SOFIE - Secure Open Federation for Internet Everywhere

779984

Deliverable 2.6

SOFIE Federation Architecture, Final 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	31.10.2020
Author(s)	Yki Kortesniemi (AALTO), Tommi Elo (AALTO), Dmitrij Lagutin (AALTO), Lei Wu (AALTO), Nikos Fotiou (AUEB-RC), Giuseppe Raveduto (ENG), Mait Märdin (GT), Filippo Vimini (LMF), Ahsan Manzoor (ROV), Yannis Oikonomidis (SYN)
Responsible persons	Yki Kortesniemi (AALTO), Yki.Kortesniemi@aalto.fi
Target Dissemination Level	Public
Status of the Document	Completed
Version	1.0
Project web-site	https://www.sofie-iot.eu/

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

Table of Contents

List of abbreviations

API	Application Programming Interface
AS	Authorisation Server
BLE	Bluetooth Low Energy
DEFM	Decentralised Energy Flexibility Marketplace
DID	Decentralised Identifier
DLT	Distributed Ledger Technology
DNS	Domain Name System
DNS-SD	DNS Service Discovery
DSO	Distribution System Operator
EDEX	Energy Data Exchange
FA	Federation Adapter
FM	Fleet Manager
FSC	Food Supply Chain
GE	Generic Enabler
HTLC	Hash Time-Locked Contract
IAA	Identity, Authentication, Authorisation (Architecture component)
IL	Interledger (Architecture component)
IoT	Internet of Things
JSON	JavaScript Object Notation
KSI	Keyless Signature Infrastructure
MP	Marketplace (Architecture component)
MQTT	Message Queuing Telemetry Transport
P&D	Provisioning and Discovery (Architecture component)
PDS	Privacy and Data Sovereignty (Architecture component)
PEP	Policy Enforcement Point
PoI	Point of Interest
RBAC	Role Based Access Control
RFID	Radio Frequency IDentification
SR	Semantic Representation (Architecture component)
SSI-agent	Self-Sovereign Identity agent
TD	Things Description
TSO	Transmission System Operator
VC	Verifiable Credentials
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
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 thus harnessing the individual strengths of different ledgers*. Finally, smart contracts allow the automation of many transactions, and thus lower the operating costs of the system.

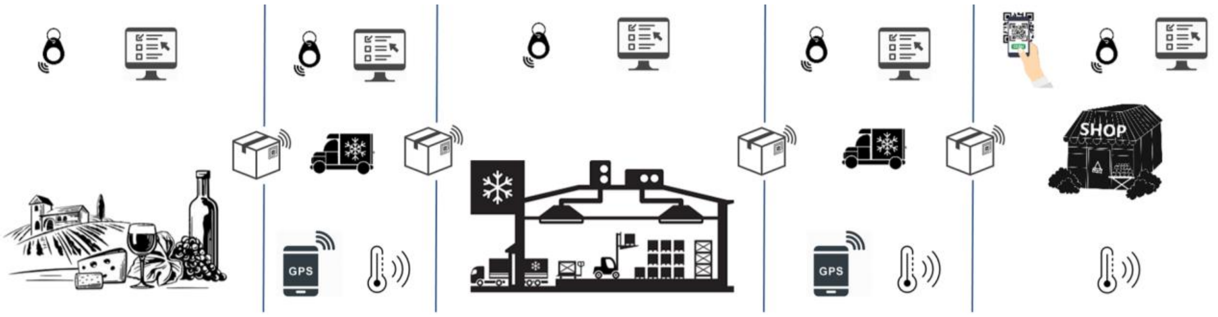


Figure 1. The SOFIE food supply chain pilot collects data as the produce moves from the farm through transporters and distributors to the supermarket.

A key benefit of the SOFIE Architecture is that it allows the creation of solutions, which 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 all the way from the field to the consumer can be recorded as it moves along the supply chain, providing accurate information to customers, while helping companies in dispute resolution.

As shown in Table 1, architectures can exist on many levels, such as framework, system, and component architecture levels. This document describes the framework-level *SOFIE Architecture,* which provides a high level overview of the SOFIE components and federation adapters, and the interactions between them. The system-level architectures specific to each SOFIE pilot are then briefly described in Section 3 of this document, with more details provided in 'Deliverable 5.3 - End-to-end Platform Validation' [D5.3]. Finally, the SOFIE Framework [Framework], an open-source software implementation of the SOFIE Architecture for use in the SOFIE pilots, is described in the 'Deliverable 2.7 - Federation Framework, final version' [D2.7], due in December 2020. In the rest of this document the term *architecture* refers to the framework-level SOFIE Architecture unless otherwise specified.

Table 1. Multiple levels of architecture

Architecture Level	Scope	Level of Detail	SOFIE deliverable
Framework	SOFIE	Broad	This document (D2.6)
System	SOFIE Pilot	Pilot specific	D5.3
Component	SOFIE Component	Internal structure of SOFIE components and adapter	D2.7 (December 2020)

This document supersedes the previous Architecture deliverable ‘D2.4 - SOFIE Federation Architecture, 2nd version’ [D2.4]. The structure of this deliverable is as follows: Section 2 presents the SOFIE Architecture as well as the requirements it was based on. Section 3 describes the system architectures of the four SOFIE pilots. The SOFIE Framework components and federation adapters are described in Sections 4 and 5, respectively. Section 6 discusses integrating external components to the SOFIE Architecture and, finally, Section 7 summarises the SOFIE Architecture.

2 SOFIE Architecture

The SOFIE Architecture is a *framework architecture*, i.e. an architecture that *defines types of functionalities but not a single exact implementation* for those functionalities due to the fact that SOFIE can be used in so many types of applications in different fields that no single set of functionalities or APIs is capable of serving them all. Therefore, the architecture is designed as a foundation that can be built upon to define suitable functionalities and interfaces for each application domain.

The SOFIE Architecture is also a *modular architecture*, i.e. it consists of 6 components (introduced in Section 2.2 and detailed in Section 4), some or all of which can be utilised in any particular system, depending on the application requirements. The SOFIE pilots and some additional use cases were used to identify the 6 components that perform *key functions for IoT federation* and are, therefore, common to a large part of federation use cases.

Finally, one of the most fundamental design goals for the SOFIE Architecture 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 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 federations to be IoT- and ledger-agnostic, SOFIE enables interoperability across technology silos. To this end, the ledger-related Interledger component supports *ledger adapters* that make it easy to add support for new types of ledger, and the Architecture also includes the concept of *Federation Adapters* (introduced in Section 2.2 and detailed in Section 5) that link IoT systems to the Architecture.

Together, the modular framework architecture and adapters make it easy to adapt the SOFIE Architecture to different IoT federation situations.

The *SOFIE Framework* described in SOFIE Deliverable ‘D2.7 - Federation Framework, final version’ [D2.7] provides example implementations of each of the components and the federation adapters as open-source software [Framework]. The provided examples are tailored to the requirements of the SOFIE pilots, but they can be freely adapted and expanded to suit the needs of other applications, as the SOFIE Architecture supports customising the components to the needs of the individual application areas.

2.1 Requirements for the SOFIE Architecture and Framework

Requirements for the SOFIE Architecture were gathered from different sources: during the first half of the project, meetings were organised between pilot consortium members and end-users of all relevant application domains to capture and define business and end-user requirements. In particular, in the energy domain, several meetings and sessions were organised with TSOs and DSOs (Transmission and Distribution System Operators) experts from cross-functional areas, especially from Estonia and Denmark, to gather industrial requirements and to get insights about recent standardisation directions at EU level, while ASM Terni, as a consortium member of SOFIE, provided valuable feedback on prioritising needs and achieving a consensus in setting up the final requirements. In the food supply chain domain, the “7 grapes–Pegasus Coop” company was subcontracted as an end user and early adopter of the corresponding pilot to transfer knowledge about food supply chain business operations and to assist in end-user requirements elicitation. Finally, in the mobile gaming domain, several hackathons were organised both internally in Rovio, a consortium member, and with external experts in digitisation and gaming customer services to identify business opportunities of using DLT and IoT in mobile gaming and to understand customer needs. Overall, through these processes, all relevant types of end-users to the SOFIE pilots have been invited to discuss and to identify pilot-oriented, end-user requirements.

