SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D2.4

SOFIE Federation Architecture 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	1.7.2019
Author(s)	Tommi Elo (AALTO), Pekka Nikander (AALTO), Dmitrij Lagutin (AALTO), Yki Kortesniemi (AALTO), Vasilios Siris (AUEB-RC), Nikos Fotiou (AUEB-RC), Giuseppe Raveduto (ENG), Priit Anton (GT), Margus Haavala (GT), Mikael Jaatinen (LMF), Petri Laari (LMF), Antonio Antonino (LMF), David Mason (ROV), Sotiris Karachontzitis (SYN)
Responsible person	Tommi Elo (AALTO), tommi.elo@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	Authorization Server
CSO	Charging Station Owner
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DSO	Distribution System Operator
EV	Electrical vehicle
FSC	Food Supply Chain
GE	Generic Enabler
HTLC	Hash Time-Lock Contract
IAA	Identity, Authentication, Authorization
IoT	Internet of Things
PoI	Point of Interest
RFID	Radio Frequency IDentification
TD	Things Descriptor
TSO	Transmission System Operator
WoT	Web of Things

1. Introduction

Fragmentation and lack of interoperability among different platforms is a major issue with the Internet of Things (IoT). Currently, IoT platforms and systems are vertically oriented silos unable (or unwilling) to exchange data with, or perform actions across, each other. This leads to multiple problems: reduced competition and vendor lock-ins as it is difficult for customers to switch IoT providers, worse privacy as vendors usually force their customers to move at least some of their data or metadata to the vendor's cloud, and reduced functionality compared to what would be possible with better interoperability. Since IoT systems are becoming prevalent in everyday life, lack of interoperability and limited use of relevant data is growing into a significant problem for individuals, organisations and the society as a whole.

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. The SOFIE architecture is a way of overcoming the lack of interoperability by federating the actions between different IoT systems using interledger technologies. Blockchains and distributed ledgers (DLTs) form a natural basis for building trust between different parties by providing transparency and accountability to operations. Interledger technologies then build on top of the strengths of individual ledger technologies by enabling cross-ledger transactions. Finally, smart contracts allow the automation of many transactions and, thus, lower the operating costs of the system.

A key benefit of the SOFIE architecture is that it allows the creation of solutions that connect many individual systems to a whole that provides significant new functionality. For instance, as depicted in Figure 1, the growth and transportation conditions of agricultural produce is recorded as it moves along the supply chain, providing accurate information to customers while helping companies resolve disputes.

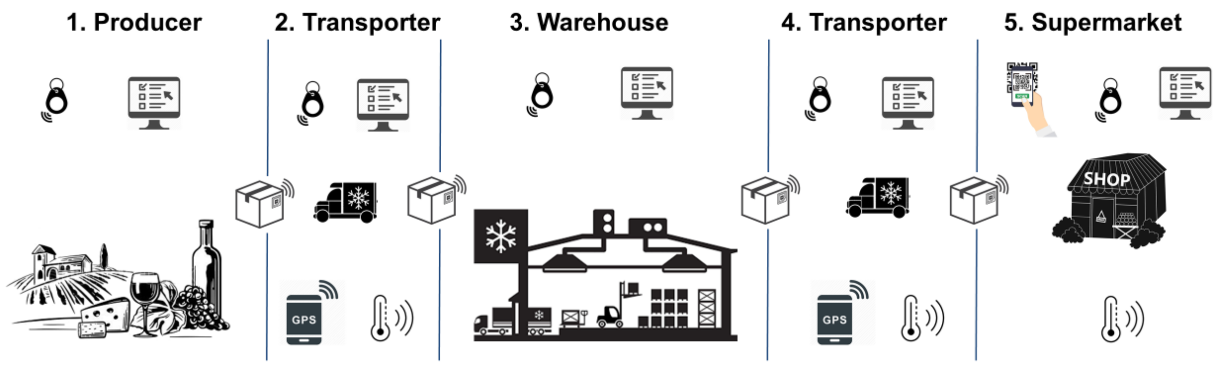


Figure 1. An overview of the SOFIE food supply chain pilot, demonstrating how data is collected as produce moves from the farm to the supermarket through transporters and distributors.

As shown in Table 1, architectures can exist on many levels, such as framework architecture, system architecture, and component architecture. In the scope of SOFIE, this document describes the *SOFIE framework architecture*, which provides a high level overview of the overall SOFIE system, its components, entities, actors, and the interactions between them. The system-level architectures specific to SOFIE pilots are briefly described in Section 3 of this document, with more details provided in SOFIE Deliverable “D5.2 - Initial Platform Validation” [Oik2019]. Finally, the SOFIE components will be described in the SOFIE deliverable “D2.5 - Federation Framework, 2nd version”, due in August 2019. In the latter part of this document the term *architecture* refers to the *framework* architecture unless otherwise specified.

Table 1. Multiple levels of architecture

Architecture Level	Scope	Level of Detail	Relevant SOFIE deliverable
Framework	SOFIE	Broad	This document (D2.4)
System	SOFIE Pilot	Pilot specific	D5.2 (June 2019)
Component	SOFIE Component	Internal structure of component	D2.5 (August 2019)

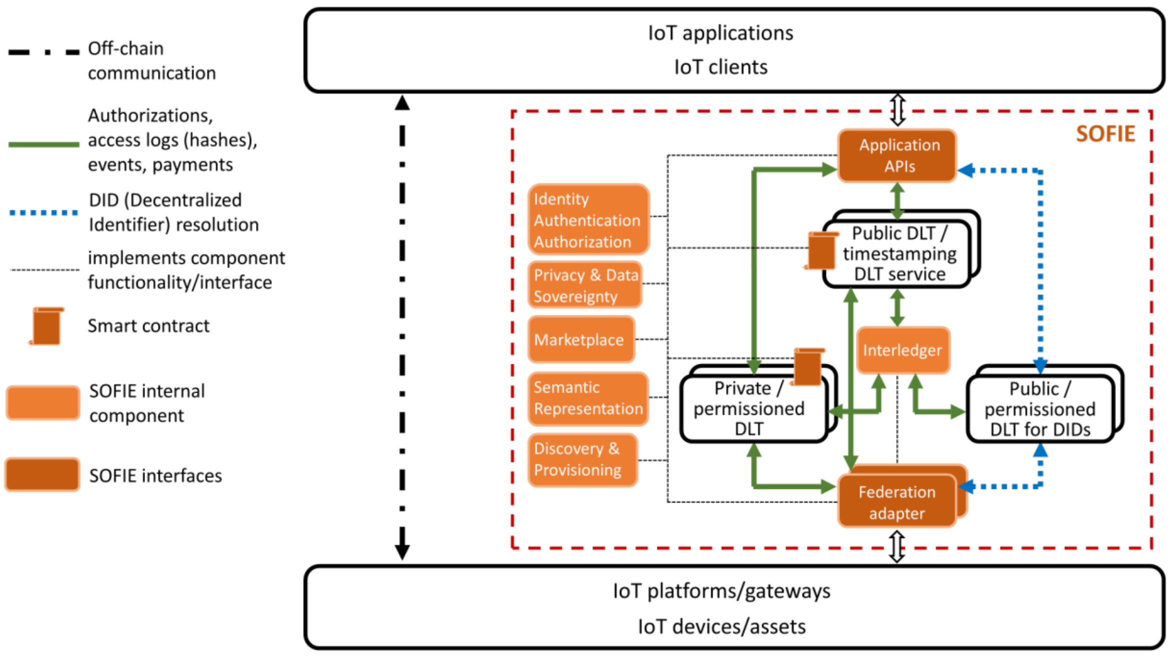
The structure of this document is as follows: Section 2 presents the SOFIE architecture, including entities, roles, and actors, as well as the requirements that pilots and framework components pose on the architecture. Based on the overall SOFIE architecture, Section 3 describes the system architectures of the four SOFIE pilots. The SOFIE framework components and utilized external components are described in Sections 4 and 5 respectively, while Section 6 discusses deployment considerations.

2. SOFIE Architecture

One of the most fundamental assumptions of SOFIE is that it has to be able to support different types of IoT and ledger technologies without requiring changes to those technologies. This is due to the large installed base of existing technologies that do not allow for changes and the fact that different parties and consortiums will continue to select their own IoT and distributed ledger technologies based on the different strengths of those technologies. By allowing the federation of such self-selected ledgers, SOFIE enables interoperability across the technology silos created by the manufacturers, who control these silos.

2.1 Architecture overview

Figure 2 provides a functional overview of the SOFIE architecture. It depicts the main functional components of the SOFIE framework (light orange boxes), SOFIE interfaces (dark orange boxes) and their cross-domain interactions with external domains/components (white boxes).



Off-chain communication
Authorizations, access logs (hashes), events, payments
DID (Decentralized Identifier) resolution
implements component functionality/interface
Smart contract
SOFIE internal component
SOFIE interfaces
IoT applications
IoT clients
SOFIE
Application APIs
Identity Authentication Authorization
Privacy & Data Sovereignty
Marketplace
Semantic Representation
Discovery & Provisioning
Public DLT / timestamping DLT service
Interledger
Private / permissioned DLT
Public / permissioned DLT for DIDs
Federation adapter
IoT platforms/gateways
IoT devices/assets

Figure 2. The SOFIE framework architecture

The lowest level of the architecture contains *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 that are used to transport products. IoT assets can be connected to or integrated in actual devices. *IoT platforms* include platforms with data stores, where the measurements from sensors are collected and made available to third parties, and also servers providing IoT services.

The *federation adapters* are used to interface the IoT platforms with the SOFIE framework and implement the functionality of the federation API. 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 can utilize different types of federation adapters, which implement only the required parts of the SOFIE functionality.

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 decentralized 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/permissionless blockchains are completely open to everyone, which is necessary to achieve wide-scale decentralized trust and transparency but forgoes any privacy. On the other hand, private/permissioned DLTs involve the collaboration of peers that belong to a specific permissioned set and can arrange their records to be opaque 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 authorization, or providing resolution of decentralized identifiers (DIDs) [Ree2019]. Utilizing 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 utilize different distributed ledger technologies.

The other SOFIE framework components are: *Identity, Authentication, and Authorization (IAA)*, which provides identity management and supports multiple authentication and authorization techniques; *Privacy and data sovereignty*, which provides mechanisms that enable data sharing in a controlled and privacy preserving way; *Semantic representation*, which provides tools for describing services, devices, and data in an interoperable way; *Marketplace*, which allows participants to trade resources by placing bids and offers in a secure, auditable, and decentralized way; and *Discovery & provisioning*, which provides functionality for the discovery and bootstrapping of services.

The architecture 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 are the *application APIs*, which provide the interfaces for IoT clients and applications to interact with the SOFIE framework.

2.2 Entities and Roles

In the scope of SOFIE, an **entity** can be either physical or non-physical, with a distinct and independent existence. Every entity has at least one unique identity, which separates it from all other entities, and each identity utilises one or more identifiers. Identities can express various roles of the entity, e.g. a person can have identities related to work and private life. For example, a personal mobile phone could be used as a key to both the office photocopying machine and as a personal car key. In order to protect privacy from correlation attacks there can be a large number of identifiers associated with a single identity. An important feature of identifiers is self-sovereignty, i.e., the entity can generate, manage and discard identifiers by itself, without permission from any third-party.

A *SOFIE-based system* is a system that follows the SOFIE architecture. Table 2 presents the various entities of SOFIE-based systems, and Table 3 presents the entities, platforms, devices and resources participating in the SOFIE pilots.

Table 2: Different types of entities in a SOFIE-based system.

Entity	Description
Person	Individual (human) that can act e.g. as a customer or organization employee.
Organization	Entity which can own or operate platforms and resources, and which consists of employees (persons).
Service	Mechanism to enable access to one or more capabilities[1].
Agent	An automated or semi-automated software component acting on behalf of a person or organization within the constraints defined by the originating entity.
Device	IoT device, such as a sensor or actuator.
IoT platform	Hardware and software entity providing an IoT service.
IoT gateway	Entity that interconnects one or more devices to a wide area network.
Resources	Physical or digital assets.
Network	Used for communicating data and control. Examples include Internet, private networks, device-to-device (D2D)
Distributed ledger	Transaction bookkeeping mechanism implemented in a decentralised manned. There are different types, e.g., public/permissionless, permissioned (or consortium).
Smart contract	Program executed on the virtual machine of a distributed ledger (e.g., Ethereum's EVM). A smart contract can implement a subset of the functionality of a framework component.
Interledger	Entity implementing interledger (operations spanning 2 or more ledgers) functionality. Can also be viewed as a specialized agent acting on behalf of a smart contract.
Oracle	Entity allowing smart contracts to interact with the Internet (e.g. call APIs). Can also be viewed as a specialized form of an agent acting on behalf of a smart contract.

