interacts with the FA, the FA attaches an HTTP header, containing the JWT token along with its payload:

```
HTTP_AUTHORIZATION = Token <jwt_token>
```

PDS interactions with the FAs and other components are depicted in Figure 3.2.h:

3.2.2.3 Identity, Authentication and Authorisation component

As mentioned in 3.2.2.2, the FA uses an HTTP header which contains a JWT token when interacting with the Supervisor Web Server component. The Supervisor component invokes the IAA component to verify the JWT token against the given FA wallet address:

```
payload = {'token-type':'Bearer', 'token':jwt}
response = requests.post("http://iaa:9000/verifytoken", data = payload).text
```

If the response is HTTP_200 and the wallet address which was embedded during the PDS configuration steps (please refer to 3.2.2.2, “payload.metadata.sub” field) matches the wallet address retrieved by the ECRECOVER Ethereum algorithm, the DA payload is considered valid and secure.

IAA interactions with the FAs and other components are depicted in Figure 3.2.h:

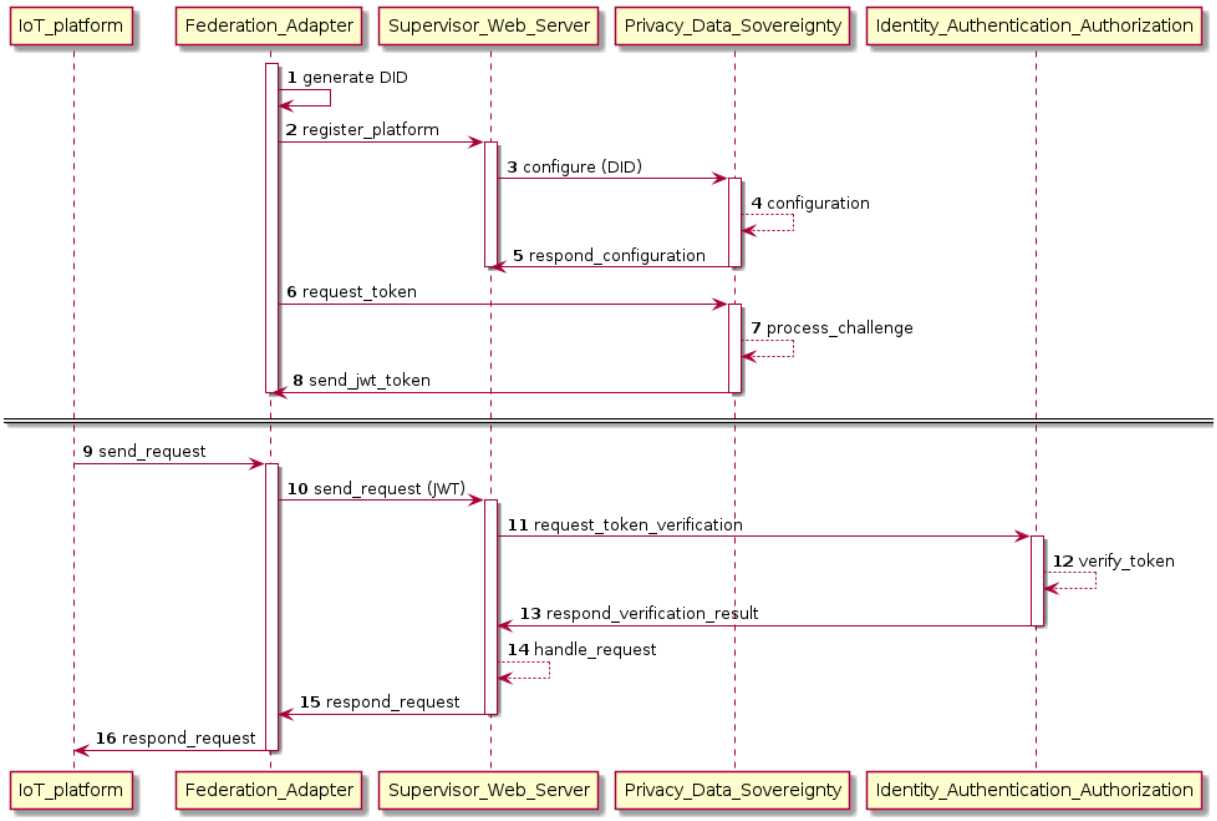

Figure 3.2.h*: PDS/IAA interactions in the FSC FA.*

3.2.2.4 Semantic Representation

In the scope of the FSC pilot, the Semantic Representation (SR) component of the SOFIE platform is used in the Federation Adapter (FA) - Supervisor Web Server (SWS) communication. The FA of an IoT platform participating in the pilot has to provide an endpoint (usually the base endpoint / of the FA API) that once accessed via an HTTP GET call, outputs the schema of the FA. The Supervisor component reads the schema, retrieving information about the endpoints and functionality provided by the FA for the underlying IoT platform.