Once collected, these requirements were further analysed by the SOFIE technical partners to identify system requirements for the SOFIE Architecture and Framework components. 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. However, due to privacy requirements and the need to maintain business secrets, it is not feasible to store all data to a single (public) DLT - instead, multiple DLTs should be used: all data is stored in a private ledger and then a hash of a transaction tree is stored to the public ledger as a trust anchor, which increases the security, transparency, and auditability of the system, as parties cannot modify existing transaction logs after the hash has been publicly revealed. Furthermore, several DID operations rely on a related DLT (e.g. Hyperledger Indy based DIDs rely on a permissioned ledger to manage information related to public DIDs), therefore in order to connect identifier creation, authorisation, and authentication functionality to the rest of the SOFIE Framework, interaction between the DLT ledger and other ledgers used by SOFIE is necessary. Finally, various DLTs have trade-offs in terms of throughput, latency, cost and scalability; therefore, interledger operations between the DLTs must be supported by SOFIE.

Tables 2-4 list the requirements for the SOFIE Architecture, the privacy-related requirements for the implementation and deployment of the SOFIE Architecture, and the requirements for the SOFIE Framework. Each requirement is associated with a unique reference ID, a short description, a priority level, and a category. Six categories are used (QUALITY, AUDITABILITY, INTEROPERABILITY, USABILITY, SECURITY, POLICY & REGULATION). Two priority levels are used 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 2. 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.	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 all the architectural requirements* (RA01 - RA08 in Table 2), as explained below. Section 2.2 of this document explains how the SOFIE Architecture separates data management, control and representation, hence fulfilling the first requirement (RA01). The SOFIE Architecture consists of six pluggable framework components that can be used independently depending on the required functionality (RA02). The interfaces of framework components and federation adapters are defined and documented in SOFIE Deliverable 2.7 (RA03). Requirements RA04 and RA05 are satisfied through the usage of DLTs in the SOFIE Architecture, while requirements RA06 and RA08 are satisfied by the SOFIE Identity, Authentication, Authorisation (IAA) component. The SOFIE Privacy & Data Sovereignty (PDS) component fulfils requirement RA07 (principle of least privilege) by providing different APIs and interfaces and related access control policies for different uses.

Table 3. Privacy requirements for implementation and deployment of the SOFIE Architecture

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

Though satisfying this requirement is ultimately up to each implementation and deployment, the SOFIE Architecture *provides all the necessary tools* to this end. For instance, using the Interledger component it is possible to store all confidential information only in each organisation's private storage and then store a hash to a public ledger for increased trust. If a dispute requires auditing the information in the private storage, the hash proves that the data was not modified after the hash was published. Further, the Privacy and Data Sovereignty component also supports privacy preserving surveys using differential privacy, thus avoiding the storage of privacy compromising information in the first place.

Table 4. Requirements for the SOFIE Framework

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 authorisation tasks for their resources.	MUST	OPERATIONAL
RF04	The IAA component must provide users the capability to revoke authorisations.	MUST	SECURITY
RF05	The IAA component must allow individuals to control their personal information and digital identities (e.g. support self-sovereign identity technology).	MUST	SECURITY
RF06	The IAA component must support secure, tamper-proof, and verifiable logging of transactions and events.	MUST	SECURITY

RF07	The IAA component must support Role Based Access Control (RBAC).	MUST	SECURITY
RF08	Cryptographic algorithms used by SOFIE should be open-source, transparent, and as independent as possible of any particular architecture.	SHOULD	AUDITABILITY
RF09	SOFIE should support the execution of authorisation and authentication functionality on devices with constrained processing, storage, battery, and network connectivity.	SHOULD	OPERATIONAL
Privacy & Data Sovereignty			
RF10	SOFIE must follow the data minimisation principle for personal data and only request or process what is necessary for the situation and purpose.	MUST	OPERATIONAL
RF11	Processing of individual’s personal data is justified by a valid legal basis, e.g. a valid consent from the individual.	MUST	POLICY & REGULATION
RF12	Consent to process personal data must be revocable at any time.	MUST	POLICY & REGULATION
RF13	SOFIE must allow organisations and actors to manage (create, update, delete) their own data privacy policies.	MUST	POLICY & REGULATION
RF14	SOFIE should support user privacy even when aggregate statistics are made public (e.g. using differential privacy mechanisms).	SHOULD	POLICY & REGULATION
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 standardised 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

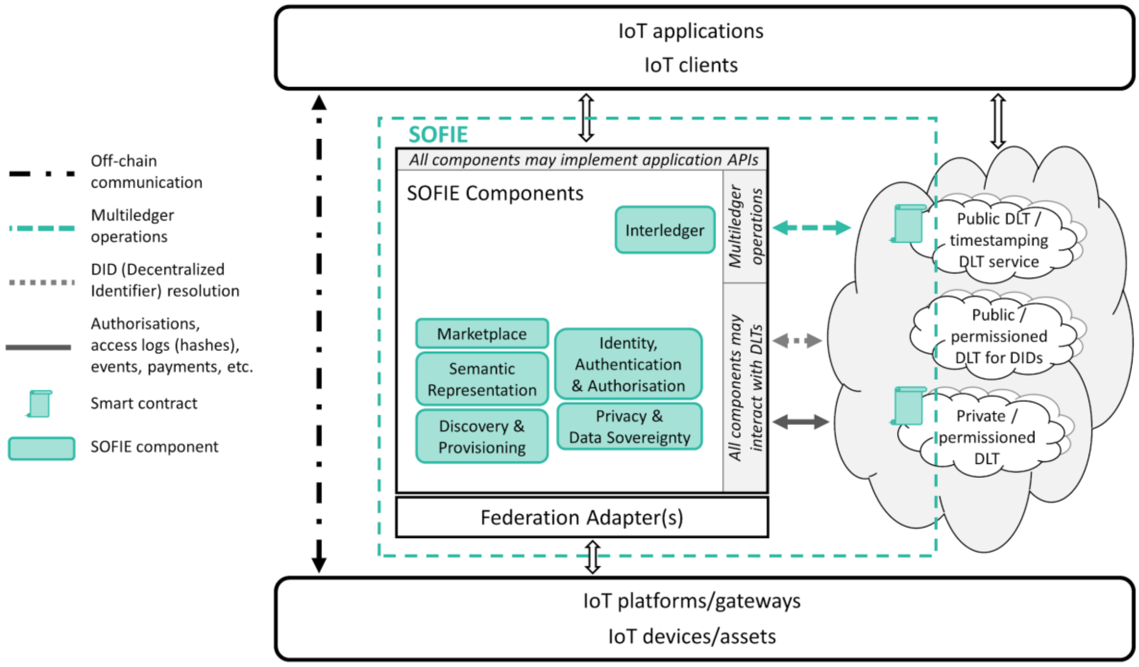
RF21	The marketplace must provide actors the capability to post/claim offers and sell/negotiate/exchange/buy resources and digital objects.	MUST	INTEROPERA-BILITY
RF22	The marketplace must support transparent trading of resources, i.e. the bids/offers matching process and the payments must be transparent.	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
Federation Adapter			
RF25	SOFIE deployments can utilise one or more Federation Adapters each capable of representing one or more IoT Devices/Platforms.	MUST	OPERATIONAL
RF26	The IoT device/platform must be able to utilise all the SOFIE functionalities it requires through the Federation Adapter representing it.	MUST	OPERATIONAL
RF27	Federation Adapters must not require changes to the IoT device/platform it represents.	MUST	OPERATIONAL

Validation of the requirements for the SOFIE Framework is reported in deliverable ‘D2.7 - Federation Framework, final version’ [D2.7].

2.2 Architecture overview

The key role of the SOFIE Architecture is to enable the secure federation of IoT systems. Figure 2 provides a functional overview of the SOFIE Architecture. In particular, it depicts the six components that provide the SOFIE functionality (green boxes) and the Federation Adapter(s) used to interact with the IoT platforms and devices.

The lowest level of the architecture contains the *IoT assets* (or resources), that include e.g. IoT sensors for sensing the physical environment, actuators for acting on the physical environment, and boxes with RFID tags 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, as well as servers providing IoT services.