[1] A **service** is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description [Mac2006].

Table 3: Entities, platforms, devices and Resources in the SOFIE pilots.

Pilot	**Organizational entities (OE)**	**Other entities**	**Platforms**	**Devices**	**Resources (assets)**
Food Supply Chain	Producer Transporter Warehouse Supermarket	Employees (of OEs) Consumer	Farm (SynField) IoT Transportation IoT Warehouse (Aberon) IoT	SynField nodes Sensors RFID readers Smartphones	Boxes
Decentralised Energy Flexibility Marketplace	DSO CSO Fleet Manager (Electricity provider)	Charging station EV user	Electricity management system Charging station platform	Sensors Smart meters IoT device in EVs	EVs Energy Smart meter measurements Incentive tokens
Decentralised Energy Data Exchange	TSO DSO Smart meter system operator Energy producer Energy consumer	Smart meter owner	Smart meter platform	Smart meters	Energy Smart meter measurements
Mixed Reality Mobile Gaming	Game company Ad company PoI company	Game developer Challenge designer Ad manager PoI employee Player	Game server	Sensors Smartphones Beacons	Gaming rewards In-game assets

An actor is a participant or its delegate that interacts with the SOFIE system. The service roles in the SOFIE system are shown in Table 4. In some scenarios, the same actor can assume different roles or the same role for different purposes. For example, a farm owner can be both an owner and producer of agricultural produce. Moreover, an actor can have one or more roles concurrently and may change them over time and in different contexts, even over the course of a particular interaction [Mac2006].

Table 4: Service roles in the SOFIE-based system

Role	Description
Consumer	Role performed by a participant who interacts with a service in order to fulfil a need.
Provider	Role performed by a participant who offers a service.
Intermediary	Role that facilitates the interaction and connectivity to provide a service.
Owner	Role performed by a participant that claims or exercises ownership over an entity or service.
Delegate	Role played by a person or by an automated or semi-automated agent on behalf of a participant. The delegate must have the participant's authority.

Table 5 describes actors of SOFIE pilots, while Section 3 contains more details about pilots.

Table 5: Actors in the SOFIE pilots.

Pilot	**Actors**
Food Supply Chain	*Producer:* Provider that produces agricultural produce, e.g., a farm. *Transportation company employee (transporter):* Provider that is responsible for transporting produce, e.g. between farm and warehouse, or warehouse and supermarket. *Warehouse employee:* Individual employed by the warehouse. *Supermarket employee:* Individual employed by the supermarket. *Supermarket customer:* Consumer buying produce from the supermarket. *Consortium certifier organization:* Provider that administers the common ledger system that stores data related to asset tracking. It acts as an authority that grants and enables access to participants' data. It is also the actor who activates and supervises the process of audit and dispute resolution in the case of a breach or when a customer reports an issue about product quality.
Decentralised Energy Flexibility Marketplace	*Energy provider (DSO)*: Provider who manages the electrical grid and provides electricity transfer services. *Fleet manager*: Provider that manages a fleet of EVs and offers related services (e.g., rental) to its customers *EV user:* Consumer using an EV.
Decentralised Energy Data Exchange	*Energy service provider*: Provider responsible for providing the energy service to the end-user (customer). Can have the role of a DSO or an energy company. Main communicator between customer and grid operator (i.e., the TSO). *Smart meter system operator*: Provider responsible for the national or regional smart meter network. Acts as a delegate of smart meter owner.

	Smart meter owner: Individual or company that is legally bound to a smart metering point and is interested in consuming/producing energy.
Mixed Reality Mobile Gaming	*Game player:* Consumer who can join any challenge, view their profile and reward data through a mobile application. *Game developer (administrator):* has complete access to the game and its data. It can view and edit all the challenges, player profiles, and related information. *Challenge designer:* Provider who can create new challenges, assets, tasks and puzzles for the existing beacons *Ads manager (administrator):* Provider, who can monitor and approve the advertisements shown in the application. *PoI employee:* can view data about the PoI challenges, offer rewards, or create new PoI challenges.

2.3 Actor interaction model

The actor interaction model in Figure 3 presents the interactions between the various actors identified in the previous subsection and the entities performing the actor roles. The interfacing of the providers (IoT platforms) with the network and the DLTs is implemented using federation adapters as discussed in Section 2.1, and these interfaces can span ownership and administration boundaries. Similarly, the security, management, and governance actions can also span ownership and administration boundaries. The interfaces and actions across boundaries are scenario and pilot specific; more details are presented in the specific pilot descriptions in Section 3 of this deliverable. Finally, consumers (IoT applications and clients) access services through application APIs, as depicted in the SOFIE federation architecture in Figure 2.

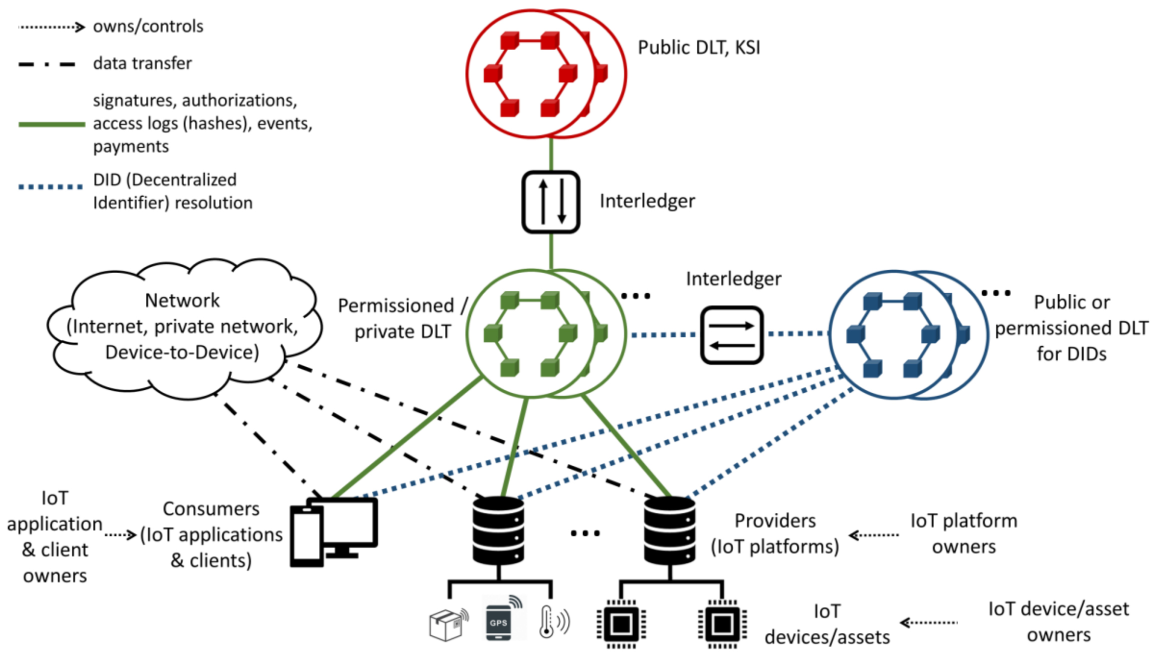

owns/controls
data transfer
signatures, authorizations, access logs (hashes), events, payments
DID (Decentralized Identifier) resolution
Public DLT, KSI
Interledger
Network (Internet, private network, Device-to-Device)
Permissioned / private DLT
Interledger
Public or permissioned DLT for DIDs
IoT application & client owners
Consumers (IoT applications & clients)
Providers (IoT platforms)
IoT platform owners
IoT devices/assets
IoT device/asset owners

Figure 3. Actor interaction model

The interaction of the various actors takes place with messages, through which the exchange of information and value pertaining to the specific service will be performed. The behavior and actions across the interfaces will be defined by the corresponding protocols that will be discussed in the SOFIE deliverable "D2.5 - Federation Framework, 2nd version", due in August 2019.

2.4 Requirements for SOFIE

This section describes the requirements for the SOFIE architecture and its components, many of which have been derived from the needs of the SOFIE pilots. For instance, in the SOFIE pilots there is a need for accountability and auditability between multiple parties, who do not always fully trust each other, which can be achieved using DLTs: storing a hash of a transaction tree to the public ledger would provide a trust anchor and further increase the security, transparency, and auditability of the system as parties cannot modify existing transaction logs after the hash has been publicly revealed. However, due to privacy requirements and the need to maintain business secrets, it is not feasible to store all the data to a single DLT - instead, multiple DLTs should be used. Furthermore, various DLTs have tradeoffs in terms of throughput, latency, cost and scalability, therefore interledger operations between the DLTs must be supported by SOFIE. Finally, several DID operations rely on a related DLT, therefore in order to connect identifier creation, authorization, and authentication functionality to the rest of the SOFIE framework, interaction between the DLT ledger and other ledgers used by SOFIE is necessary.

Tables 6-8 list detailed requirements for the SOFIE architecture, the SOFIE framework components, and privacy-related requirements for the implementation and deployment of the SOFIE architecture. Each requirement is associated with a unique reference ID, a short description, a priority level, a category and the SOFIE architecture/framework component which is responsible for addressing it. Six categories are used (QUALITY, AUDITABILITY, INTEROPERABILITY, USABILITY, SECURITY, POLICY & REGULATION). Three priority levels are considered according to the following rule:

Must – The requirement is a "must have"
Should – The requirement is needed for improved operation, and the fulfilment of the requirement will create immediate benefits

Table 6. Requirements for the SOFIE architecture

Req. ID	Requirement Description	Priority	Category
RA01	SOFIE architecture must define a clear separation between data management, control, and representation processes.	MUST	QUALITY
RA02	SOFIE architecture must be modular to enable different use cases and reuse of components.	MUST	QUALITY
RA03	The interfaces of the SOFIE components must be well-defined and fully documented.	MUST	QUALITY
RA04	Transactions must be immutable and verifiable. Parties must not be able to modify existing transactions without other parties noticing it. Every party should be able to independently verify the validity of transactions.	MUST	SECURITY
RA05	The system must provide auditability.	MUST	SECURITY
RA06	Support for transactions, where only authorised entities can participate. Minimal amount of information should be disclosed during authentication.	MUST	SECURITY

RA07	All external and internal interfaces and communication links of the system must conform to the principle of least privilege[2].	MUST	SECURITY
RA08	The SOFIE architecture should be flexible and support different means of user authentication, including password-based, certification-based, and token-based.	SHOULD	SECURITY

The SOFIE architecture satisfies architectural requirements (RA01 - RA08) in the following way: Section 2.1 of this document explains how the SOFIE architecture separates data management, control and representation, hence fulfilling the first requirement (RA01). The SOFIE architecture is divided into six framework components and therefore satisfies requirement RA02. The interfaces of framework components will be defined and documented in upcoming SOFIE deliverable D2.5 (requirement RA03). Requirements RA04 and RA05 are satisfied through usage of DLTs in SOFIE architecture, while requirements RA06 and RA08 are satisfied by the SOFIE IAA component. The Privacy & data sovereignty component fulfills the requirement RA07 (principle of least privilege) by providing different APIs and interfaces for different uses and requiring access control policies for all APIs and interfaces.

Table 7. Requirements for SOFIE framework components

Req. ID	Requirement Description	Priority	Category
	Interledger		
RF01	User interaction is not required for interledger operations.	MUST	USABILITY
RF02	There should be support for atomic interledger operations.	SHOULD	SECURITY
	IAA		
RF03	Resource owners must be able to delegate the authentication and authorization tasks for their resources.	MUST	OPERATIONAL
RF04	The IAA component must provide users the capability to revoke authorizations.	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	AUDITABILITY

[2] https://en.wikipedia.org/wiki/Principle_of_least_privilege

RF09	SOFIE should support the execution of authorization and authentication functionality on devices with constrained processing, storage, battery, and network connectivity.	SHOULD	OPERATIONAL
	Privacy & Data Sovereignty		
RF10	SOFIE must follow the data minimization principle for personal data and only request or process what is necessary for the situation and purpose.	MUST	OPERATIONAL
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 organizations 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
	Semantic representation		
RF15	SOFIE must define an IoT things description model based on well-known standards (e.g. W3C standards).	MUST	AUDITABILITY
RF16	SOFIE must implement standardized metadata and data representation formats and support various data modalities.	MUST	AUDITABILITY
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	AUDITABILITY
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	INTEROPERA-BILITY
RF19	SOFIE should support the semantic update and enhancement of resources' descriptions and associations in a dynamic way.	SHOULD	INTEROPERA-BILITY
	Marketplace		
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	INTEROPERA-BILITY
RF22	The marketplace must support transparent trading of resources, i.e. the bids/offers matching process and the payments must be transparent.	MUST	OPERATIONAL
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	OPERATIONAL

These requirements and how the SOFIE components address them will be discussed in more detail in D2.5.