When a new IoT platform is registered via the Web Application by the pilot administrator, and the Ethereum transaction is mined, a *LogPlatformRegistered* event is triggered. When the SWS acknowledges the event, the SR component is invoked on endpoint /api/add_schema and the FA schema is registered with the Semantic component. The SWS component then retrieves the schema of the given platform and via an HTTP POST request to the SR component to the /api/add_schema the platform is added. An example of POST payload is given in Figure 3.2.i:


```
{
  "name": "TransportIoT",
  "schema": {
    "name": "TransportIoT",
    "schema": {
      "@context": "https://www.w3.org/2019/wot/td/v1",
      "title": "TransportationThing",
      "id": "urn:dev:wot:com:sofie:fcp:adapter:transportation",
      "description": "Transportation Federation Adapter Thing Description model for the
      Food Supply Chain pilot",
```

```
    "securityDefinitions": {
      "nosec_sc": {
        "scheme": "nosec"
      }
    },
    "security": "nosec_sc",
    "properties": {
      "transports": {
        "type": "array",
        "readOnly": true,
        "description": "The array of transport ids served from the underlying Transportation IoT
platform",
        "items": {
          "type": "string",
          "description": "The transport id in hashed/encrypted form"
        },
        "forms": [{
          "op": "readproperty",
          "href": "https://192.168.1.167/transportation/api/transports",
          "contentType": "application/json"
        }]
      },
      ...
    }
  }
}
```

Figure 3.2.i: SR schema.

Upon every SWS communication with the FA, the FA payload is validated via the SR component for the given schema. The SR component /api/validate endpoint is invoked. If an IoT platform is removed from the FSC pilot platform, a *LogPlatformRemoved* Ethereum event is generated. The SWS acts on this event to inform the SR component of platform deregistration. The SWS component invokes the endpoint *api/remove_schema* of the SR component.

The interaction between the IoT platform, the FA, the SWS and the SR components are show in Figure 3.2.j:

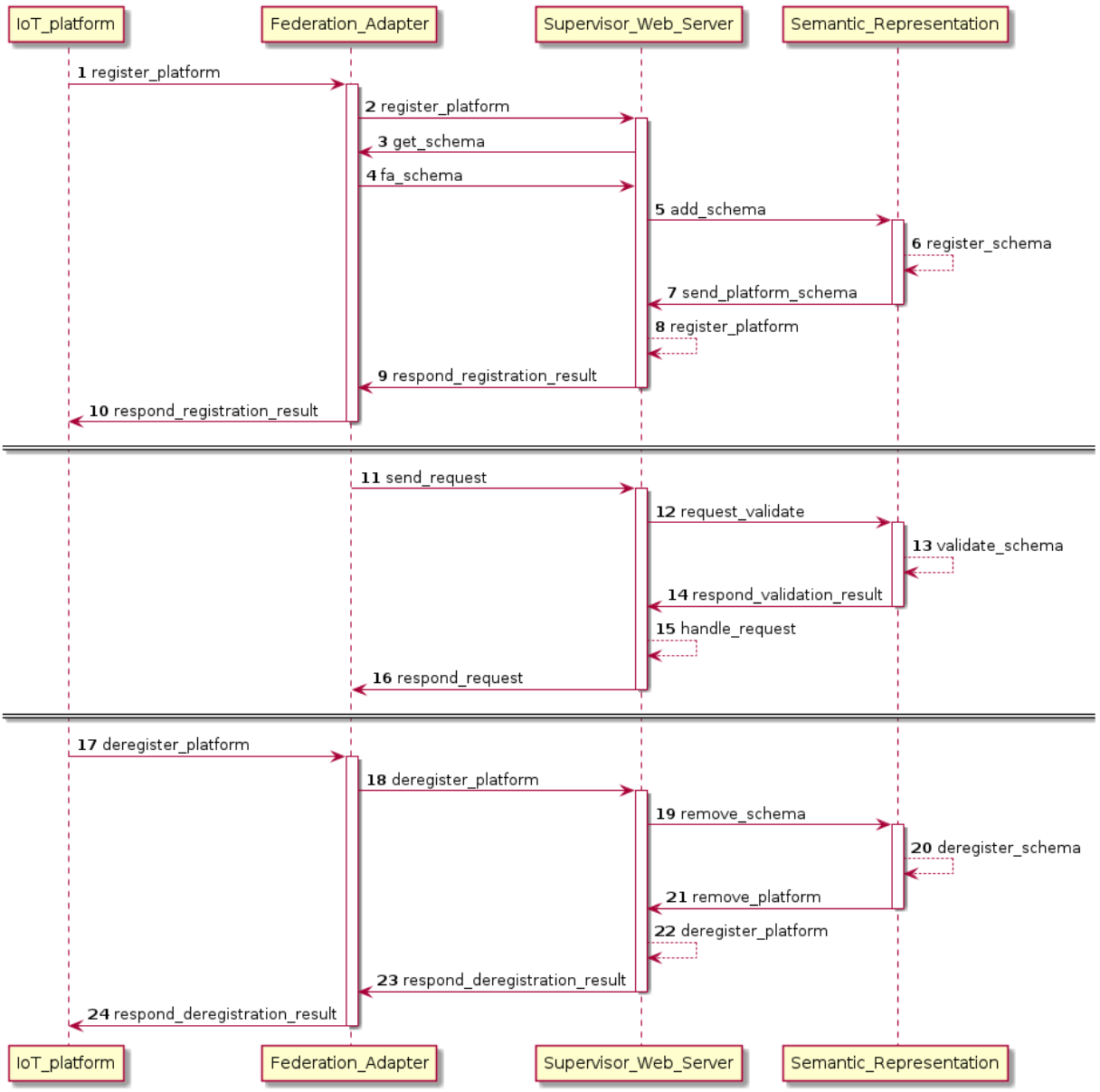

Figure 3.2.j: Semantic Representation interactions in the FSC FA.

3.2.2.5 Federation Adapters

The Federation Adapters in the FSC pilot are three, one for each IoT platform (SynField, Aberon, and Transportation). One of the FAs (the one developed for the Transportation IoT platform) is publicly available on the projects repository [FSC adapter]. A FA interaction with the other SOFIE components as well as the pilot's platform modules is described in the previous subsections in this document.