IoT applications
IoT clients
Off-chain communication
Multiledger operations
DID (Decentralized Identifier) resolution
Authorisations, access logs (hashes), events, payments, etc.
Smart contract
SOFIE component
SOFIE
All components may implement application APIs
SOFIE Components
Interledger
Multiledger operations
Marketplace
Semantic Representation
Discovery & Provisioning
Identity, Authentication & Authorisation
Privacy & Data Sovereignty
All components may interact with DLTs
Federation Adapter(s)
Public DLT / timestamping DLT service
Public / permissioned DLT for DIDs
Private / permissioned DLT
IoT platforms/gateways
IoT devices/assets

Figure 2. The SOFIE Framework architecture

The *federation adapter(s)* are used to interface the IoT platforms with the SOFIE Architecture. This allows the IoT platforms to interact with SOFIE *without requiring any changes to the IoT platforms* themselves. Different scenarios and pilots can utilise different types of federation adapters, which expose only the required parts of the SOFIE functionality to the IoT platform.

Of the six components, the architecture emphasises the *Interledger component* responsible for interconnecting the different types of DLTs, which can have quite different features and functionality. Public (or permissionless) DLTs can offer wide-scale decentralised trust and immutability, but this necessitates a large network with many peers and a more demanding consensus mechanism, and thereby incurs a higher overall computational cost that will lead to higher costs and longer transaction confirmation times. Permissioned or consortium DLTs, on the other hand, have a lower transaction cost and 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 decentralised 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 authorisation, or providing resolution of decentralised identifiers (DIDs) [Ree2020]. Utilising multiple ledgers that are interconnected through interledger functionality, instead of a single DLT, provides the flexibility to exploit the aforementioned trade-offs. Finally, providing interledger mechanisms to interconnect different DLTs allows companies and consortiums to select private/permissioned distributed ledgers based on their requirements and constraints. Hence, interledger mechanisms can enhance interoperability across different IoT platforms that utilise different distributed ledger technologies.

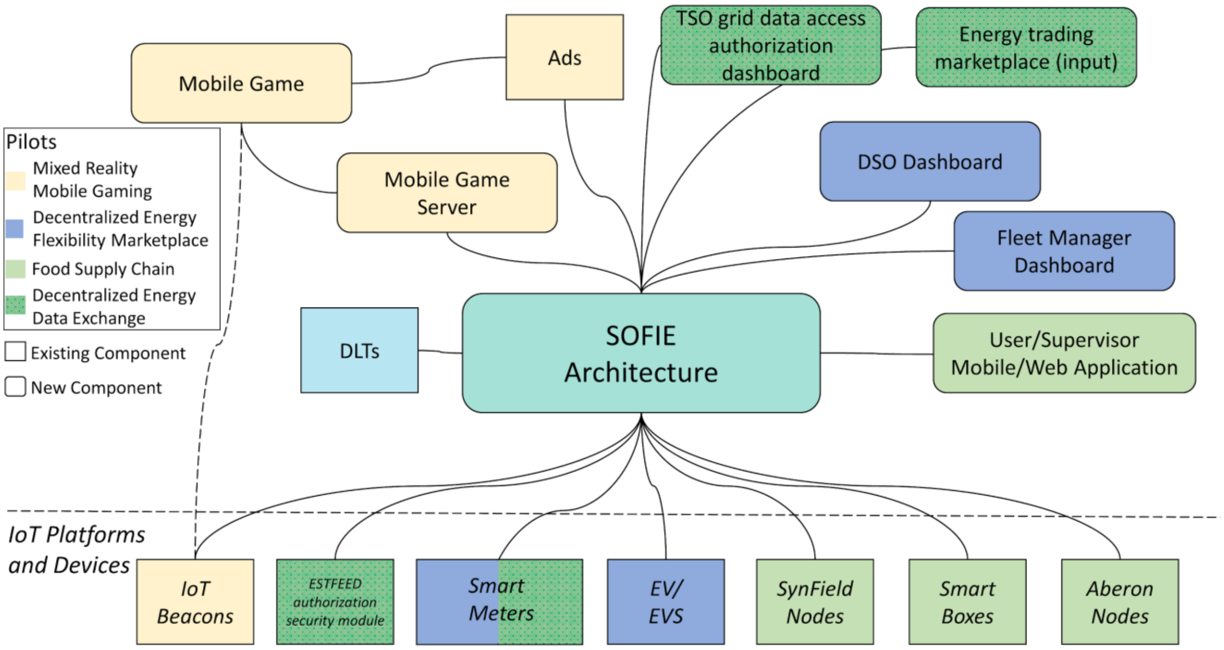
The other SOFIE Framework components are: *Identity, Authentication, and Authorisation (IAA)*, which provides identity management and supports multiple authentication and authorisation techniques; *Privacy and Data Sovereignty (PDS)*, which provides mechanisms that enable data sharing in a controlled and privacy preserving way and privacy preserving surveys using differential privacy techniques; *Semantic Representation (SR)*, which provides tools for describing services, devices, and data in an interoperable way; *Marketplace (MP)*, which allows participants to trade resources by placing bids and offers in a secure, auditable, and decentralised way; and *Provisioning and Discovery (P&D)*, which provides functionality for the management of existing and the discovery of new IoT devices.

All the components can expose *application APIs*, which provide interfaces for IoT clients and applications to interact with the SOFIE components. In Figure 2, the multiledger operations are positioned next to the Interledger component as it is mostly using that functionality, but any of the other components can also utilise multiledger operation when required. Also, the framework adapters and IoT applications can directly interact with the DLTs, but for simplicity this is not shown in the figure. The figure also does not show the interactions between the components, which include the PDS component managing access policies, which the IAA component then enforces, the P&D component relying on SR in discovering new compatible IoT devices, or the MP and IL co-operating in enabling trade, where payments are implemented on one ledger and the traded items reside on another ledger. These interactions will be discussed in more detail in D2.7

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

3 SOFIE Pilots

This section describes how the SOFIE Architecture is utilised in the four SOFIE pilots and their respective system architectures. Figure 3 depicts how pilot components are connected to the SOFIE Architecture. At the bottom of the figure are IoT devices and platforms, while other components used by the pilots are in the upper part of the figure.



Mobile Game
Ads
TSO grid data access authorization dashboard
Energy trading marketplace (input)
Pilots
Mixed Reality Mobile Gaming
Decentralized Energy Flexibility Marketplace
Food Supply Chain
Decentralized Energy Data Exchange
Existing Component
New Component
Mobile Game Server
DSO Dashboard
Fleet Manager Dashboard
DLTs
SOFIE Architecture
User/Supervisor Mobile/Web Application
IoT Platforms and Devices
IoT Beacons
ESTFEED authorization security module
Smart Meters
EV/ EVS
SynField Nodes
Smart Boxes
Aberon Nodes

Figure 3. SOFIE Architecture and Pilots

The rest of this section details the architectures of each of the four SOFIE pilots. More detailed description about the pilots themselves can be found in 'SOFIE Deliverable 5.3 - End-to-End Platform Validation' [D5.3].

3.1 Food Supply Chain Pilot

An overview of the Food Supply Chain (FSC) pilot's final architecture in relation to the SOFIE components is shown in Figure 4. This architecture offers two main applications (bundled in a web application), 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 product quality degradation events. Both of these services, as well as other simple services, are provided to the pilot actors through an FSC web application.

The SOFIE components utilised are:

- Federation Adapters (FA) (one for each IoT platform)
- Identity, Authentication, Authorisation (IAA)
- Privacy and Data Sovereignty (PDS)
- Semantic Representation (SR)
- Interledger (IL)