Table 8. Privacy requirements related to implementation and deployment of SOFIE architecture

Req. ID	Requirement Description	Priority	Category
RP01	Privacy issues and business secrets must be considered carefully when deciding what data (including authentication/authorization information, logs etc.) is collected, stored or exchanged between parties.	MUST	POLICY & REGULATION

The SOFIE Privacy & data sovereignty component also enables fulfilment of requirement RP01 by providing support for e.g. access control and differential privacy schemes. The SOFIE Interledger component will also be used to limit access to data: a subset of data will be stored in another ledger, and access to the ledger containing more data will be more restricted.

3. Pilot Architectures

This section details how the four pilots in the SOFIE project utilise the SOFIE framework architecture to implement their respective system architecture. Figure 4 presents an overview of how the SOFIE framework relates to the various pilots and the existing components used by them. At the bottom of the figure are IoT devices, while other components used by the pilots are in the upper part of the figure. Dashed line denotes a connection between existing and external components or devices.

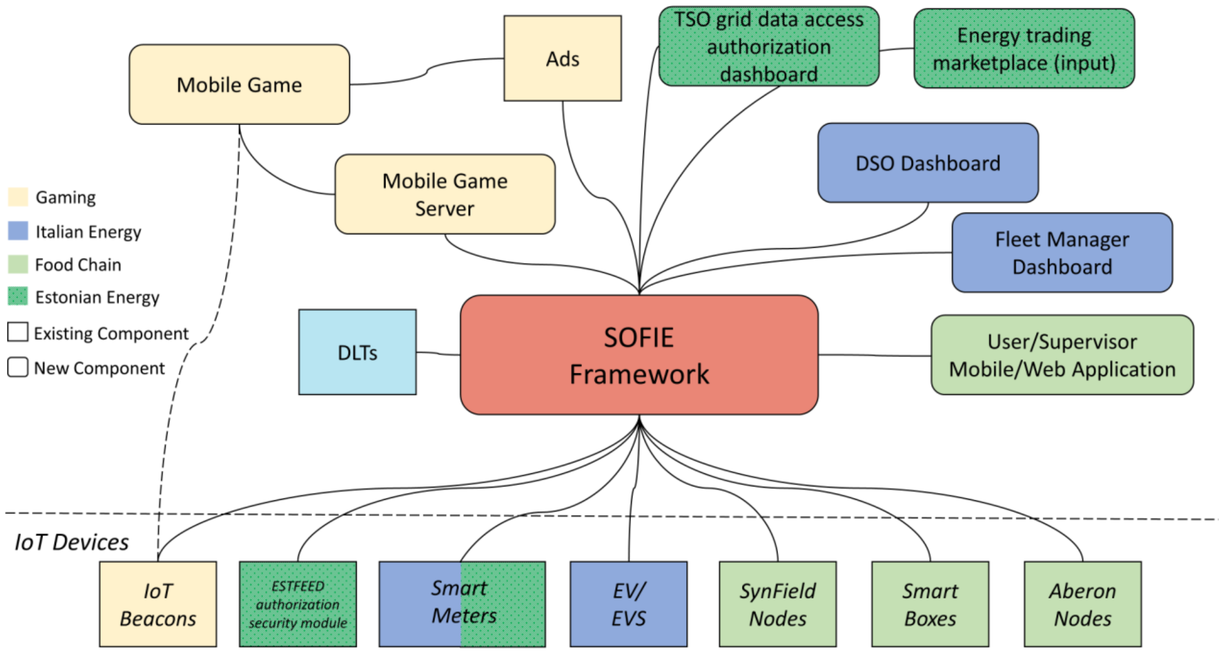


Figure 4. Overview how SOFIE framework relates to SOFE pilots and their components

The rest of this section details the architectures of each of the four SOFIE pilots. More detailed description about pilots themselves can be found in SOFIE Deliverable "D5.2 - Initial Platform Validation" [Oik2019].

3.1 Food Supply Chain Pilot Architecture

An overview of the food supply chain pilot architecture is shown in Figure 5. This architecture aims to validate the SOFIE federation framework by offering two main IoT applications, namely the usage of QR codes to encode product history from the field to the market shelf, and product quality audits and resolution of disputes in the case of product quality degradation events. Both these services, as well as other simple services, are provided to the actors through a Food Supply Chain (FSC) web application.

At the lowest level, three IoT platforms are federated, namely the SynField IoT platform that collects measurements about growing conditions in the field, a Transportation IoT platform that collects measurements about products as they are transferred from one site to another, and the Aberon IoT platform that is responsible for collecting measurements related to the storage conditions of products in the warehouse. A Federation adapter is applied on top of the northbound API of each IoT environment to adapt the corresponding data and metadata according to the SOFIE semantic representation and, also, to support authentication and interledger procedures. As shown in Figure 5, the architecture makes use of three different ledger deployments to guarantee secure data storage and integrity, i.e. a private consortium ledger where all (meta)data used to enable the pilot IoT applications are stored, the KSI blockchain which is used to periodically create signatures of these data, and a public ledger which is used to store these signatures. In addition to the SOFIE framework functional components, the pilot architecture also introduces an additional component, called the Supervisor Web Server, with the following main responsibilities: i) orchestrate the data flow defined by the architecture and take over data and metadata management, ii) expose a public API to address the requests which are received by the actors through the SOFIE FSC web application, and iii) supervise the status of each asset of the provenance business platform (in pilot terminology, the boxes that carry products over the whole food chain are considered assets) and schedule the proper execution of the services which are provided by the SOFIE components based on actors' activity.

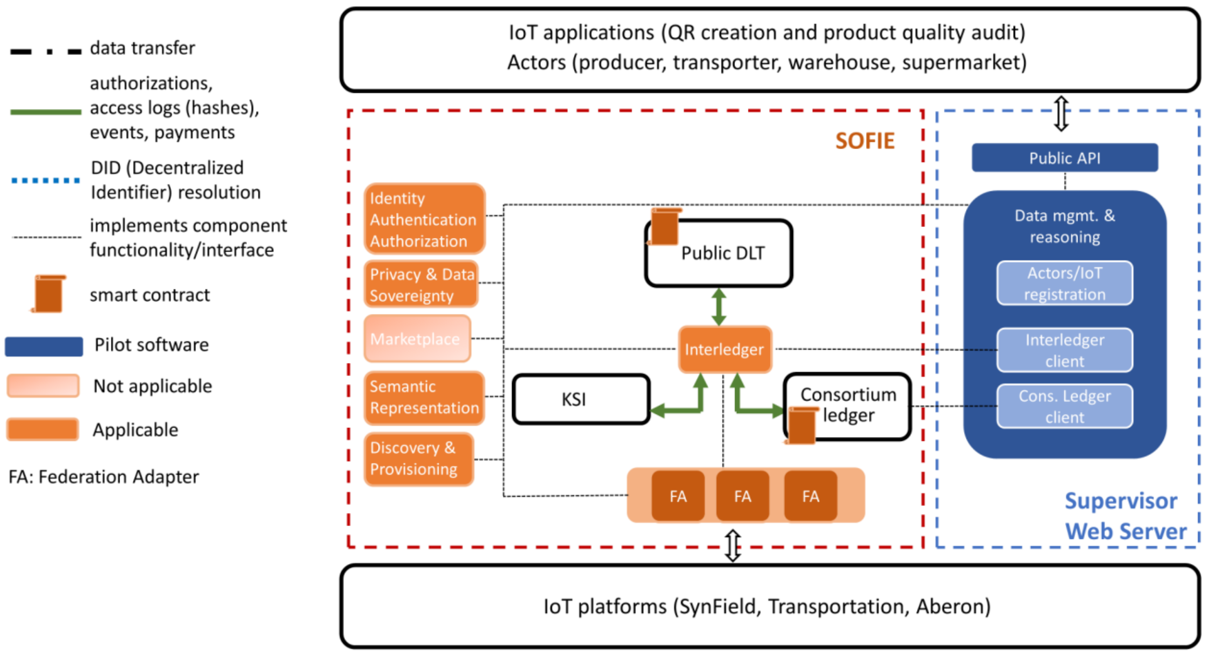

data transfer
authorizations, access logs (hashes), events, payments
DID (Decentralized Identifier) resolution
implements component functionality/interface
smart contract
Pilot software
Not applicable
Applicable
FA: Federation Adapter
IoT applications (QR creation and product quality audit)
Actors (producer, transporter, warehouse, supermarket)
SOFIE
Identity Authentication Authorization
Privacy & Data Sovereignty
Marketplace
Semantic Representation
Discovery & Provisioning
Public DLT
Interledger
KSI
Consortium ledger
FA
FA
FA
Public API
Data mgmt. & reasoning
Actors/IoT registration
Interledger client
Cons. Ledger client
Supervisor Web Server
IoT platforms (SynField, Transportation, Aberon)

Figure 5. Food supply chain pilot architecture

Table 9 briefly summarizes the use cases in the pilot and Figure 6 shows how the actors of the food supply chain interact with the system. More details of the pilot are provided in [Oik2019].

Table 9: Use cases of the Food Chain pilot

ID	Use case	Description
1	Register crop	The producer provides information about farm location, crop establishment date and product variety which will be transferred to the warehouse or supermarket.
2	Box product	The producer specifies which boxes (of those that have been received from the transportation company) will be used to carry product to the warehouse. He also provides information about the cultivation process, e.g. used fertilizers, dates of audits from public authorities, harvesting date, etc.
3	Handover PR-TR	The producer and the transporter agree to transfer responsibility of one or more boxes carrying (raw) product. Parameters such as weight of boxes, ripening level of product etc. are also confirmed between the two parties.
4	Handover TR-WH	The transporter and the warehouse employees agree to transfer responsibility of one or more boxes carrying either raw or packetised product.
5	Register session	The transporter establishes a session that refers to the product transfer from the field to the fork. He specifies one or more boxes (by using their RFID tags)

		to be used for carrying the specific product from the specific producer. The boxes are delivered to the producer.
6	Pick truck	The transporter specifies the truck that will be used to carry boxes from an origin site (field or warehouse) to a destination site (warehouse or supermarket).
7	Transfer box(es)	Boxes carrying product are transferred from one site to another by the transporter.
8	Handover TR-SM	The transporter and the supermarket (employee) agree on the delivery and transfer of responsibility of one or more boxes carrying packetised product.
9	Store box(es)	The warehouse employee specifies the storage rooms where each box is placed based on the quality and safety specifications of the contained product (e.g. ripening level, temperature etc).
10	Packetise product	The warehouse employee deposits the raw product into the food packaging automation system where packages are made. The packets are placed inside one or more boxes specified by the employee.
11	Create QR code	The supermarket employee creates a QR code for a specific box that records the history of the carried product from the field to the supermarket. A QR label is attached to each packet of the box.
12	Release box(es)	The warehouse, the supermarket, or the transporter employee releases one or more boxes which are not used any more (e.g. after removing all the contained product).
13	Read QR code	The customer uses his smartphone to scan a QR code (which is attached on the surface of a package) and gets the full history of the contained product.
14	Product audit	The supermarket employee reports a quality issue related to one or more boxes used to carry products. The consortium certifier organisation initiates the audit process to track the full history of the box(es) and identify the business segment which is responsible for that issue. The certifier finally informs the supermarket about the audit results.