3.3 Decentralized Energy Data Exchange Pilot

The Decentralized Energy Data Exchange (DEDE) pilot is a proof-of-concept for secure data exchange and management of access rights between smart meters, infrastructure owners and energy service providers (intermediaries, distributors, brokers).

3.3.1 Pilot architecture

The DEDE pilot connects data providers with data consumers. Both the data providers and the data consumers connect to the platform through their own instance of the Federation Adapter (FA), which is a common software component for both sides and needs no extension or customisation.

As shown in Figure 3.3.a, the FA of a data consumer connects directly to the FA of a data provider to exchange messages according to the FA communication protocol. Messages between the two FAs are transported securely over a mutually authenticated TLS connection, using Hyperledger Indy-based decentralised identifiers (DIDs) and verifiable credentials (VCs) to establish trust.

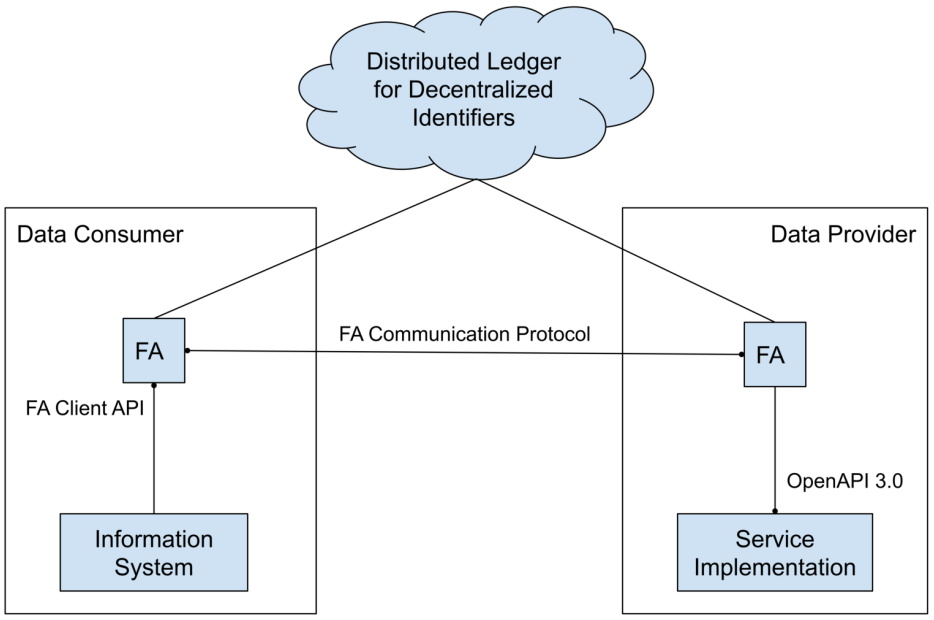

Figure 3.3.a. The architecture of the DEDE pilot.

3.3.2 Usage of SOFIE components

All the SOFIE components utilised by the DEDE pilot have been integrated in the FA. To understand the role of the components, we need to first look at the internals of the FA.

The main function of the FA is to ensure interoperability and to secure the communication with other entities on the platform. It acts as a forward proxy for the data consumer and as a reverse proxy for the data provider, but it can also be in both roles at the same time, enabling entities that are both data consumers and data providers. The FA takes care of the security aspects of the integration and lets the data provider concentrate on implementing services and

the data consumer on using these services. The only requirement for a data provider is to describe its services according to the OpenAPI 3.0 specification.

From the implementation perspective, the two main responsibilities of the FA is to proxy messages and to manage the identity of the represented entity. The internal structure of the FA shown in Figure 3.3.b mirrors this with two loosely coupled services: proxy and ssi-agent (Self-Sovereign Identity agent). Both of the components have public and private interfaces, for external and internal use accordingly.