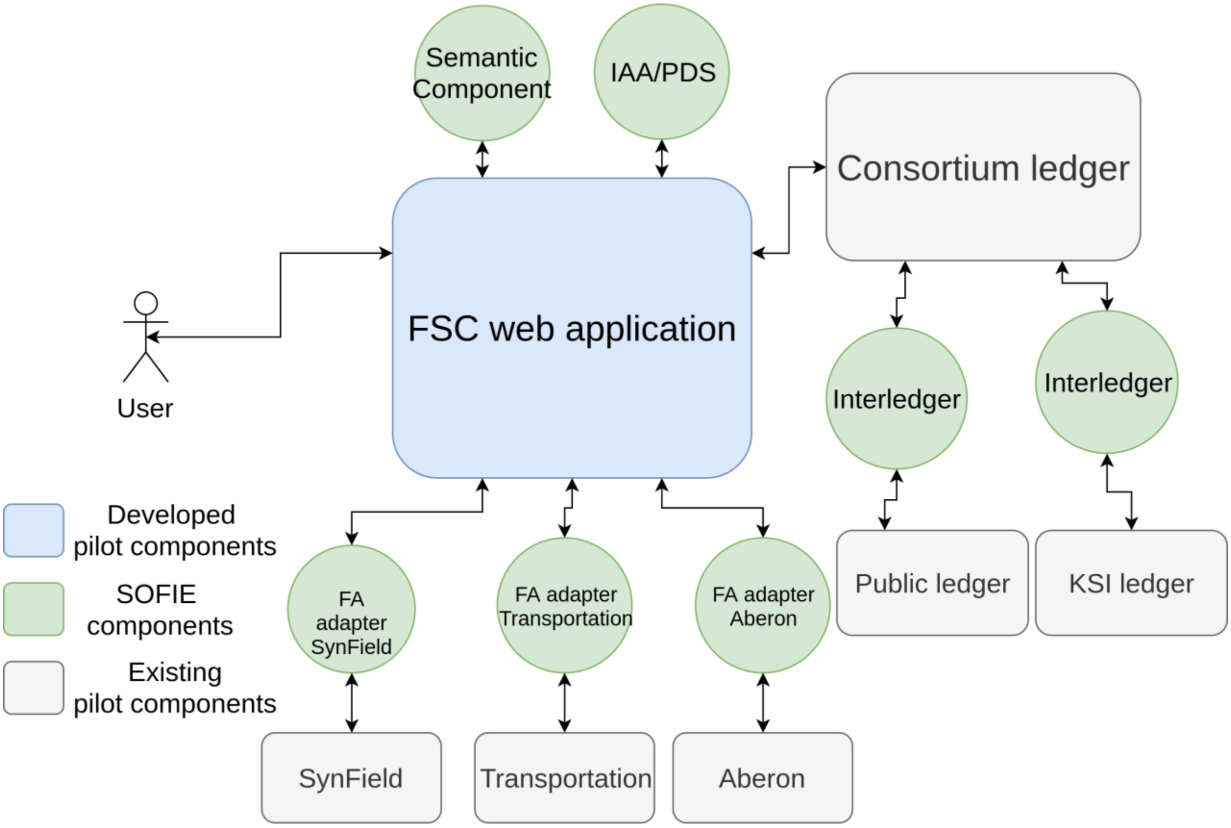


Figure 4. Food Supply Chain pilot's final architecture and SOFIE components' relation

As shown at the bottom of the Figure 4, three IoT platforms are federated:

1. the SynField IoT platform that collects measurements about the growing conditions in the field,
2. a Transportation IoT platform that collects measurements about the produce as they are transferred from one site to another, and
3. the Aberon IoT platform that is responsible for collecting measurements related to the storage conditions of produce in the warehouse.

A Federation adapter has been developed for each of these IoT platforms and has been applied on top of the northbound API of each IoT environment to adapt the corresponding data and metadata using the SOFIE Semantic Representation component and, also, to support authentication and interledger procedures.

As shown in Figure 4, 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, a public ledger which is used to store the hashes of the (meta)data, and the KSI (Keyless Signature Infrastructure) blockchain which is used to create "summary" hashes of the data. In addition to the SOFIE components, the pilot architecture also introduces an additional web application component for: 1) orchestrating the data flow and handling data and metadata management, 2) exposing a public API for the requests the actors receive through the SOFIE FSC web application, and 3) supervising the status of each asset on the provenance business platform (the boxes that carry produce over the whole food chain are considered assets) and scheduling the proper execution of the services provided by the SOFIE components.

3.2 Decentralised Energy Flexibility Marketplace Pilot

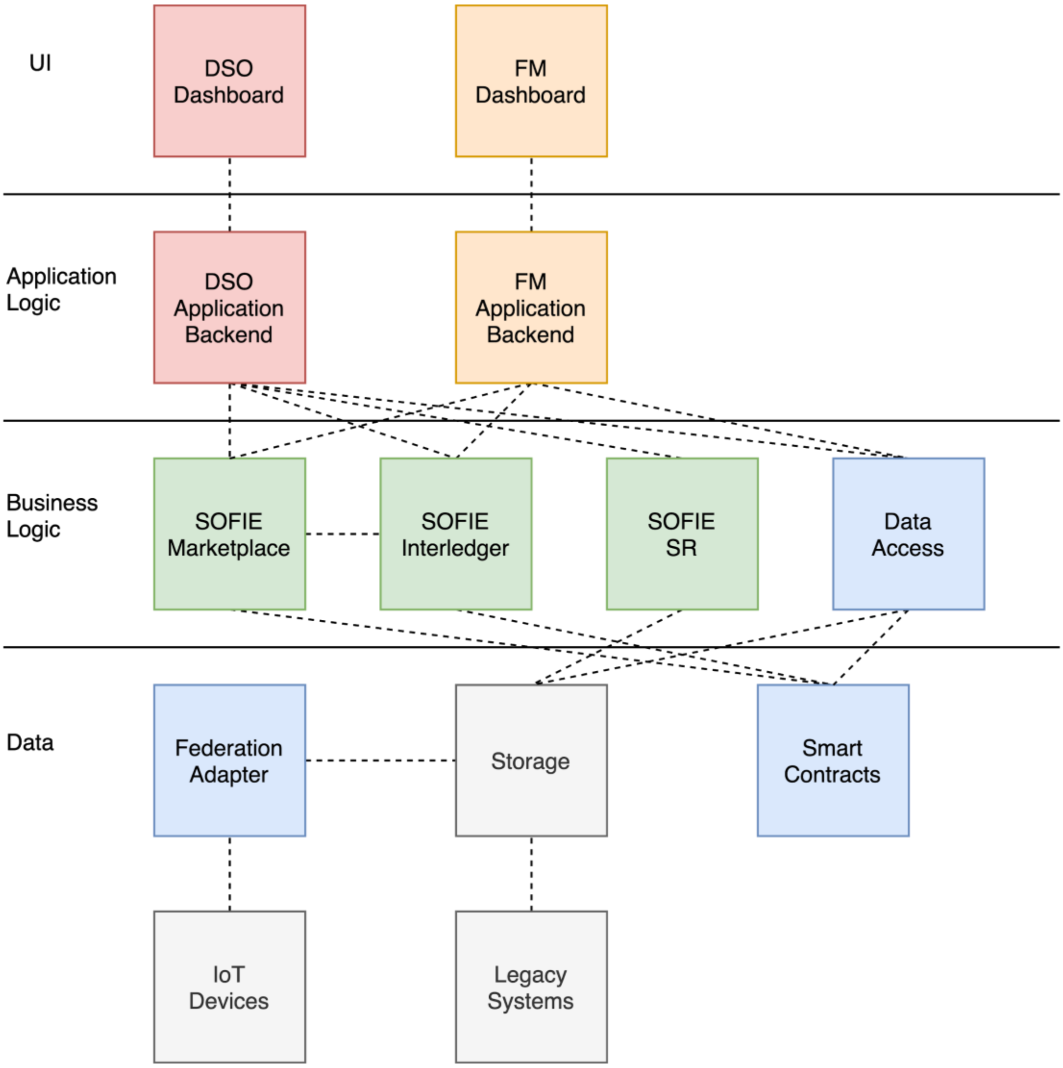


Figure 5. The Architecture of the Decentralised Energy Flexibility Marketplace Pilot

The Decentralised Energy Flexibility Marketplace (DEFM) pilot, as shown in Figure 5, integrates the IoT subsystems and software modules used in the pilot site to control the smart meters, the electric vehicles, and the electric vehicle supply equipment with three SOFIE components: Marketplace (MP), Interledger (IL), and Semantic Representation (SR). In addition to the SOFIE components, some pilot-specific components like the Federation Adapter, the Smart Contracts, and the Data Access layer are utilised to integrate the pilot architecture with the existing infrastructure. The pilot is then completed by the applications used by the two main actors, the Fleet Manager (FM) and the DSO operators. Each application has its own dashboard and backend which provide the application-specific logic.

The Marketplace component can be considered to be the pilot's core component. It provides all the required functionalities to operate a decentralised marketplace built on top of the Ethereum blockchain. The basic functionalities (e.g. creating a new marketplace request or participating in an ongoing marketplace request with an offer) were abstracted in an extensible interface that can be adapted to different use cases. For the DEFM pilot, an extended smart contract was developed, specifying the offer-selection process (*lowest bid* auction) and the attributes characterising a marketplace transaction (network zone, delivery time window, quantity). The MP component also includes the tools needed for compiling, testing, and deploying the derived smart contract. The use case is complemented by the Semantic Representation and the Interledger components. The SR component validates the data feed from the smart meters before creating a new market request. In this way, the operator knows that data gathering is progressing as expected and the market request will be based on valid data. On the other hand, the Interledger component is used to add transparency to the key events during and after the auction: the selection of the winning offer and the payment to the winning fleet manager after the delivery time were modeled as *trigger* events and the component propagates both to a public ledger every time a new event is emitted.