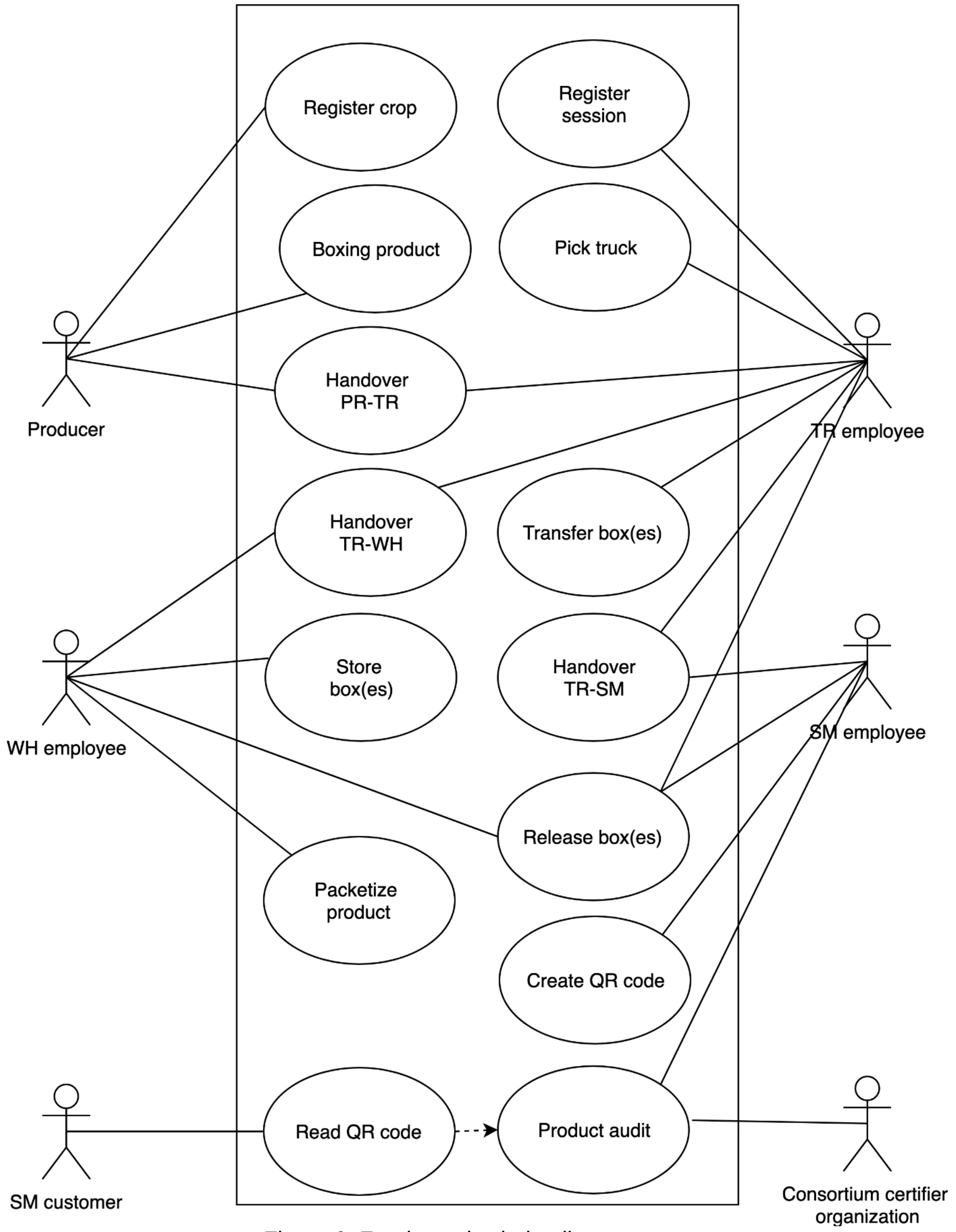


Figure 6. Food supply chain pilot use cases

3.2 Decentralised Energy Flexibility Marketplace Pilot Architecture

The architecture presented in Figure 7 enables the two main actors to participate to the SOFIE decentralized marketplace for the Electric Vehicle pilot. At the lowest level, the smart meters (SMX), the electrical vehicles (EV), and the electric vehicle supply equipment (EVSE) IoT systems are federated through the SOFIE Federation Adapters, providing data and services representation according to the SOFIE semantic representation and supporting authentication and authorization.

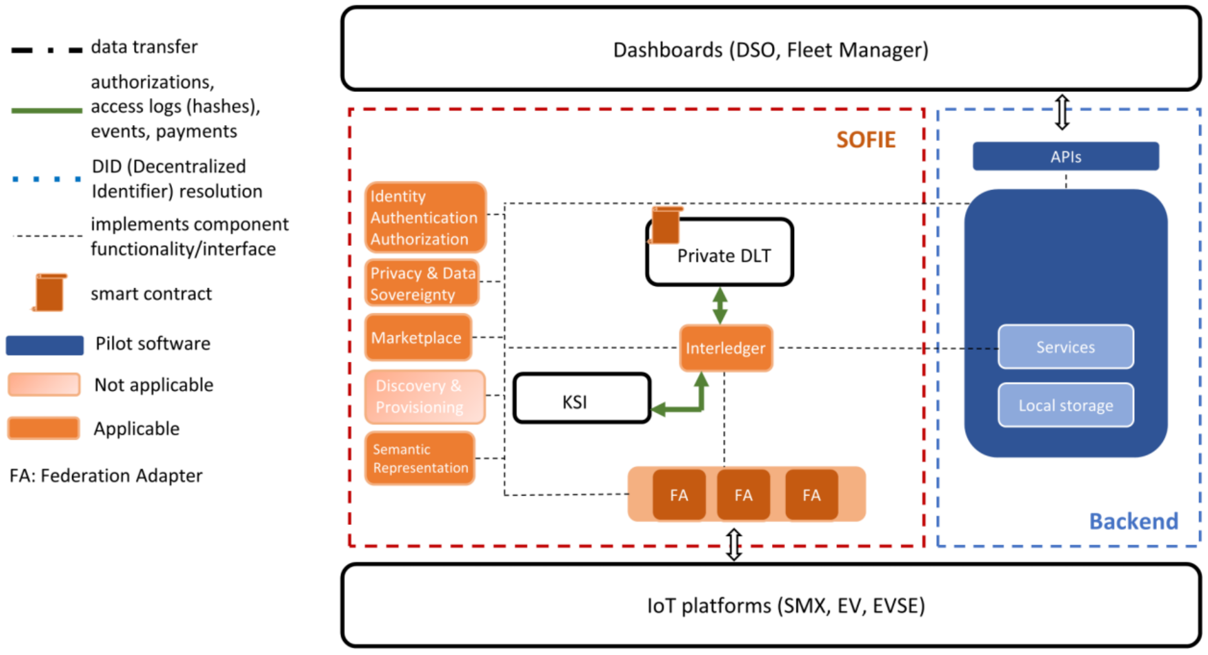

data transfer
authorizations, access logs (hashes), events, payments
DID (Decentralized Identifier) resolution
implements component functionality/interface
smart contract
Pilot software
Not applicable
Applicable
FA: Federation Adapter
Dashboards (DSO, Fleet Manager)
SOFIE
Identity Authentication Authorization
Privacy & Data Sovereignty
Marketplace
Discovery & Provisioning
Semantic Representation
Private DLT
Interledger
KSI
FA
FA
FA
APIs
Services
Local storage
Backend
IoT platforms (SMX, EV, EVSE)

Figure 7. Decentralised energy flexibility marketplace pilot architecture

A private ledger is used to run the smart contract governing the decentralized marketplace, while the KSI blockchain is used periodically to create signatures of the private ledger status. The SOFIE Interledger component manages the two different ledgers operating and securing the pilot. The pilot architecture utilises also backend components in charge of exposing APIs addressing the requests received by the actors via the dedicated dashboards and orchestrating the communication with the SOFIE components.

Table 10 summarises the use cases of this pilot and Figure 8 shows how the actors interact with the system. [Oik2019] provides a more detailed description for each use case.

Table 10: Use cases of the Electric Vehicle pilot

ID	Use Case	Description
1	Flexibility Request	When the DSO foresees a potential reverse flow, the IoT system creates a new request in the flexibility marketplace
2	EV Offers Request (Pull)	When the fleet owner performs day ahead itinerary and charging plans for its EV fleet, he will accept the flexibility requests available in the flexibility marketplace if the requests are compatible with his needs
3	EV Offers Request (Push)	When the user receives a discounted price notification, he will accept the flexibility request available in the flexibility marketplace if the request is compatible with his transport needs.
4	EV/EVSE Fleet Monitoring	To perform both Energy Pilot Scenarios, Fleet Manager have to constantly monitor the EV/EVSE fleet

5	EVSE Fleet Management	To perform both Energy Pilot Scenarios, Fleet Manager must be able to remotely control the EVSE, thus having the ability to remotely start or stop a charging session or change the power output.
6	EV Load Forecasting	To perform the Pull Offers Scenario, the Fleet Manager has to constantly calculate EV load forecasting to estimate the amount of energy that electric vehicles can consume to meet the DSO's flexibility demand.
7	District Forecasting	To perform Energy Scenarios, the DSO has to constantly calculate building consumption forecasting, PV production forecasting and manage batteries to estimate the amount of energy demand at ASM substation
8	Electricity Supply Request	When the Fleet Manager accepts the flexibility requests available in the flexibility marketplace, he will request an electricity supply to energy retailers.
9	Electricity Supply Offer	The energy retailer that offers the electricity supply at the lowest price signs a micro contract with the Fleet Manager.

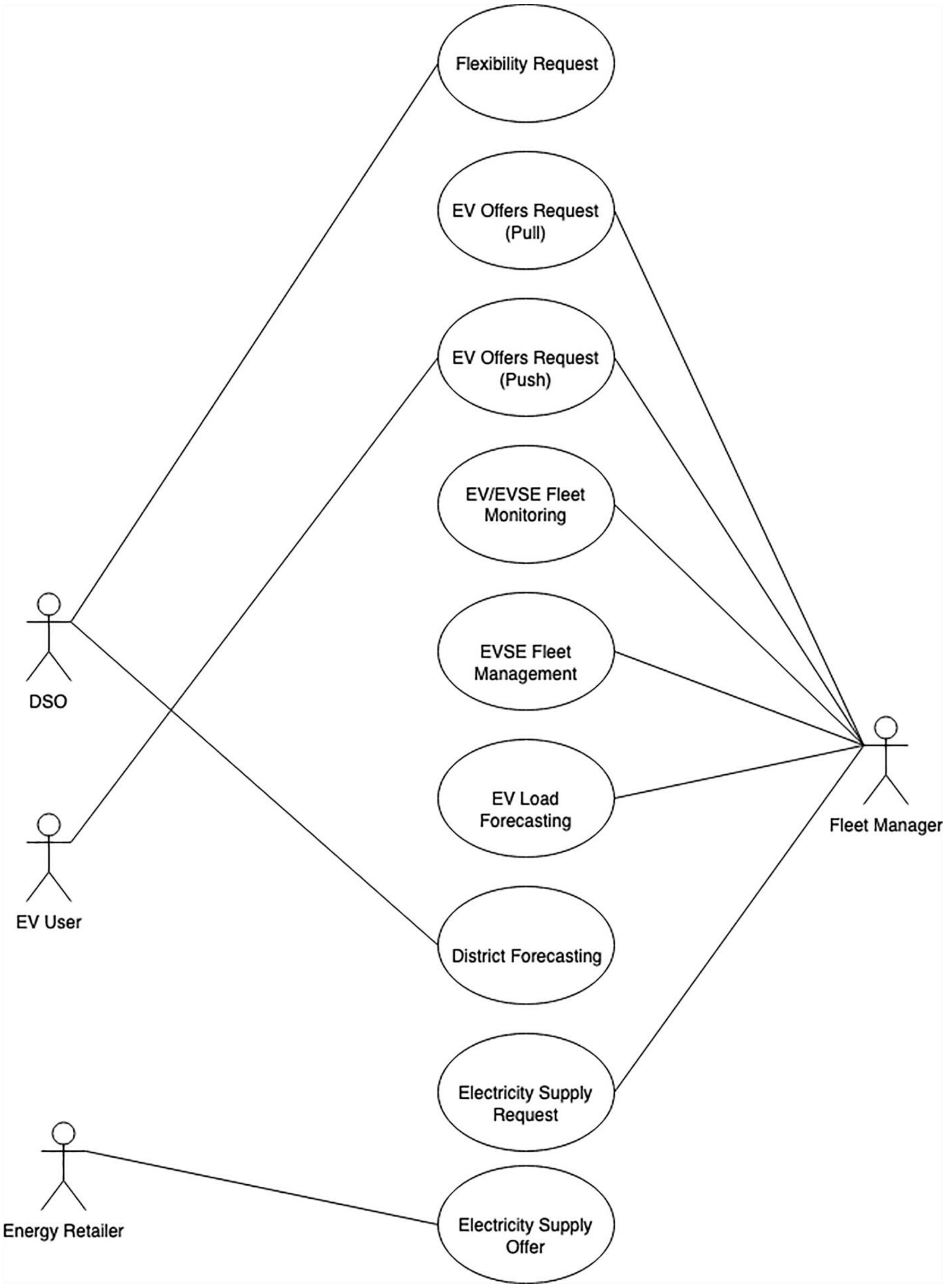


Figure 8. Decentralised energy flexibility marketplace pilot use cases

3.3 Mixed Reality Mobile Gaming Pilot Architecture

The gaming pilot leverages SOFIE to provide new gaming features for players. In the first use-case, the gaming pilot uses a DLT platform to provide players with direct ownership of their assets as well as transparency and consistency of asset attributes and transactions. The SOFIE marketplace is then used for trading gaming assets and for providing security and traceability. In the second use-case, the pilot uses the SOFIE framework to establish a hybrid data organization, where some data is stored locally and some is shared, and the SOFIE identity and authentication component secures access to the data. Finally, the SOFIE interledger module is used for end-to-end security for data transactions.