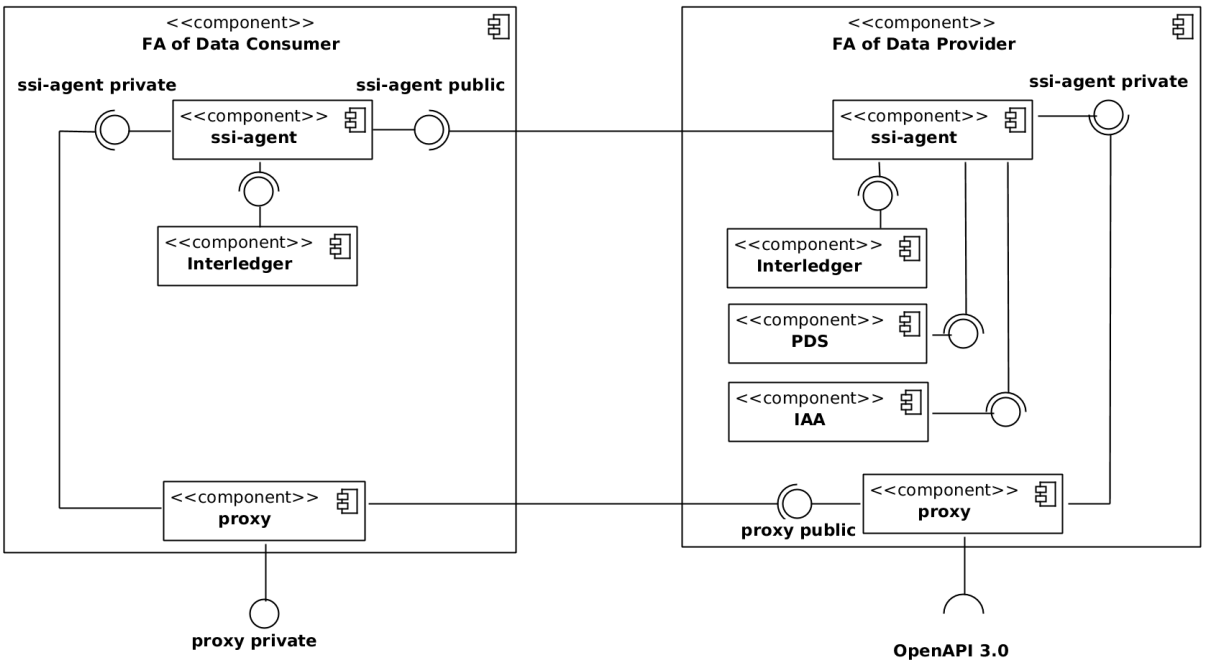

Figure 3.3.b. Internal structure of the DEDE FA.

The information system of the data consumer first sends a request to the proxy's private interface. The proxy uses the ssi-agent private interface to resolve the endpoint of the target DID and to sign the request with the source DID. Then, it initiates a secure connection to the public interface of the data provider (target DID) proxy. Both sides use the ssi-agent private interface to retrieve the hash of the currently valid certificate to verify the authenticity of the connection. Once the connection is set up, the data provider proxy will use the ssi-agent private interface to verify the request's signature. If the request is for a service that requires further authorisation, the data provider proxy will also use the ssi-agent private interface to get the proved values of the attributes required for the authorisation decision. If the data provider's ssi-agent receives such a request, it will send a proof request to the public interface of the data consumer's ssi-agent. Once the data provider's proxy has values for all the proved attributes, it can forward the request to the service implementation that is described using the OpenAPI 3.0 specification. The signing of the response and the verification of the response message signature is analogous to the processing of the request.

Following is a description of how the different SOFIE components are used within the pilot.

3.3.2.3 Interledger

Each node can increase their trust for the Hyperledger Indy instance by periodically recording its state in the KSI blockchain. The Interledger component takes care of this. It is not strictly required for the protocol to work, but can serve as an additional tamper-proofing mechanism for private Hyperledger Indy deployments.

The Interledger component must be configured to observe DIDs on the Indy ledger. The configuration (Figure 3.3.c) is done by specifying *indy* as the left side *type* and the direction as *left-to-right*, since writing to the Indy ledger is not supported:

```
[service]
direction=left-to-right
left=left
right=...

[left]
type=indy
pool_name=sofie
protocol_version=2
target_did=<target-did>
genesis_file_path=<path-to-genesis-file>
```

Figure 3.3.c: Interledger configuration.

The ssi-agent component in the FA generates one DID by default and this can be specified as the *target_did* property. The *genesis_file_path* property points to the genesis file of the Indy ledger that is used. In the DEDE pilot, this is the SOFIE Consortium Indy ledger.

3.3.2.1 Privacy and Data Sovereignty

The service providers decide which attributes their clients have to prove about themselves. By default, the protocol expects the attributes to be proven with verifiable credentials, but the PDS component offers a simpler alternative by issuing a JSON Web Token (JWT) that proves certain attributes about the holder of the token. This token can be sent together with a request and thus avoid the extra round trip for proof request.

To configure the PDS component for token issuance, the default configuration can be used (Figure 3.3.d):

```
{
  "port":9001,
  "adminport":9002,
  "wallet_config": {
    "id": "as_wallet",
    "storage_config":{
      "path":"conf"
    }
  },
  "wallet_credentials": {
    "key": "server_wallet_key"
  },
  "only_wallet_lookup": true,
  "as_public_key":"conf/keys/as_public_key.pem",
  "as_private_key":"conf/keys/as_private_key.pem"
}
```

Figure 3.3.d: PDS configuration.

It is necessary to generate the key pair specified by the *as_public_key* and *as_private_key* configuration properties. The *wallet_config* and *wallet_credentials* properties can remain with the default values, as these features are not used when issuing tokens.

3.3.2.2 Identity, Authentication, Authorisation

This is the counterpart component for the PDS component, if JWT based proved attributes are used. If the client sends a JWT together with a service request, the IAA component can verify this token. In this case, the authorization decision can be made without an extra round trip to the client requesting proof of credentials.

To configure the IAA component for token verification, the defaults in the configuration file can be used (FIgure 3.3.e):

```
{
  "port":9000,
  "wallet_config": {
    "id": "server_wallet",
    "storage_config": {
      "path":"conf"
    }
  },
  "wallet_credentials": {
    "key": "server_wallet_key"
  },
  "only_wallet_lookup": true,
  "as_public_key":"conf/keys/as_public_key.pem",
  "target":"sofie-iot.eu",
  "tokens_expire": true
}
```

Figure 3.3.e: IAA configuration.

It is, however, important to make sure that the *as_public_key* configuration property points to the public key file generated for the PDS component. So, this public key file needs to be shared between the PDS component that issues the tokens and the IAA components that verify these tokens.

3.4 Decentralized Energy Flexibility Marketplace Pilot

The Decentralized Energy Flexibility Marketplace (DEFM) pilot demonstrates the applicability of blockchain-based solutions in the context of energy flexibility marketplaces, with special focus on features like smart contracts and fungible crypto-tokens.