3.3 Mixed Reality Mobile Gaming Pilot

The focus of the mobile game pilot shown in Figure 6 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.

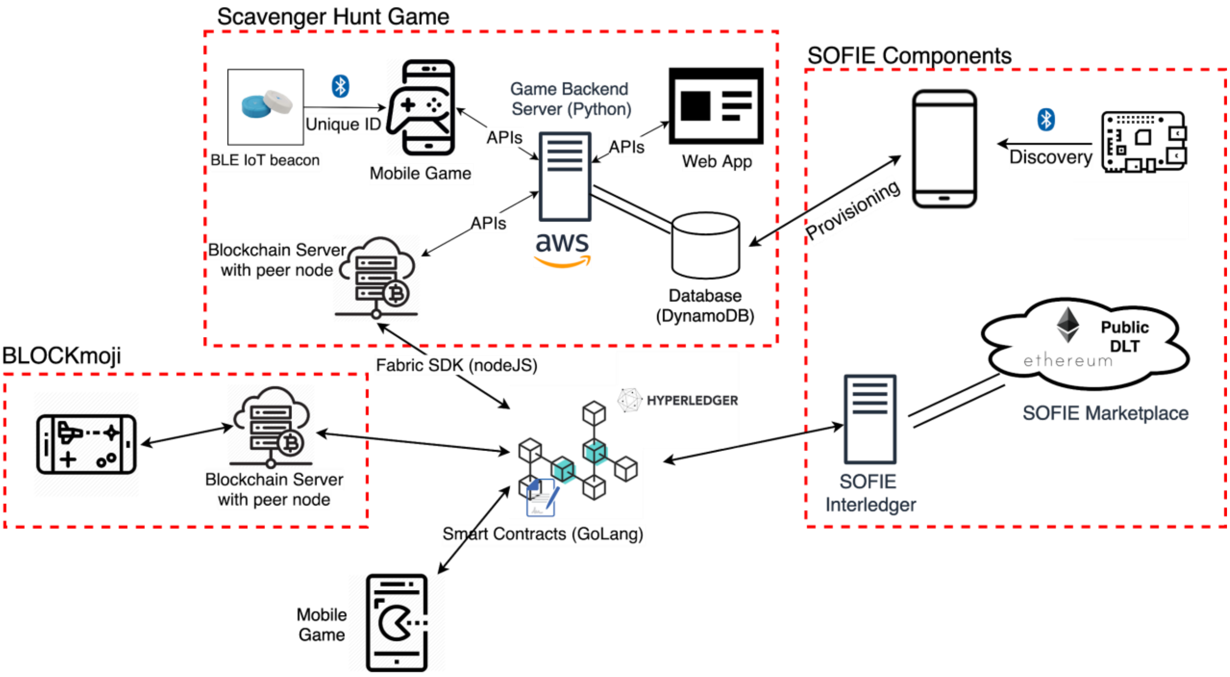


Figure 6. Mobile Gaming Architecture with SOFIE components

A prototype has been developed to understand the use of DLTs for content ownership by players, enabling them to collect and trade in-game content with other players (e.g. characters, weapons, equipment, parts). DLTs provide player ownership of the asset, transparency, and consistency of asset attributes and transactions.

The second use-case studied is a context-aware scavenger hunt game prototype using IoT beacons and an ecosystem backed by a DLT. In the prototype, the player needs to solve riddles using clues to reveal the location of the next IoT beacons. These beacons are used to provide the proximity location of the players when they visit Point of Interests (PoI). At each correct PoI, the player answers a question by observing their physical surroundings. If they answer the question correctly, the player receives the riddle for the next location. After visiting all locations in a hunt, the player is rewarded with in-game coins and item rewards, which are stored on a DLT.

The gaming pilot also leverages SOFIE components to provide new gaming features and enhance player experience as described in Table 5.

Table 5. SOFIE components used and their benefits for Mixed Reality Mobile Gaming Pilot

SOFIE Component	Usage	Benefit
Interledger	● Communication between the permissioned Hyperledger Fabric and public Ethereum networks. ● Linking the closed ecosystem of games and game developers to the public ecosystem of trading games and other virtual assets. ● Achieved by calling special smart contracts and emitting and receiving events	● Interoperability between different distributed ledgers. ● New business opportunities as closed platforms can be connected. ● Benefits of specific ledgers: Fabric for throughput and privacy, and Ethereum for security and traceability.
Provisioning and Discovery	● Scans nearby IoT devices, learns their capabilities, and determines which devices are suitable for provisioning. ● The component configures the IoT devices to work as BLE beacons for the Scavenger Hunt game prototype. ● Provisioned devices can be used in the creation of new Hunts in the game. ● Permissions to configure devices can be managed through DLTs	● Take advantage of the already deployed Web of Things (WoT). ● New business opportunities for device holders, namely micropayments for services used. ● Can be trusted and automated through DLT.
Semantic Representation	● Semantic description of devices using WoT-Things Description. ● Used along with the Discovery and Provisioning component. ● Encoded in a JSON format that also allows JSON-LD processing	● Devices can describe their own capabilities. ● Service interoperability. ● Unify different WoT devices used as beacons.
Marketplace	● Players can place bids for trading in-game virtual assets. Enables the actual trade of resources in an automated, trusted, and decentralised way. ● Once digital assets have been stored on the ledger, the ownership and the item itself cannot be altered. ● DLTs also help maintain the scarcity of a virtual item in a secure and verified way.	● In-app assets have real world value, 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.

3.4 Decentralised Energy Data Exchange Pilot

The Decentralised Energy Data Exchange (EDEX) platform depicted in Figure 7 connects energy data providers with energy data consumers in a secure, open and decentralised way. Both the data providers and the data consumers connect to the platform through their own instance of the same Federation Adapter (FA). 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 and verifiable credentials to establish trust.

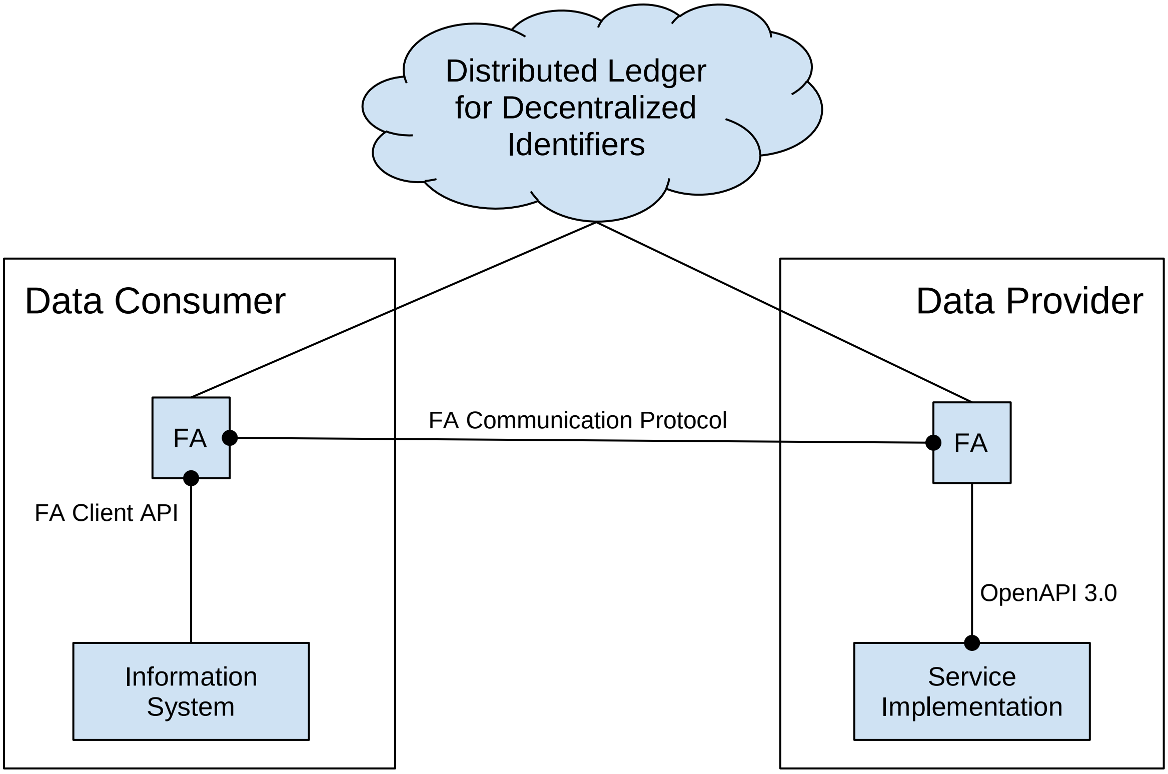


Figure 7. Overview of the EDEX platform

The FA is the key component in enabling the EDEX platform. Its main function is to ensure interoperability and to secure the communication with other entities on the EDEX 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 to use these services. The only requirement for a data provider is to describe its services in OpenAPI 3.0 format.