An overview of the game pilot architecture is shown in Figure 9.

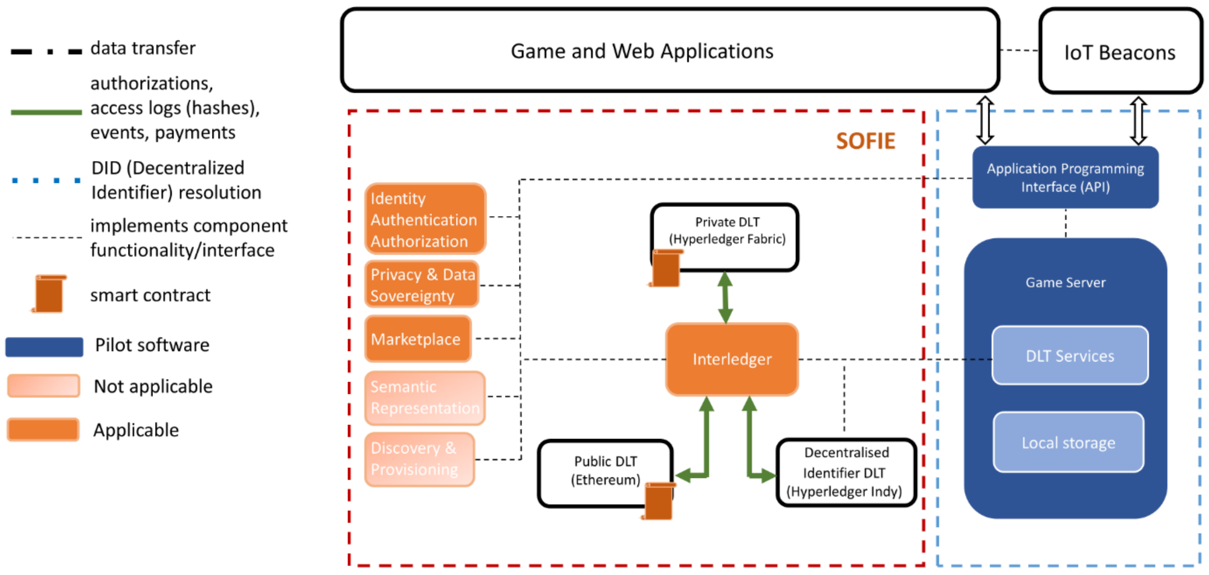

data transfer
authorizations, access logs (hashes), events, payments
DID (Decentralized Identifier) resolution
implements component functionality/interface
smart contract
Pilot software
Not applicable
Applicable
Game and Web Applications
IoT Beacons
SOFIE
Identity Authentication Authorization
Privacy & Data Sovereignty
Marketplace
Semantic Representation
Discovery & Provisioning
Private DLT (Hyperledger Fabric)
Interledger
Public DLT (Ethereum)
Decentralised Identifier DLT (Hyperledger Indy)
Application Programming Interface (API)
Game Server
DLT Services
Local storage

Figure 9. Mixed reality mobile gaming pilot architecture and their components

The gaming pilot will consist of several components:

- **Mobile Application:** A mobile application with a graphical user interface running on the Android platform. Players will install the application to play the challenges, to redeem rewards and to trade assets on the SOFIE marketplace. This application communicates with the game server using REST APIs.
- **Web Application**: A web interface for services related to the game. It is only accessible by the game company and Point of Interests (PoI). It can be used to configure game related services, access the Google cloud to get beacon-related information and also provide a GUI to do transactions with blockchain. A PoI can use the web application to create custom challenges and also provide rewards.
- **Game Server:** A server that provides services to the game and also acts as middleware for communicating with the SOFIE platform. It can be accessed through the REST APIs. It will also be connected to a private database to store the information related to the game and players.
- **Hyperledger Fabric:** A permissioned blockchain to store data from the game. Smart contracts will be coded and used to generate transactions that will be recorded on ledger.
- **IoT beacons server:** A server that provides services to design a new challenge and also acts as middleware to communicate with the SOFIE platform. It is responsible for handling IoT beacon services such as providing beacons status or pushing clues / tasks to the games. It can be accessed through the APIs.

Table 11 summarises the use cases of the pilot and Figure 10 shows how the actors interact with the system. [Oik2019] provides a more detailed description for each use case.

Table 11: Use cases of the mixed reality mobile gaming pilot.

ID	Use case	Description
1	Play challenges / tasks	The player can join any challenge, receive the clues and compete for the reward.
2	Redeem rewards	After completing the challenge, points are calculated for each player and the winner receives the reward (Coupons, Tokens, e.t.c.)
3	View In-App Advertisements	During the challenges, players will be given the option to view advertisements.
4	Asset trading	Players can trade coupons and tokens on the marketplace.
5	Design new challenges	New challenges are created using the installed beacons. Custom clues can be added for each beacon.
6	Access management	New accounts for developers, PoI employees, and ad managers need to be approved by the game company.
7	Offer rewards	The offered rewards should be added to the blockchain using the smart contracts before publishing the new challenge.
8	Publish new advertisements	New ads for In-App advertisement can be published using the smart contract.

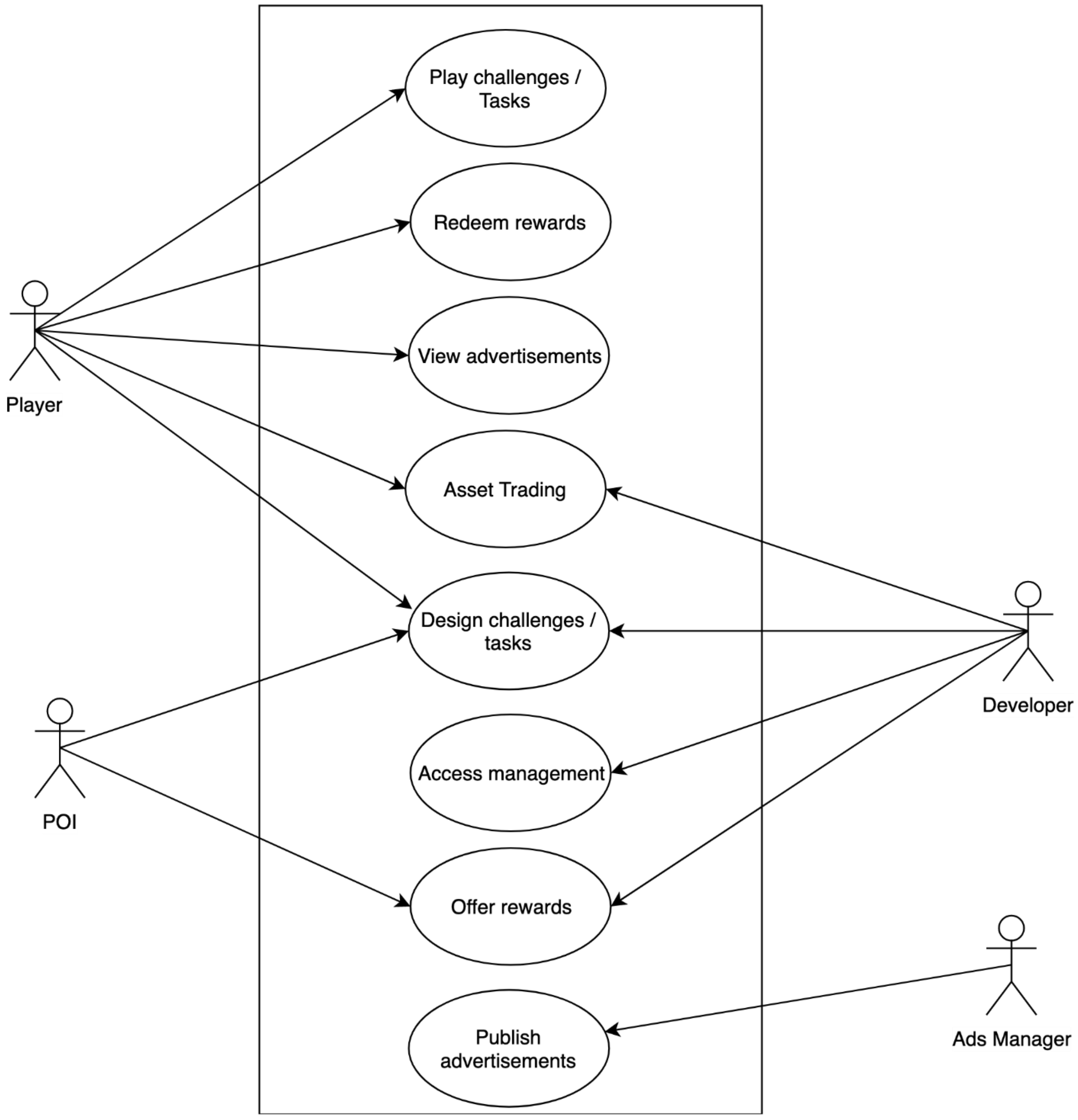


Figure 10. Mixed reality mobile gaming pilot use-cases

3.4 Decentralised Energy Data Exchange Pilot Architecture

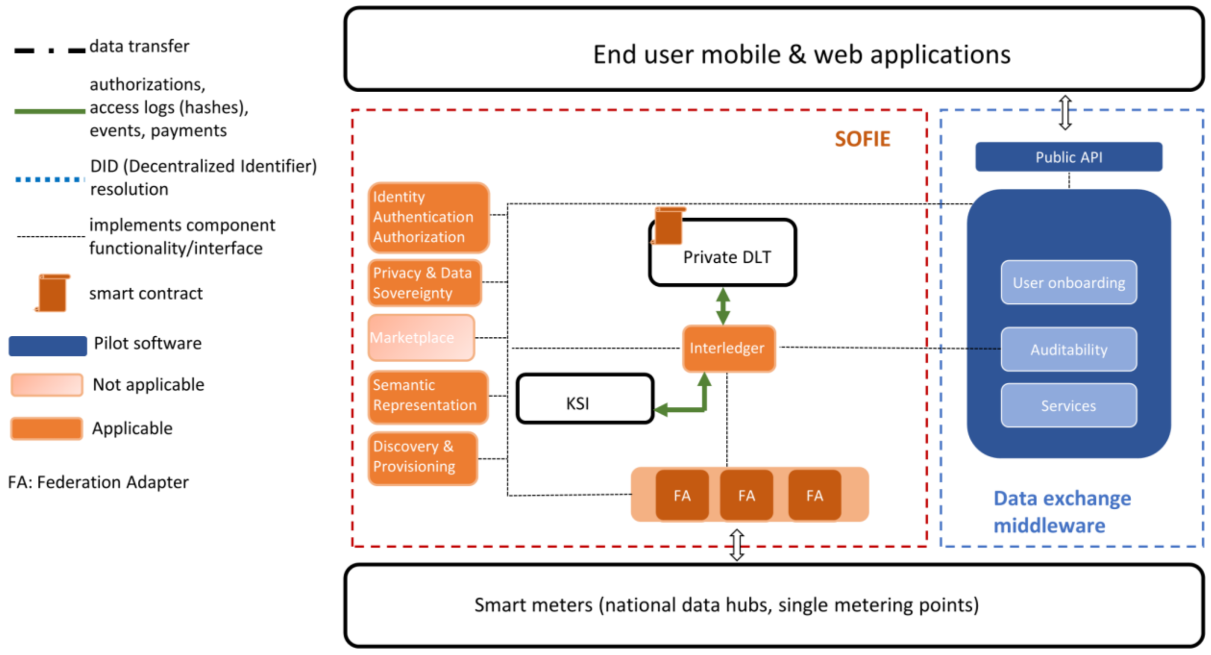
In the Decentralised Energy Data Exchange pilot, the SOFIE federation adapters will be used to enable data exchange with different smart meter systems:

- **National data hubs** - existing information systems having non-standard integration options. The existing data hub has information about users and their consumption history. Each data hub needs to be integrated separately.
- **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 also production data.