The pilot leverages the SOFIE Marketplace component to run a marketplace in which electric Distribution Service Operators (DSO) can pay for the services of Electric Vehicles (EV) fleet managers that, using specific charging stations in specific time windows, contribute to balance the production on the network.

A detailed description of the pilot, including the scenarios and use cases considered, can be found in deliverables D5.1 *Baseline System and Measurements* and following.

3.4.1 Pilot architecture

The pilot platform consists of:

- SOFIE Components

- Pilot-specific components
- Existing services and infrastructures

Figure 3.4.a shows an overview of the platform architecture, highlighting the different groups of software components.

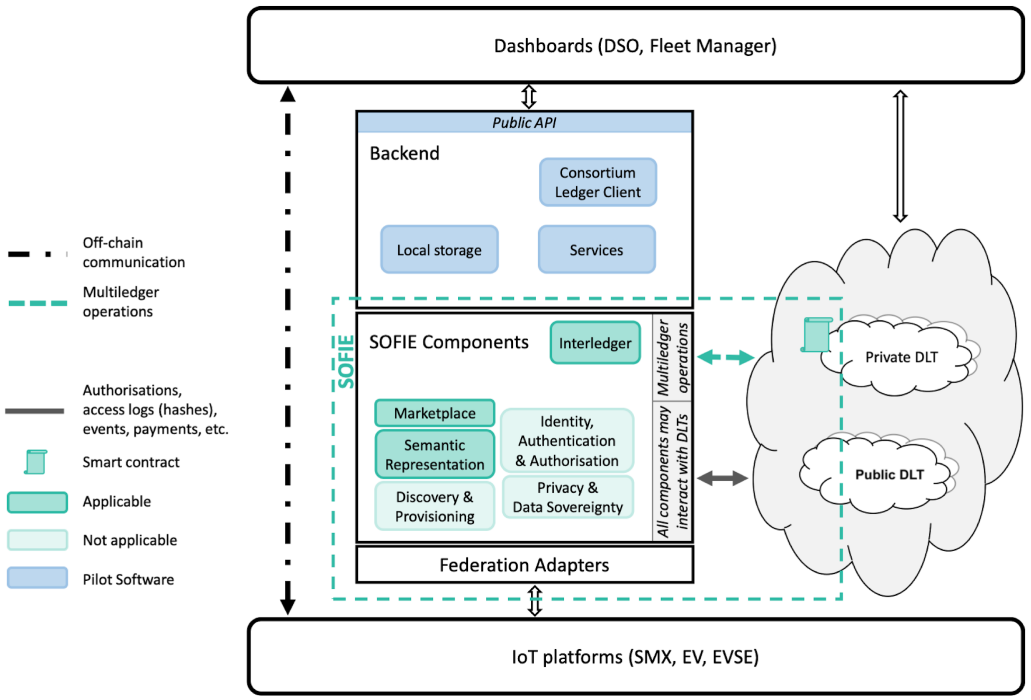

Figure 3.4.a: DEFM Pilot Architecture.

The SOFIE components used are the Marketplace, Interledger (IL), and Semantic Representation (SR) components. Section 3.4.2, below, describes the integration of each component while D2.7 *Federation Framework, final version* describes in detail the components functionalities. The connection with the existing IoT devices is enabled by a specific Federation Adapter (FA).

Finally, the application logic and the interaction with the end users are delegated to the backend and frontend of the web applications used by the DSO and fleet manager operators.

Figure 3.4.b, below, illustrates the integration of the pilot platform. The DSO application consists of two software modules for the frontend and the backend.

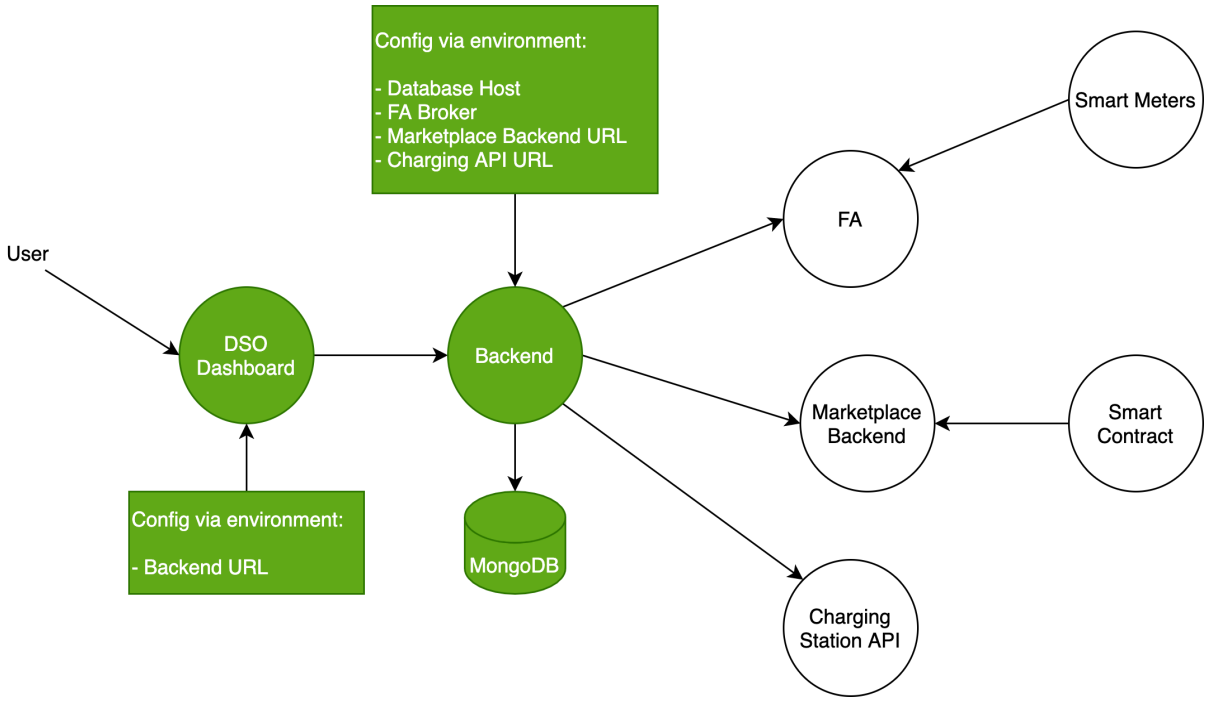

Figure 3.4.b: Pilot Integration.