Each entity on the EDEX platform is identified by a Decentralised Identifier (DID), which form the base layer for Verifiable Credentials (VC) that are used to make authorisation decisions in the FA. Every DID is associated with a public key and the mapping is published on a Hyperledger Indy instance - a distributed ledger built for this purpose. The private key that gives control over the DID is stored in a wallet managed by the FA. This makes it possible for the FA to sign every message sent out from the FA. Use of DIDs and VCs makes the EDEX platform not dependent on DNS names and traditional certificate authorities. Although there are many different methods to define DIDs and basic operations to manage them[1], the EDEX platform currently supports the Sovrin method implemented by Hyperledger Indy.

The FA is also a natural place to construct the audit log of the messages exchanged between entities. A data provider will have a log of signed request messages that can be used to prove which data consumer has asked for which data. Likewise, a data consumer will have a log of signed response messages to prove which data provider gave out which data. The FA will take care of securing this audit log with KSI.

[1] https://w3c-ccg.github.io/did-method-registry/

4 Architecture Components

The SOFIE Architecture consists of the following 6 components: Interledger (IL), Identity, Authentication, Authorisation (IAA), Privacy and Data Sovereignty (PDS), Semantic Representation (SR), Marketplace (MP), and Provisioning and Discovery (P&D).

The SOFIE Framework [Framework] provides example implementations of all the components and the component descriptions in this section often refer to the Framework implementation when discussing implementation details.

4.1 Interledger

The main purpose of the SOFIE Interledger (IL) component is to enable transactions between actors belonging to different (isolated) IoT platforms or silos. SOFIE assumes that each IoT silo either utilises or is connected to one or more DLTs. The Interledger component then enables interaction between these DLTs.

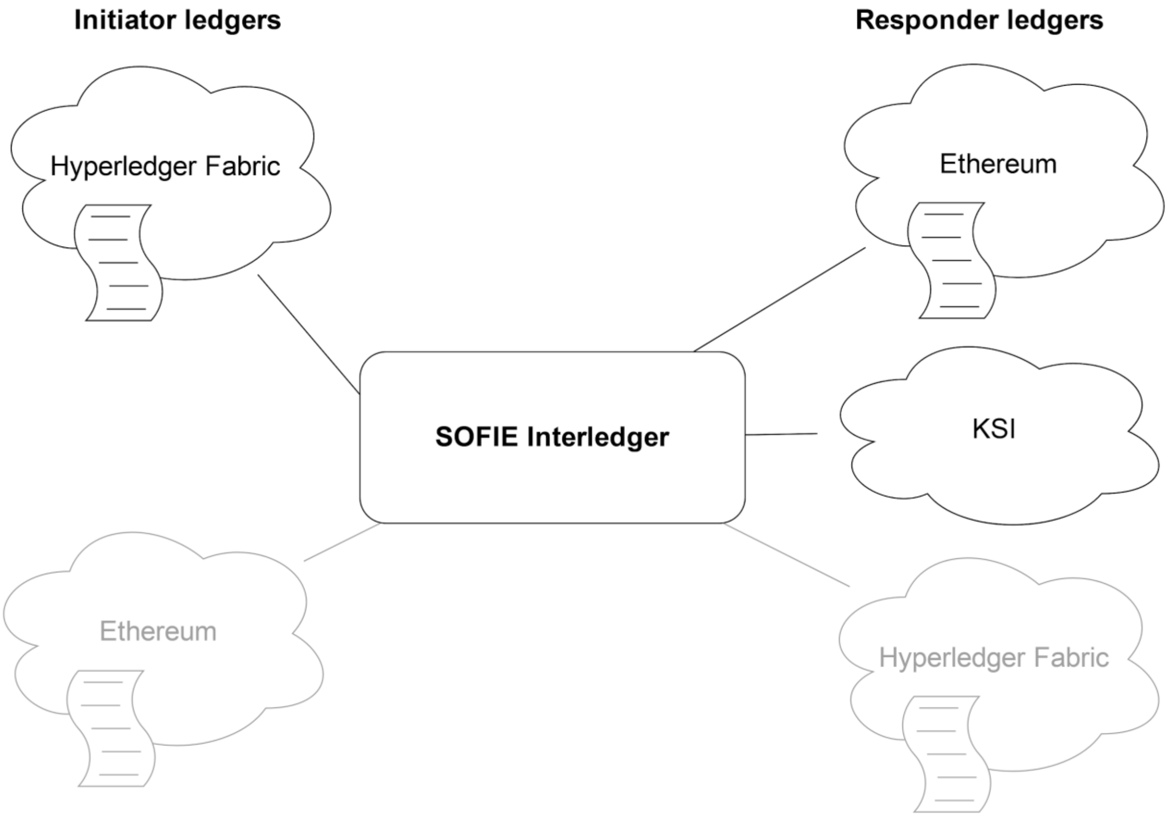


Figure 8. A high-level overview of Interledger component

Using different DLTs is often necessary because of the advantages and disadvantages each of them has. For instance, the public Ethereum ledger offers a high level of immutability, and is very suitable for handling payments and automating tasks via smart contracts when specific conditions are triggered, such as a payment. Nevertheless, Ethereum has a high cost and 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 is permissioned and uses a Byzantine Fault Tolerant consensus mechanism, which makes transactions execute almost immediately, but it does not provide the same level of immutability and trust as e.g. Ethereum does. Using multiple ledgers is also often desirable from the privacy point of view: personal data should not be stored in public ledgers.

Typically, the Interledger component provides support for different types of ledgers such as Ethereum, KSI, Hyperledger Fabric, and Hyperledger Indy ledgers as shown in Figure 8. Generally speaking, it should enable two types of functionalities across ledgers:

- Transferring data from one ledger to another.
- Transferring value from one ledger to another.

In these transactions, one ledger acts as the Initiator, which sends the data/value and the other(s) act as the Responder(s) that receive the data/value. These transactions are, therefore, unidirectional, but if bidirectional connections are required, this can be achieved with two unidirectional connections in opposite directions.

Depending on the use cases and IoT platforms, IL can support e.g. the following features (the SOFIE Framework’s implementation of the Interledger component supports all these use cases):

- Transferring data from one ledger to one or more ledgers so that all or none of the recipients receive the data.
- Transferring data from one ledger to multiple ledgers so that at least the defined minimum number of recipients (k-out-of-N) receive the data.
- Storing hashes of data to a (public) ledger based on the data stored in a (private) ledger.
- Atomic transfer of asset state between two ledgers. The asset can exist in both ledgers, but remains active in only one of them, and its state is simultaneously changed in both or none within a transaction.
- When using Hash Time-Locked Contracts (HTLCs) [Sir2019], the Interledger component can be used to automate the asset exchange between two ledgers using HTLCs.
- Strengthening the security of the ledger providing DIDs by periodically auditing the hash fingerprint that is recorded in a public ledger.

The interledger functionality can be implemented either by a stand-alone node run by a single trusted (third) party, or in a decentralised manner by multiple nodes run by a consortium of (third) parties as shown in Figure 9. The implementation manner depends on the trust model within a certain application. If there is a trusted or neutral party, the Interledger node can be implemented and run in a centralised-controlled way. On the other hand, the decentralised mode can significantly reduce the trust required compared to a single node, while at the same time it can provide better scalability in terms of performance and improved resiliency.