Consumption data is stored on a data owner and data hub level. When a secure connection is established between the parties, data exchange will be performed point-to-point.

End users will interact with the system through web interfaces and mobile applications. A middleware layer will provide different APIs for those applications to enable onboarding, interaction with SOFIE components and other activities required for secure data exchange. SOFIE components will be used to help manage service discovery, IAA, privacy and data sovereignty, and for following semantic representation rules.

An overview of the Estonian energy pilot architecture is shown in Figure 11.



data transfer
authorizations, access logs (hashes), events, payments
DID (Decentralized Identifier) resolution
implements component functionality/interface
smart contract
Pilot software
Not applicable
Applicable
FA: Federation Adapter
End user mobile & web applications
SOFIE
Identity Authentication Authorization
Privacy & Data Sovereignty
Marketplace
Semantic Representation
Discovery & Provisioning
Private DLT
Interledger
KSI
FA
FA
FA
Public API
User onboarding
Auditability
Services
Data exchange middleware
Smart meters (national data hubs, single metering points)

Figure 11. Decentralised energy data exchange pilot architecture

Table 12 briefly summarizes the use cases in the pilot and Figure 12 shows how the actors of the energy pilot interact with the system. More details of the pilot are provided in [Oik2019].

Table 12: Use cases of the decentralised energy data exchange pilot.

ID	Use case	Description
1	Configure access to metering data	Data owner initiates the connection to the SOFIE network to enable access to their smart meter data and related access rights.
2	Request metering data	A service provider is interested in providing energy service to data owner and needs access to the energy consumption data. After access rights have been granted, the service provider can start downloading the data from data owner and use this data to fulfil the contract.
3	Give access rights	Data owner can grant access rights for their data to service providers
4	Remove access rights	Data owner can revoke previously granted access
5	Request audit log	Any actor of the pilot can request an audit log of their activities and interactions with other parties.
6	Handle dispute	In case of dispute any actor of the pilot can get proof of their activities related to data exchange and granting access. Data integrity and time can be verified by an external expert.

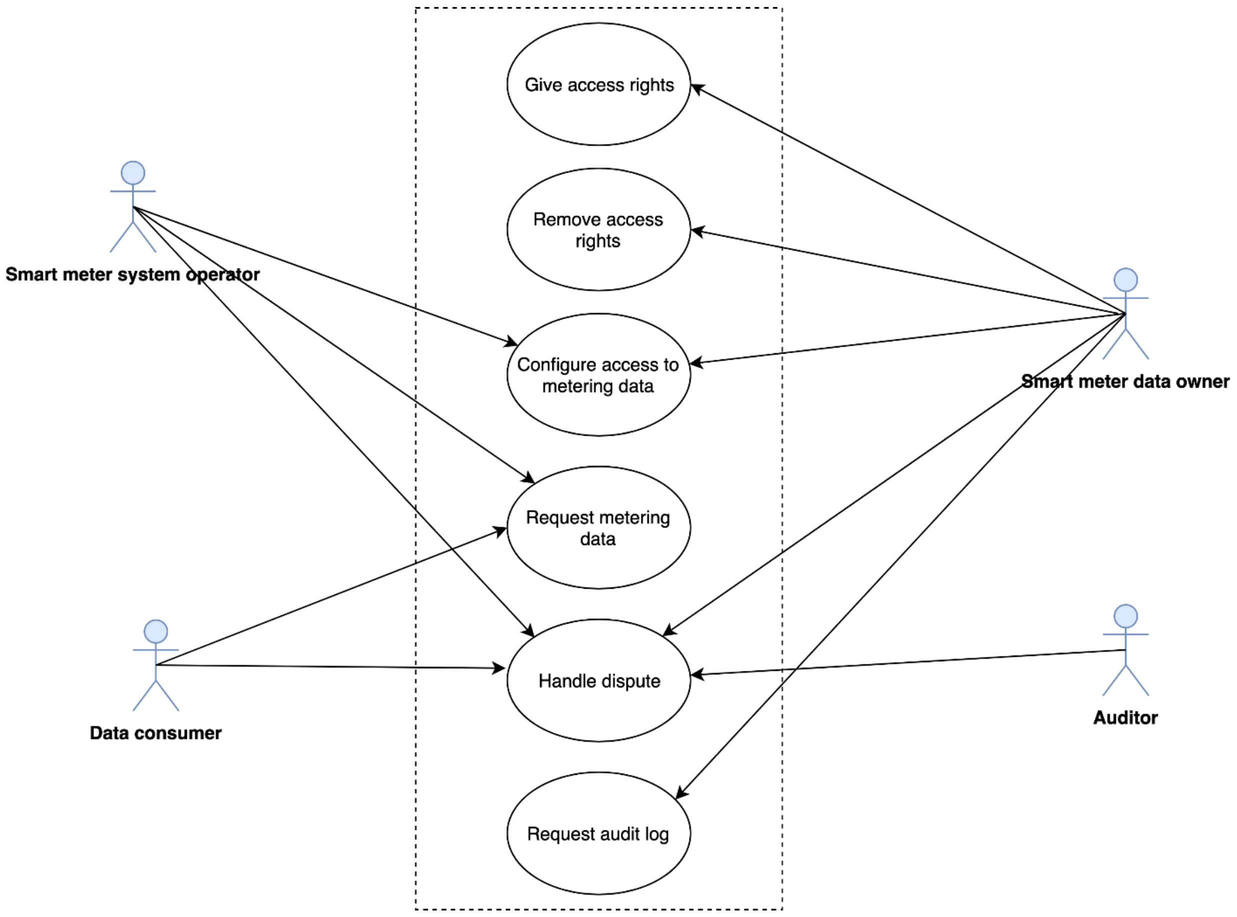


Figure 12. Decentralised energy data exchange pilot use cases

4. Framework Components

This section provides a high-level description of the SOFIE framework components that can be used to implement a specific system architecture for a pilot or any other system following the SOFIE framework architecture. Furthermore, this section also describes SOFIE federation adapters and application APIs. SOFIE deliverable "D2.5 - Federation Framework, 2nd version" due in August 2019 will describe the design and implementation of the components in more detail.

4.1 Interledger

The main purpose of the SOFIE interledger component is to enable transactions between actors and devices belonging to different (isolated) IoT platforms or silos. Each IoT silo either utilizes or is connected to one or more DLTs. The interledger component then enables interaction between these DLTs. By providing interledger transaction capabilities, SOFIE enables the semantic level communication between the different silos by connecting devices residing in different silos and their respective ledger realms.

Using different DLTs is often necessary because of the advantages and disadvantages each of them has. For instance, the Ethereum blockchain is very suitable to handle payments and automate tasks via smart contracts when specific conditions are triggered, such as a payment. Nevertheless, the Ethereum blockchain uses a consensus mechanism which causes delays in the execution of transactions, which might not be suitable for an IoT use case. On the other hand, the Hyperledger Fabric blockchain is permissioned and uses a Byzantine Fault Tolerant consensus mechanism, which makes transactions almost immediate to execute.

SOFIE's pilots and evaluation scenarios will utilize Ethereum, HyperLedger Fabric, Guardtime KSI blockchain and HyperLedger Indy. Cross-chain transactions can take different forms depending on the specific scenario and its requirements. For example, interactions between a public and a permissioned ledger can use hashed time-lock contracts to cryptographically link transactions and events on the two ledgers. In such a scenario, the public ledger can record payments while the permissioned ledger can record authorization transactions and events. Alternatively, hashes of records stored on the permissioned ledger can be periodically recorded on the public ledger in order to provide a timestamped anchoring point, exploiting the wide-scale decentralized trust provided by the public ledger. Finally, interactions between a public or permissioned ledger and a ledger storing DID documents can focus on the resolution of DIDs to DID documents. The interledger functionality can be implemented in different entities, which include the entities that are interacting, a third party, or multiple third parties. In the latter case, some coordination between the entities may be necessary. A detailed survey of interledger approaches is contained in [Sir2019].

Hashed Time-Locked Contracts (HTLCs) are one type of interledger mechanism. They rely on two mechanisms that are already widely used in the blockchain domain. The first mechanism is a hashlock. A party creates a transaction according to the specific needs and also a hash of it using a nonce and its private key, which makes the transaction spendable, and hence valid, only upon revealing the used nonce, which is known only by the transaction creator. This prevents the transaction receiver from spending the transaction unless the creator reveals the used nonce. In this context, revealing the nonce is associated with a condition verified on another blockchain. The second mechanism is timelock. A transaction containing a time-lock allow its creator to revert it in the case of the transaction not being finished within the agreed upon timeframe. Put together, Hashed Time-Locked Contracts allow for conditional transaction execution (hashed contracts) without holding the transaction creator resources involved in the transaction indefinitely in case the conditions are never met (time-locked contracts).

In SOFIE, HTLCs can be used to e.g. enable secure actuations, where a permissioned DLT first requires a cryptographic proof of payment before allowing access to the resource. Such payment can be performed on the public Ethereum blockchain, and the information stored on

Ethereum only contains a hash of the actual payment. The preimage of the transaction hash can then be used to prove the authenticity of the payment, so that the receiver can verify whether it fulfills the agreed upon conditions. If the conditions are met, access is granted to the IoT device or a set thereof. Furthermore, these transaction receipts are auditable by both the members of the consortium ledger i.e. the private DLT realm, and by the independent auditors. The public blockchain also extends the non-repudiation guarantees of the private DLT so that it becomes resistant even to collusion of the consortium ledger members.

If the federated IoT silo relies upon a consortium ledger, these consortium ledgers can be connected via SOFIE to the degree allowed by both the device owner, and the connected ledger governance or owner, provided that the silo has been enabled to support SOFIE federation.

4.2 Identity, authentication, authorization

The goal of the Identity, Authentication, Authorization (IAA) component is to provide mechanisms that can be used for entities' and services' identification and authentication, and consumers' authorisation. To this end, it supports the following Identification/Authentication mechanisms: URIs (e.g., Web of Things URIs) for identification coupled with digital certificates for authentication, usernames for identifications bounded to secret passwords used for authentication, and *decentralized identifiers (DIDs)* associated with a *DID document*[3], and used for authentication. A popular DID implementation, also considered by our component, is Hyperledger Indy.[4] Consumers' authorisation is primarily implemented with the widely-used OAuth2.0 protocol. The IAA component supports vanilla OAuth2.0, OAuth2.0 tailored for constrained devices (as defined by the IETF ACE-group), and OAuth2.0 combined with DIDs. Furthermore, it supports various token types and encodings. In addition to OAuth2.0, the IAA component supports the UMA (User-Managed Access) protocol.

The IAA component can use smart contracts in order to link authorization decisions with payments, as well as for logging transaction-specific information that can be later used for auditing and dispute resolution. Moreover, authorization decisions can be linked to IoT events that are recorded on the blockchain.

4.3 Privacy and Data sovereignty

The goal of the Privacy and Data sovereignty component is to enable data sharing in a controlled and privacy preserving way. This component considers privacy preservation as a two dimensional problem. The first dimension concerns the privacy of the data *provider*, whereas the second dimension concerns the privacy of the data *consumer*. Data provider privacy is related to the amount and the accuracy of information a 3rd party (including the consumer) can deduce about the provider from all the available data. This can be achieved by reducing or obfuscating the data stored on a public ledger. A mechanism to reduce the data is to store only hashes on a public blockchain, while the actual information is stored in private/permissioned ledgers. Mechanisms to obfuscate data include *differential privacy* mechanisms. In particular, this component enables election of a special purpose node that acts as a *data accumulator* which is in charge of adding *noise* to the (encrypted) collected data. An alternative can be adding noise directly at the sources; however, in order to achieve the required amount of privacy and accuracy of the results, this approach requires a large number of sources. The coordination among the entities, namely the data provider, data consumer, and data accumulator, is achieved through a smart contract. Consumer privacy is related to the amount and the accuracy of information a 3rd party (including the provider) can deduce about the consumer during the authentication, authorization, and payment processes, and is enabled through the use of *verifiable credentials*. To this end, our component supports attribute-based access control

[3] Organizational DID documents may be stored in the ledger for extra security, auditability, and availability.