The application frontend is developed using Vue.js, an open-source JavaScript framework which follows the model-view-viewmodel design pattern, while the backend is developed in Javascript using the Express.js library and Node.js as runtime. Both software modules are developed to run as *containers*: isolated instances virtualized at the OS level that can be managed through specific applications such as Docker. The use of containers ensures that the application is tested under the same environment of a production instance and simplifies the deployment process. To deploy a new version of the application, it is sufficient to pull the current images from the registry and run it using a container orchestration system.

3.4.2 Usage of SOFIE components

Following is a description of how the different SOFIE components are integrated in the DEFM pilot, and what benefits they bring.

3.4.2.1 Marketplace

The marketplace component is the core component of the DEFM pilot. The marketplace includes a set of Ethereum smart contracts and the related backend functionalities and API.

The DEFM marketplace was developed by extending the generic SOFIE marketplace smart contract to fit the pilot's specific needs. In particular, the requests and the offers extend the generic ones by adding the relevant attributes to the energy flexibility use case such as the *network zone* to be used for charging the EV, the *energy* amount requested by the DSO, and the *time window* for the flexibility delivery. The business logic was also adapted to the specific use case: the lowest bid is in fact selected as the winning offer for each request, with the timestamp as a tie-breaker in case of multiple winners and the requests are finalised only after the flexibility requested is delivered by the fleet manager.

The DEFM smart contracts were developed from scratch following the examples provided by the marketplace component but were subsequently included in the marketplace release as additional examples. The marketplace backend component is deployed on the pilot site as a containerized application while the smart contracts are deployed on the DEFM private

Ethereum network. The marketplace is configured to point to the smart contracts address and manages the communication with the smart contract through specific client libraries.

3.4.2.2 Interledger

The Interledger (IL) component mirrors the main events (i.e. the selection of a winning offer and the final payment) of each market request from the internal private ledger to a public ledger. The usage of a private and permissioned ledger for the operational purposes coupled with a public ledger as a trust anchor provides an optimal trade-off between privacy, security, trust, and execution costs. The IL component runs on the pilot platform as a containerized application. The component was configured to operate in the so-called *left-to-right* mode (one direction). On the *right* side (i.e. the receiving end), it was used as the *receiver* smart contract, included as a sample with the component. The *left* side (i.e. the sender) was developed by implementing the *sender* smart contract interface. The component configuration (smart contracts addresses, the account of the contracts owner, connection details) was provided by editing the dedicated configuration files.

The private/public ledgers, being decentralized networks, are not part of the IL component and are accessed remotely using specific client libraries.

3.4.2.3 Semantic Representation

The Semantic Representation (SR) component validates the data gathered from the smart meters ensuring that the format and the attributes returned are consistent with the expected model. Since the day ahead forecasts are the starting point for each flexibility request on the marketplace, it is important that the data provided by the smart meters are reliable and annotated following the SOFIE representation.

The SR component is not deployed within the pilot platform, but instead it is integrated as an external service called via an API. The SR component can be used as a containerized application. The component is used through its APIs and the integration process consists in the definition of a JSON-schema that follows W3C TD data model structure, as shown in Figure 3.4.c. This data model is then saved in the component internal DB.

```
{
    'title': 'Data',
    'description': 'Energy data',
    'type': 'array',
    'items':
        {
            'type': 'array',
            'items': {'type': 'number'},
            'minItems': 2,
            'maxItems': 2
        },
    'minItems': 1
}
```

Figure 3.4.c: DEFM data model.

3.5 Context-Aware Mobile Gaming Pilot

The focus of the mobile game pilot is to explore how DLTs can be used to provide new gaming features for players, as well as to validate the potential of location-based IoT use cases. It seeks to overcome a known technical issue: the ability of DLTs to scale cost-effectively to support millions of active users per day with hundreds of transactions per second. Multiple use-cases that leverage IoT and blockchain technology are studied and implemented throughout the pilot to test their technical applicability and performance for mobile gaming. The gaming pilot also leverages SOFIE components to provide new gaming features and enhance the player experience.