The Interledger component can cooperate with other SOFIE components, providing the possibility to interact with multiple distributed ledgers at the same time. For instance, the Mixed Reality Mobile Gaming Pilot utilises the Interledger and Marketplace components for trading the gaming assets: the Interleger component makes sure that the active state of a gaming asset is switched between the gaming consortium ledger and the public trading ledger in an atomic manner.

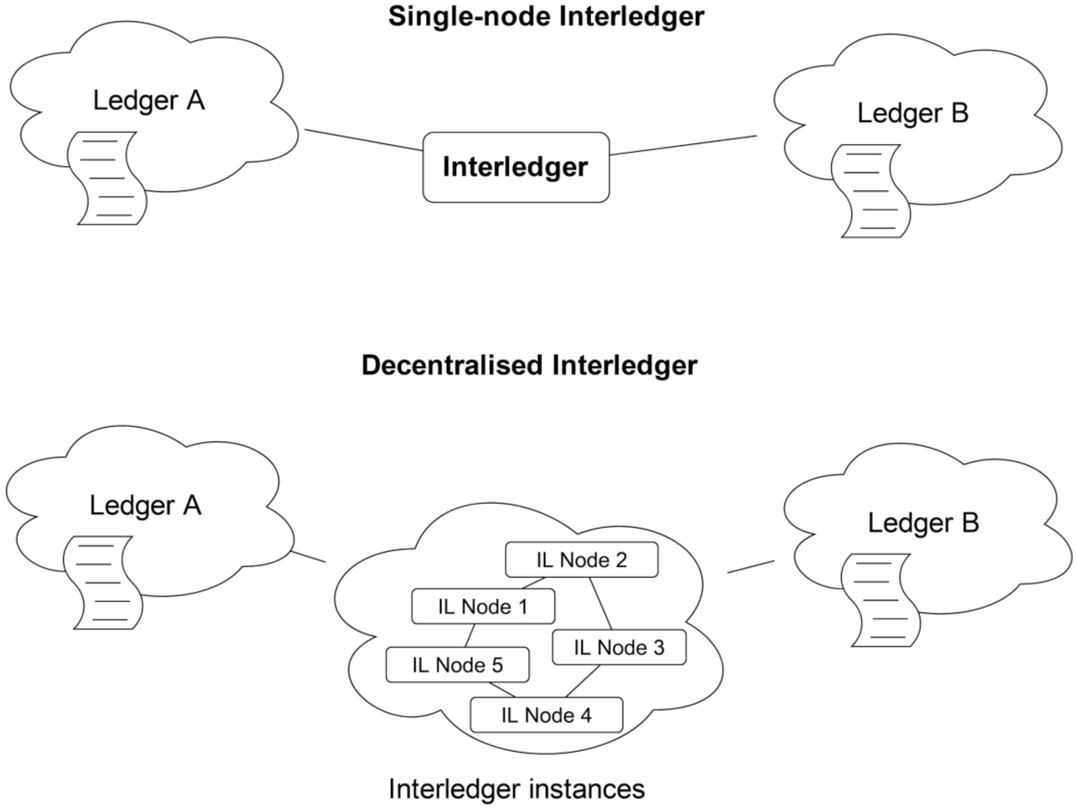


Figure 9. Interledger - a stand-alone node or a consortium of nodes

4.2 Identity, Authentication, Authorisation

The Identity, Authentication, and Authorisation (IAA) component enables access control for shared resources. The component operates as a *firewall* that stands between end-users and resources, and filters out unauthorised requests. As illustrated by Figure 10, IAA can either authenticate and authorise users by itself, or it can just act as a *Policy Enforcement Point (PEP)* and enforce access control decisions that have been taken by another entity (e.g., by the Privacy and Data sovereignty component described below). Compared to existing approaches, IAA has many benefits. IAA builds on emerging standards for user identification that support decentralisation, self-sovereignty, and privacy preservation. Furthermore, by leveraging blockchain technology and Ethereum smart contracts, IAA supports revocation and immutable logging.

When IAA performs user authentication and authorisation, it relies on a number of identification mechanisms and pre-configured access control policies. Currently, the IAA component in the SOFIE Framework supports user identification based on Hyperledger Indy Decentralised Identifiers and Verifiable Credentials,[2] as well as based on W3C's Verifiable Credentials.[3]

When IAA acts as a PEP, it expects *access tokens* that include authorisation decisions. These tokens are validated by the component using pre-configured token verification rules. Currently, the Framework IAA component supports JSON web tokens,[4] as well as blockchain-based

[2] https://www.hyperledger.org/use/hyperledger-indy

[3] https://www.w3.org/TR/vc-data-model/

[4] https://tools.ietf.org/html/rfc7519

tokens stored in an Ethereum smart contract using ERC-721[5] and the solution described in [Fot2020]. Additionally, it supports two modes for associating an access token with the corresponding user: the *bearer* mode, in which any users that has a token is considered its legitimate owner, and the *proof-of-possession* mode, in which the access token includes a public key and the user has to prove that s/he is the owner of the corresponding private key.

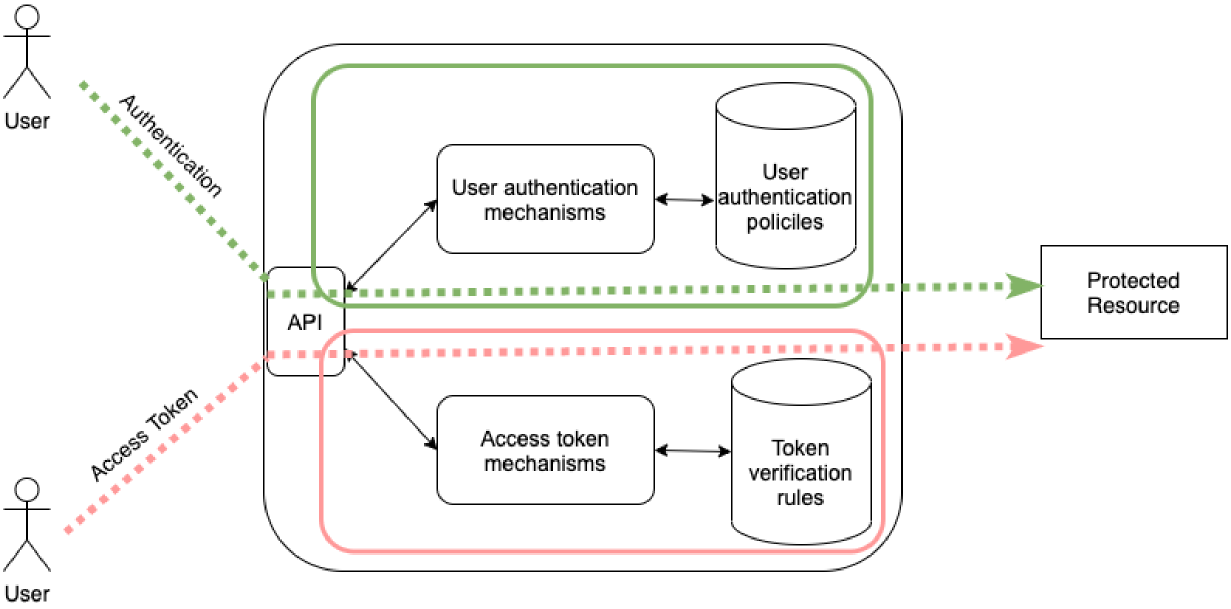


Figure 10. IAA can perform both user authentication as well as user authorisation based on access tokens.

IAA implementations should allow flexible access control policies and token verification rules, so that users may take full advantage of the supported mechanisms. For instance, SOFIE Framework's implementation of IAA supports JSON Path,[6] a feature-rich query language that can be used for validating JSON-encoded objects. Furthermore, all IAA implementations are encouraged to be as transparent to the protected resources as possible, this way facilitating their adoption even in use cases that include constrained resources. As an example, SOFIE Framework's IAA implementation acts as an HTTP forward proxy that intercepts all requests between end-users and the protected (HTTP) resources. Using this approach, the implementation can be used to protect any HTTP-based service.

4.3 Privacy and Data sovereignty

The Privacy and Data Sovereignty (PDS) component enables access control delegation, as well as privacy preserving data sharing.