[4] https://www.hyperledger.org/projects/hyperledger-indy

where consumers can prove the possession of some attributes using *verifiable credentials* and *zero-knowledge proofs*. The underlying mechanisms support the minimum disclosure of information necessary to obtain a service. Additionally, multiple identifiers can be used to further improve privacy.

Data sovereignty is achieved through access control mechanisms. Our component supports two access control schemes, namely access control through delegation to an *authorization server* [Fot2018], and crypto *token-based* access control imposed by smart contracts. The first scheme enables data owners to define an authorization server (AS), i.e., a special type of mediator that vouches about the eligibility and/or handles payments made by a consumer to access a particular resource. In this scheme a smart contract is used as an AS registry, which handles payments and can verify that an AS is indeed authorised by an owner to implement the access control policy. Verification of the AS can be performed using verifiable credentials. The second scheme leverages blockchain-backed crypto tokens and enables owners to define access control policies based on these tokens. Crypto tokens can be granted only through a blockchain transaction and blockchain-specific functions, such as transfer, aggregation, etc, can be applied on these tokens. On the other hand, an access control policy can be verified either by interacting with a smart contract in the blockchain or by executing the smart contract locally.

4.4 Semantic Representation

Communication between SOFIE framework components and different IoT devices requires common understanding of the Thing, Service, and Data descriptions. The *Thing Description* (TD) allows other entities to discover the properties of the Things and Services using commonly agreed definitions. Similarly, the Data Model used in the SOFIE components and other entities defines the way the data is structured. To be semantically interoperable, all devices and software components have to use similar descriptions.

The Semantic Representation component handles the required TD and data model processing as well as the potentially needed translations between different data models.

4.4.1 Service and Thing descriptions

The goal of the Service/Things description model of SOFIE framework is to define a common representation model for IoT Things and services that enables interoperability and automation in the deployment of enabling services and applications on top of federated IoT environments. To this end, the SOFIE framework makes use of the W3C WoT Things Description model [Kae2019].

WoT relies on well-established web technologies and RESTful interfaces to expose IoT Things, services and resources. It introduces a conceptualization of Web resources into the IoT world by modeling the notions of *Thing Description* and *Interaction*, as the core WoT resources. By making use of W3C semantics vocabulary, the WoT TD model accomplishes the following two critical objectives: i) it describes Things instances with general metadata (such as name, ID, human-readable information etc.) and makes these descriptions exchangeable with other agents, and ii) it exposes Things to the Web through a set of interactions, which correspond to their interface to the physical world. In particular, the model is based on the following technologies:

- Semantic metadata for the Thing itself by using WoT semantics vocabulary.
- An interaction model based on WoT's Properties, Actions, and Events paradigms.
- A JSON based semantic schema to make data models machine understandable.
- Several web linking features to establish relations among Things.

In SOFIE, WoT TD modeling is implemented as part of the Federation Adapter which is responsible for adapting and annotating the corresponding, federated IoT environment. The hierarchical view of the used classes and fields is depicted in Figure 13. The WoT TD model can be seen as the *index.html for Things,* as it provides the entry point to the SOFIE discovery and provision services. As already mentioned, the used *information model* is based on the W3C semantic vocabulary, which is split into three independent parts referring to the TD core model, the TD data schema model, and the TD security model, respectively. The used representation format is JSON-LD that enables more advanced and enriched semantic processing than raw JSON (which is also supported) of the metadata and is also aligned with Linked Data.

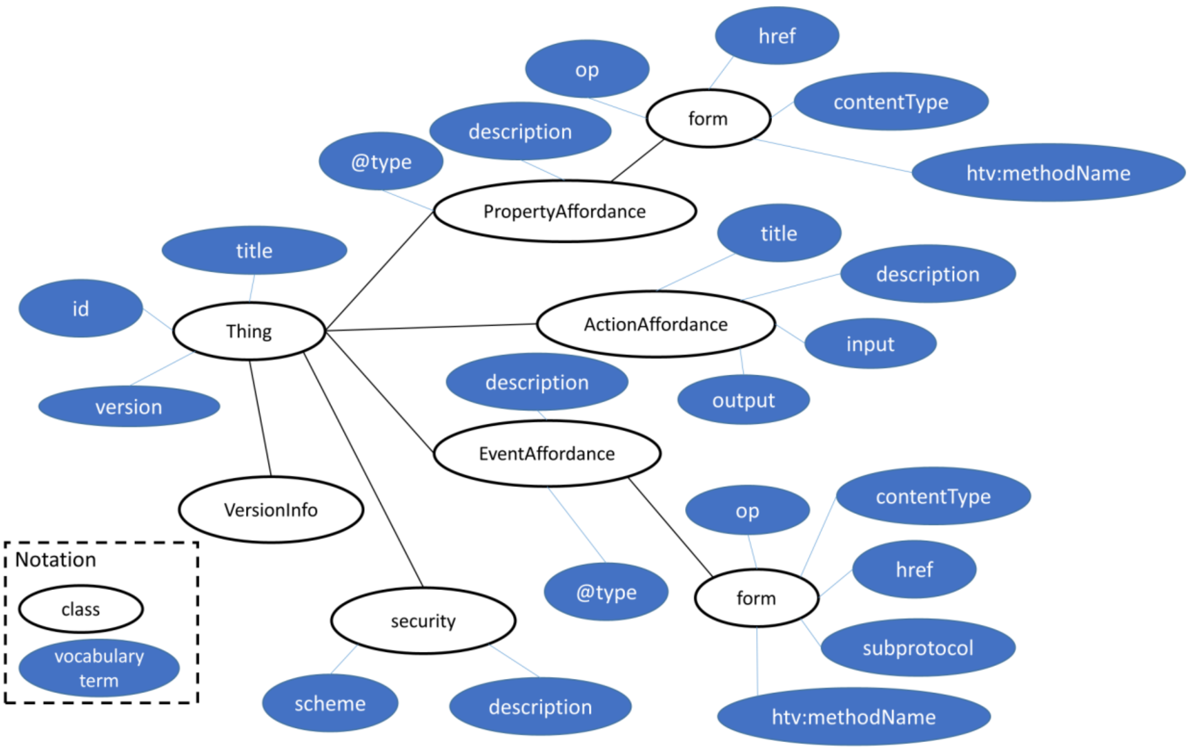


Figure 13. Overview of TD classes and vocabulary used by SOFIE

4.4.2 Data description

Data description can be divided into two separate models: a high level information model and a more detailed data model. In this document, guidelines are given about what is expected from the information and data models. SOFIE deliverable D2.5 will contain description of high level information models, while the detailed data models will be fully described in SOFIE deliverable “D2.7 - Federation Framework, final version”.

SOFIE Information Model

The information model describes each component’s functionality as well as the interfaces it uses to communicate with other components. This model does not go into implementation details, but remains on a high level to give the reader an understanding of the capabilities that the corresponding component provides. In this document, the high level descriptions of the SOFIE components are given in Section 4. SOFIE deliverable D2.5 will have a more detailed description of the components used in the SOFIE platform. It will also describe the interfaces that the components have.

Data Models

Following the defined Information Model, separate Data Models are generated for different implementations, i.e. in SOFIE for both platform internal data flows as well as for pilot project internal and external communications. These descriptions go more into implementation details, describing the protocol usage in communication, the format of data and the serialization of data.

Each component needs to define the data model it is using. This is essential for the other components in the system to understand the data that they will retrieve. In addition to data formats, the data description will also contain various information about e.g. data capturing, amount of data captured, and business relations.

Data retrieval and handling

The TD defines the services that are provided by the Things. In addition, the information model describes the interfaces that are used by the components and entities. Using this information any entity can communicate with any other entity.

Once the data is retrieved, the parsing of data can be done using the defined data model. As it is unlikely that all components and entities use the same data model, a translation may be needed from one model to another. This will be done in the Semantic Representation module.

4.5 Marketplace

The goal of the SOFIE marketplace component is to enable the trade of different types of resources (e.g. electricity for charging a vehicle) in an automated, decentralized, and flexible way. In this context, a decentralized marketplace is a marketplace that does not have a single entity owning or managing it, which in turn increases competition and enhances its security, resiliency, transparency, and traceability. The decentralized marketplace can be partially decentralized, when e.g. a group of independent agriculture producers and retailers are managing it, or fully decentralized where anyone can join and use the marketplace.

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

Resources exchanged on the marketplace can include both physical and virtual goods such as energy, access to data, actuation, or spaces, in-game assets, and cryptocurrencies.

The main functionality of the SOFIE marketplace is to:

- Allow actors to list resources on the marketplace and bid for them.
- Allow actors to view and update resource descriptions.
- Match bids and offers.
- Provide evidence that the trade has been carried out and resources have been correctly exchanged.
- Keep history of all trading actions (such as offers, bids, resource descriptions, transactions, etc.).

The SOFIE marketplace is implemented on top of an Ethereum blockchain utilizing smart contracts, though the marketplace may also interact with other kinds of DLTs. The usage of a DLT facilitates interoperability between the different actors by providing high availability for shared immutable data, provides a rapid and user-friendly mechanism to negotiate micro-contracts, and affords security, transparency, and auditability.

In the future, the SOFIE marketplace will also support various sophisticated algorithms to implement dynamic pricing models, in addition to simple auction-like bids and offers.

4.6 Provisioning and Discovery

In a network scenario, provisioning and discovery protocols make it possible for client applications to use the services and devices available on the network. This applies to both hardware resources (e.g. network printers) and software (e.g. multimedia streaming). In this scenario, we can easily identify the following roles:

- **Service Providers:** network nodes offering a specific service
- **User Agents (or Clients):** devices using a service offered by service providers
- **Service Brokers:** network nodes in charge of coordinating the way services are discovered and providing information about the services present in the network.

From the architecture point of view, we can classify services into *Centralised services*, where clients send requests to selected devices acting as service brokers, which reply with information about the required services (typically, the location of the service on the network), and *Decentralised services* in which each client broadcasts its requests and service providers send back replies accordingly.

In SOFIE, a hybrid approach will be used: the *Federation Adapters* act as local service brokers within a single framework implementation, while a central *Service Broker* communicates with the adapters, interrogating them to gather the required service descriptions adopting the WoT Thing Description (TD) model: an abstraction describing physical or virtual entities interacting in the web of things.

4.7 Federation Adapter

The purpose of the federation adapter is to interface with existing IoT platforms. This allows the IoT platforms to interact with SOFIE without requiring any changes to the IoT platforms themselves. Among other things, the federation adapter will adapt the data representation of the corresponding IoT platform to the semantic representation model used by SOFIE. Different SOFIE deployments will utilize different IoT platforms, and therefore use different types of federation adapters. SOFIE deliverable D2.5 will describe in more detail federation adapters used by SOFIE pilots.

4.8 Application APIs

SOFIE application APIs help connect the SOFIE framework with applications and services, such as dashboards, user and management interfaces, web and mobile clients, etc. Application APIs will be described in more detail in upcoming SOFIE deliverable D2.5.

5. External Components and Interfaces

This section describes the external components and interfaces used by SOFIE.

5.1 Web of Things (WoT) discovery

The WoT model[5] enables the retrieving of lists of Web Things in response to an HTTP GET request on the destination URL of a things link.

The provisioning and discovery SOFIE component, unifying the different IoT platforms under the same TD model, enables interoperability with existing WoT-compliant platforms.

5.2 FIWARE

The FIWARE platform provides a group of powerful set of APIs that ease the development of Smart Applications in multiple vertical sectors[6]. In context/data management, FIWARE delivers a number of Generic Enablers (GEs) to collect, exchange and analyse data in an efficient way[7]. One of the core GEs is the Orion Context Broker, an implementation of the NGSIv2 REST API[8].

The Context Broker operates together with different platform components, supplying context data (from IoT sensors for example), processing, analyzing or visualizing data.

The usage of the Orion Context Broker is the minimum requirement for an application to qualify as “Powered by FIWARE” so, in order to communicate with external platforms using FIWARE, the provisioning and discovery SOFIE component is compatible with the NGSI v2 specifications[9]. In this way, any other siloed platform compliant with FIWARE will be able to be "SOFIE-compliant" through the NGSI broker.