3.5.1 Pilot architecture

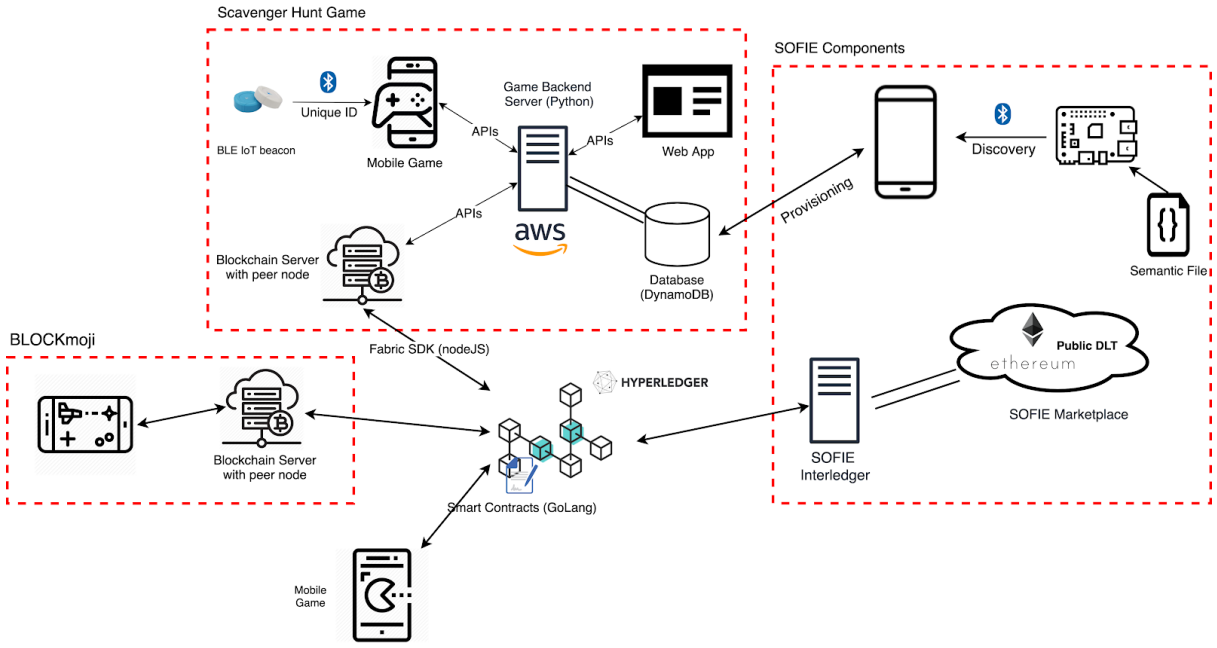

Figure 3.5.a: Mobile Gaming pilot architecture with SOFIE components.

We developed a Scavenger Hunt game prototype to explore location-based IoT gaming. In the game, the player starts a hunt, which takes them on a journey of predetermined real-world locations. At each location, a Bluetooth Low Energy (BLE) beacon is deployed and when the mobile game client detects the beacon, they receive a task in the form of a question. By observing their real-world surroundings, the player can learn the answer to the question, and receive the clue on where the next correct location is. At the end of a hunt (a series of tasks and clues), the player receives rewards that can bring in-game advantages in the next hunts. As additional rewards, the player receives items that are stored on a distributed ledger as non-fungible tokens. To browse and manage these items, a companion application was created: Blockmoji.

In addition to the use cases, we integrated SOFIE Interledger and Marketplace components into our pilot for trading the in-game assets. Furthermore, the SOFIE Provisioning and Discovery along with the Semantic Representation component can be used to discover IoT beacons and add them to the database for a location-based game, such as for our Scavenger Hunt prototype.

3.5.2 Usage of SOFIE components

3.5.2.1 Interledger

The mobile gaming pilot uses the SOFIE Interledger component to communicate between the permissioned Hyperledger Fabric and public Ethereum networks. In-game assets from the Scavenger Hunt use-case are stored on a permissioned Hyperledger Fabric ledger, and the Interledger component is used to transfer those assets to public Ethereum, to be traded on the SOFIE marketplace.

In order to integrate SOFIE interledger, an instance of the component is hosted on an AWS EC2 instance, as our Fabric ledger network was also managed by AWS. After cloning the GitHub repository, we configured the network connection profile (fabric/fixtures/network.json) according to our AWS managed Hyperledger Fabric, that helps the component to connect to the network by providing all the required information. We also updated the smart contracts to emit events that can be listened to by the component. Once the Interledger component is triggered, it passes the data payload containing asset information from the Fabric ledger to the Ethereum ledger. There is no single piece of code that we can provide, We modified the SOFIE interledger component with our network details and linked our chaincode for events.

The component links the closed ecosystem of games and game developers to the public ecosystem of trading games and other virtual assets. The component helps to achieve interoperability between different distributed ledgers and could be used to create new business opportunities as closed platforms can be connected securely and transparently.

The smart-contract can be found in the Mobile Gaming repository of the SOFIE project [MG smartc-contract]

3.5.2.2 Provisioning and Discovery

The mobile gaming pilot uses the SOFIE Provisioning and Discovery component for exploring the use case of discovering, configuring and maintaining the large IoT beacon deployments used in the scavenger hunt game. The component scans nearby IoT devices, learns their capabilities, and determines which devices are suitable for provisioning based on the requirements set by game developers. After provisioning the IoT device, the component configures it to work as BLE beacons, that can be used as PoIs in the scavenger hunt prototype.

We used a Raspberry Pi 4 to act as an IoT device for this component. The SOFIE Provisioning and Discovery component is used along with SOFIE semantic representation, which provides the meta-data for the device. After hosting the semantic file on AWS, we update the URL in the component and run the component. We use an Android application to discover the device and provision it to the AWS DynamoDB, which also stores all IoT beacon related information for the gaming pilot. The newly provisioned device can then also be used in the creation of new hunts in the game.

The component takes advantage of the already deployed Web of Things (WoT) devices. It also creates new business opportunities for device holders, namely micropayments for services used, and the whole process of discovering and provisioning can be trusted and automated through DLTs.