[5] http://model.webofthings.io
[6] https://www.fiware.org
[7] https://www.fiware.org/developers/
[8] https://fiware-orion.readthedocs.io/en/latest/
[9] https://swagger.lab.fiware.org/?url=https://raw.githubusercontent.com/Fiware/specifications/master/OpenAPI/ngsiv2/ngsiv2-openapi.json

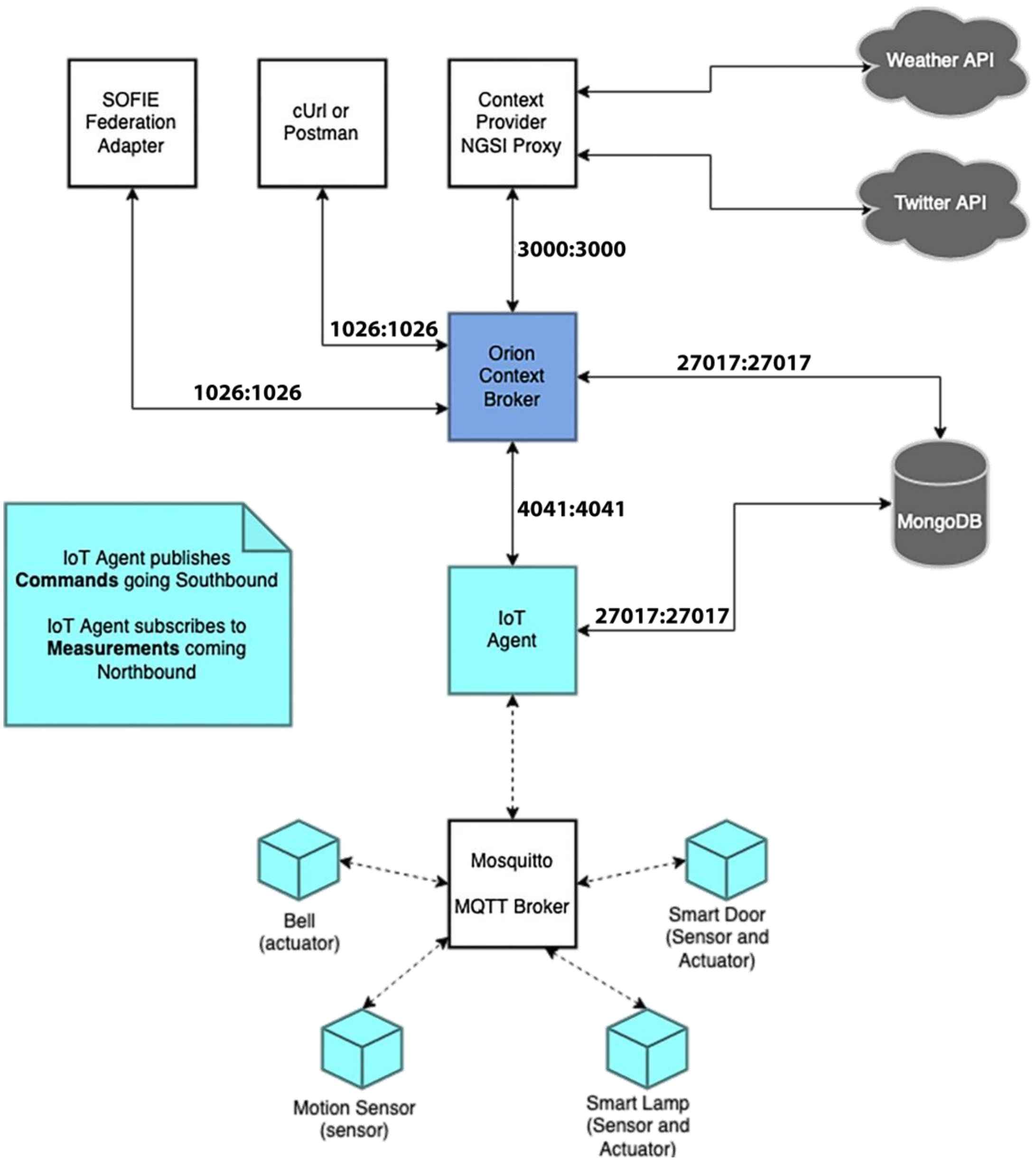


Figure 14. Sample IoT over MQTT architecture based on FIWARE components[10]

Figure 14 shows an example FIWARE architecture using the Orion Context Broker and the JSON IoT Agent. The IoT Agent is the FIWARE component acting as a bridge between simple JSON protocol and NGSI Context Brokers. The SOFIE Federation adapter is connected to the Context Broker.

[10] https://fiware-tutorials.readthedocs.io/en/latest/iot-over-mqtt/index.html

6. Deployment Considerations

This section discusses the different options for integrating existing systems with SOFIE.

6.1 Component Categories

While a specific deployment architecture cannot be mandated (deployment is within an organization's own control boundary), it is still possible to describe some typical deployment scenarios, which will in turn assist discussions on concrete technologies and implementation architectures.

There are essentially two different approaches to a deployment: 1) organizational silos extended with SOFIE-compatible interfaces and 2) building a SOFIE-enabled system from scratch.

Overall, different components within the system can be described as:

- **Adapters** when they implement a SOFIE interface either as inbound (service) or outbound (client) protocol, and they offer an open interface for the organization or developer to integrate into. An example of this type would be a code library implementing a SOFIE interface client code, and the business logic would use this library to interface with a SOFIE-compliant system.
- **Translators** when they implement two different interfaces (a SOFIE one and another one), contain both the client and server capability, and translate one protocol to another. An example of this would be a program that serves a SOFIE-compliant interface for retrieving data from IoT devices, and forwards these requests to another system.
- **Gateway**, while technically also a translator, would be a component that talks to a specific type of system, for example, an IoT gateway.
- **Native**, e.g. they implement a SOFIE-compliant interface directly.

In most cases SOFIE will utilize existing protocols, for which widespread implementations for interfacing are likely to already exist. From SOFIE's point of view, even if an organization uses these existing implementations, they would be categorized as native interfaces. "Adapters" and "translators" within SOFIE's scope are relevant only for protocols or combinations of protocols that are SOFIE-specific (see the SOFIE Framework document D.2.5). Note that this means that a set of protocols may be standard, but there exists a SOFIE adapter implementation that combines different protocol implementation to provide a specific, more narrowly defined functionality (such as a specific business platform).

6.2 Extending existing systems

An important consideration for the SOFIE architecture is how well it is suited for use with existing (aka legacy) systems. It is unlikely that a new system will be developed entirely from scratch - more likely it will be an adaptation of an existing system, or a new component that employs existing systems and interfaces. For this purpose, Figure 15 shows some potential approaches that can be taken when the goal is to add SOFIE-compliant interfaces to an existing system. Each of the approaches a-f is discussed below.

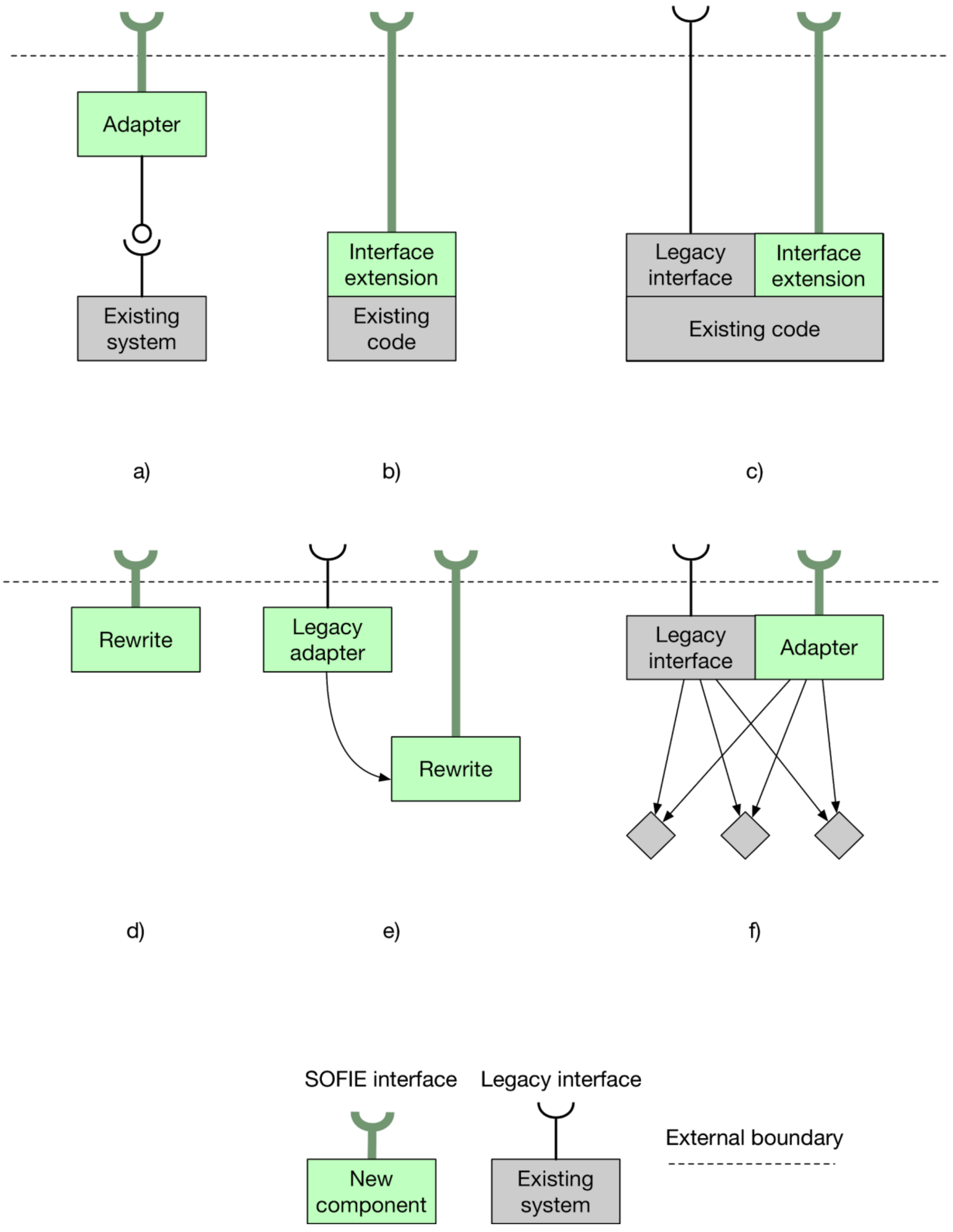


Figure 15: Different approaches to extending existing systems for SOFIE specification compatibility.

Case a: A separate adapter service (or a translator, depending on the complexity of the task) is developed, it connects to the existing service interfaces and provides a new interface. It is possible that some operations on the new interface do not have a corresponding primitive operation on the existing interface, requiring the adapter to be able to perform multiple operations on the legacy system to provide support for the new interface.

Case b: Alternatively, the new interface can be implemented directly on the existing service.

Case c: If the old interface needs to be supported, one possibility is to add a new interface that co-exists with the existing interface. This may require interlocking between the interfaces to ensure consistency.

Case d: One option is always to completely rewrite the existing system from scratch. As noted earlier, this is often not a realistic approach unless the service being replaced is lightweight.

Case e: Even if a rewrite is possible as in previous case, it may be necessary to support the old interface for legacy clients.

Case f: If the legacy service is a front to existing services such as a network of IoT devices, one option is to let the new interface access the backing resources directly while maintaining the old interface for compatibility reasons.

Eventually the approach taken depends on the particulars of each case and no specific approach can be recommended or assumed.

References

[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

[Kae2019] S. Kaebisch, T. Kamiya, M. McCool, and V. Charpenay. "Web of Things (WoT) Thing Description," W3C Candidate Recommendation 16 May 2019, May 2019. Available at: https://www.w3.org/TR/wot-thing-description/.

[Mac2006] C. M. MacKenzie et al. "OASIS Reference Model for Service Oriented Architecture 1.0", OASIS Standard, 12 October 2006. Available at: http://docs.oasis-open.org/soa-rm/v1.0/.

[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.

[Ree2019] D. Reed, "Decentralized Identifiers (DIDs) v0.13 – data model and syntaxes for decentralized identifiers (DIDs)," W3C Community Group Draft Report, June 2019. Available at: https://w3c-ccg.github.io/did-spec/.

[Sir2019] V. A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, G. C. Polyzos, "Interledger Approaches," accepted for publication in IEEE Access, 2019.