3.5.2.3 Semantic Representation

The mobile gaming pilot uses the SOFIE Semantic Representation component to describe the semantic description of devices using WoT-Things Description. This component is used along with the Provisioning and Discovery component. The semantic file is encoded in a JSON format that also allows JSON-LD processing.

After creating the semantic representation file (Figure 3.5.c) for the IoT device, we run the component instance on AWS EC2 in order to validate it. After validation, we host it on an AWS S3 bucket and use the URL from the Provisioning and Discovery component.

Using the Semantic Representation component, IoT Devices can describe their capabilities that can be later used to provision the devices. The component helps to unify different WoT devices used as beacons and also provides service interoperability.

```
{
    "@context": ["http://www.w3.org/ns/td",
                {"iot": "http://iotschema.org/"}],
    "@type" : "beacon",
    "id": "urn:dev:wot:com:sofie:gp",
    "name": "raspberry pi",
    "description" : "This is a RSP with services",
    "support" : "ROVIO",
    "security": [{"scheme": "basic"}],
    "properties": {
        "status": {
            "writable": false,
            "observable": false,
            "type": "boolean",
            "forms": [{
                "href": "https://sofie.example.com/gp/status",
                "http:methodName": "GET"
            }]
        },
        "eddystone" : {
            "writable": true,
            "observable": false,
            "type": "object",
            "properties": {
                "NAMESPACE": {
                    "type": "string",
                    "const": "784DAFB85D277AAFE5D7"
                },
                "instanceID": {
                    "type":  "number"
                }
            },
            "forms": [{
                "href": "https://sofie.example.com/gp/eddystone",
                "http:methodName": "POST",
                "contentType": "application/json"
            }]
        }
    },
    "actions": {
        "toggle": {
            "safe": false,
            "idempotent": false,
            "forms": [{
                "op": "invokeaction",
                "href": "https://sofie.example.com/gp/toggle",
                "contentType": "application/json"
            }]
        }
    }
}
```

Figure 3.5.c: W3C WoT TD schema.

3.5.2.4 Marketplace

The mobile gaming pilot uses the SOFIE Marketplace component for the trading of the in-game assets. Players can place bids for trading in-game virtual assets on a public Ethereum ledger. The component enables the actual trade of resources in an automated, trusted, and decentralized 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.

We used the Solidity smart contracts provided by the component and deployed them on the Ethereum network hosted on the AWS EC2 instance. We updated the smart contracts with the emitting event and configured them to work with the SOFIE Interledger component. The assets stored on the Fabric ledger are transferred and traded on SOFIE marketplace.

Using the Marketplace component, the in-game assets can have some real-world value also, e.g. trade skin with electricity. DLT-based marketplace grants security, transparency, and traceability, with the effect of increasing a healthy competition among the players participating.

4 Summary

In the SOFIE project WP3 is responsible for integration of the SOFIE Business Platforms and coordinating the SW pipelines per SOFIE pilot. This deliverable presents the methodologies and the work the SOFIE partners have been done to integrate the SOFIE framework components into the business platform. These platforms demonstrate how the SOFIE framework can be used to create systems that solve relevant business problems, and how SOFIE components can be integrated to such business platforms to satisfy specific business requirements. SMAUG pilot demonstrates for a decentralized marketplace use case how all the framework components can work together and seamlessly integrate in one system. The SOFIE deliverable 5.4 'Final Validation Replication Guidelines' [D5.4] provides a deeper view on the business platforms and their goals, together with validation results to confirm that the components are able to enable business application.

Moreover, the business platforms integration demonstrates that the framework component integration is seamless and coherent among all the pilots. The framework components are handled as containers which are configured with pilot's specific information, with the exception of the interledger and marketplace components. The interledger component requires the creation of a smart-contract to work properly with diverse blockchains. The Marketplace has been extended to add pilot specific functionalities, such as support for instant-rent offers and Interledger sender/receiver in the SMAUG pilot and adding the relevant attributes to the energy flexibility use case in the DEFM pilot.

5 References

[D2.6] Y. Kortesniemi et al. “SOFIE Deliverable 2.6 - Federation Architecture, final version”, Nov 2020. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D2.6-Federation_Architecture_final_version-2.pdf

[D2.7] Y. Kortesniemi et al. “SOFIE Deliverable 2.7 - Federation Framework, final version”, Dec 2020. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D2.7-Federation_Framework_final_version.pdf

[D4.5] S. Vasilios et al. “SOFIE Deliverable 4.5 - Final Architecture, System, and Pilots Evaluation Report”, Dec 2020. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D4.5-Final_Architecture_System_and_Pilots_Evaluation_Report.pdf

[D5.4] I. Oikonomidis. “SOFIE Deliverable 5.4 - Final Validation & Replication Guidelines”, Dec 2020. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D5.4-Final_Validation_Replication_Guidelines.pdf

[MG smart-contract] Available at: https://github.com/SOFIE-project/Marketplace/blob/master/solidity/contracts/HouseRenovationMarketPlace.sol

[FSC private] Food Supply Chain repository, smart-contract adapter for consortium blockchain. Available at: https://github.com/SOFIE-project/fsc-consortium-smart-contracts/blob/master/contracts/FoodChainAdapter.sol

[FSC public] Food Supply Chain repository, smart-contract adapter for consortium blockchain. Available at: https://github.com/SOFIE-project/fsc-public-adapter-contract/blob/master/contracts/SofieFSCAdapter.sol

[FCS Adapter] Food Supply Chain repository, federation adapter repository. Available at: https://github.com/SOFIE-project/fsc-transportation-federation-adapter