Access control delegation is enabled by allowing the use of *Authorisation Servers (AS)* that act as *Policy Decision Points* (PDPs) as shown in Figure 11. An AS receives from users *authorisation grants* and outputs *access tokens*. The AS functionality provided by the PDS component in the SOFIE Framework supports the following types of authorisation grants: Hyperledger Indy Decentralised Identifiers and Verifiable Credentials, W3C's Verifiable Credentials, and pre-shared secret keys. These grants are validated based on pre-configured policies. Similarly, a PDS-enabled AS can generate JSON web tokens, as well as blockchain-based tokens stored in an Ethereum smart contract using ERC-721 and the solution described

[5] https://eips.ethereum.org/EIPS/eip-721

[6] https://goessner.net/articles/JsonPath/

in [Fot2020]. The generated access token can be sent directly to the user and/or stored (encrypted) in an Ethereum-based smart contract. In the latter case, further token exchange mechanisms can be considered, e.g. smart contract based *fair-exchange* of tokens with digital currency. Access tokens can then be consumed by service end-points that provide user authorisation (e.g. by the IAA component described in the previous section).

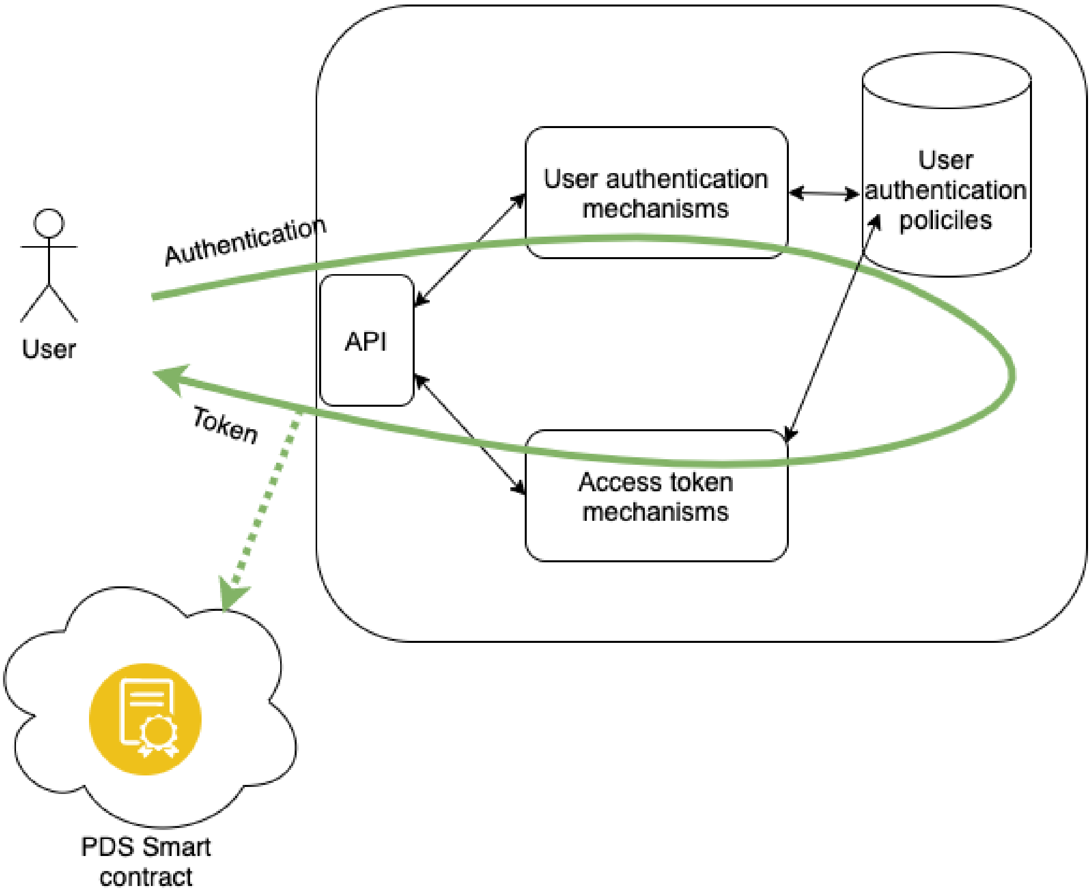


Figure 11. PDS acting as a Policy Decision Point. It receives authorisation grants from users and outputs access tokens.

Privacy-preserving data sharing can be enabled by using *local differential privacy* mechanisms, as in Figure 12. As an example, the PDS component in the SOFIE Framework provides modules that enable *data obfuscation* and the extraction of statistics out of the obfuscated data. Local differential privacy has been selected since it allows users to preserve their privacy without relying on a third party to add *noise* to the aggregated data. Furthermore, local differential privacy allows collected data to be stored in public repositories or even in smart contracts. Indeed, the PDS component contains a smart contract where (obfuscated) data is stored. From there, any third party can extract a number of statistics. PDS also supports encryption of data, so that only authorised third parties can extract statistics (e.g., users that have paid a predefined amount of digital currency).

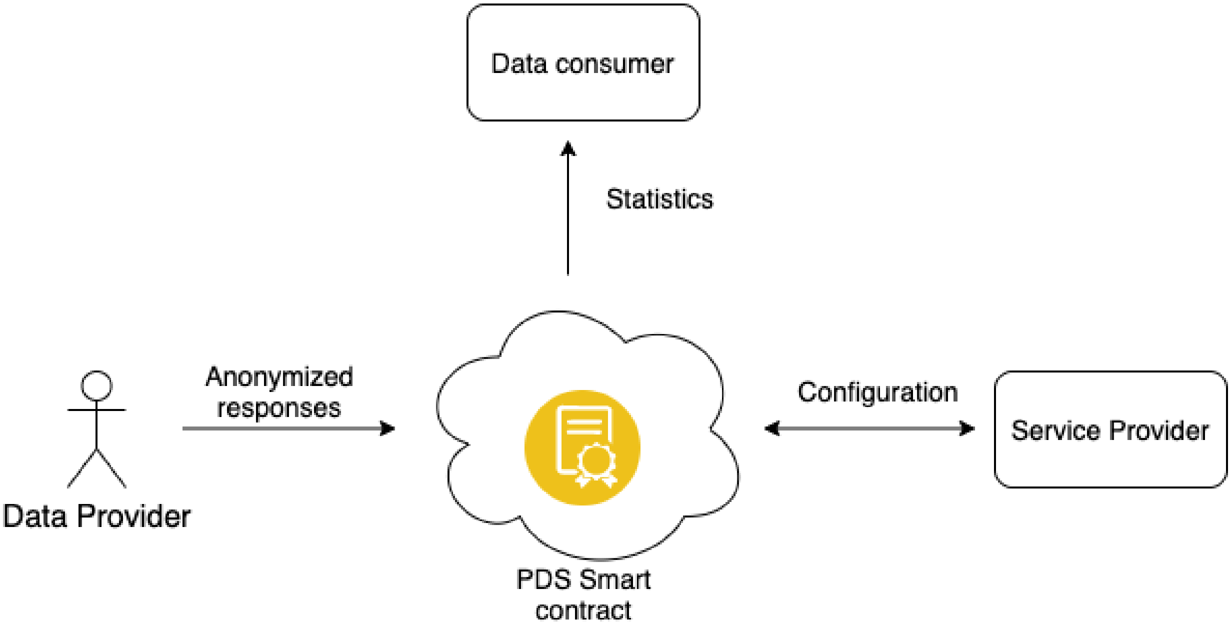


Figure 12. PDS Privacy modules. Data providers provide anonymised responses using local differential privacy. Data consumers extract meaningful statistics.

PDS implementations should follow standards-based approaches for implementing the AS functionality. The SOFIE Framework's implementation of PDS follows the OAuth2.0 standard and it treats the provided authorisation grants similarly to how OAuth2.0 handles the *client credentials grant*.[7] Similarly, PDS implementations should be flexible on how an AS should verify the provided grants. SOFIE's PDS implementation allows the definition of rules using JSON path. When it comes to privacy-preserving data sharing, PDS implementations should consider lightweight local differential privacy mechanisms so that also constrained devices can be used. SOFIE's PDS implementation uses the *RAPPOR* local differential mechanism [Erl2014] that enables end-users privacy using only lightweight operations. Furthermore, PDS implementations should not be tailored for specific types of data or aggregated statistics, instead they should be generic and allow a wide range of applications. For example, SOFIE's implementation of PDS allows the creation of *surveys* with an arbitrary number of possible responses.

4.4 Semantic Representation

Exchanging data between different IoT platforms and systems can be challenging as the platforms often utilise different formats for their data representation and definition, and one system may not know which data model the other system can handle. The SOFIE Semantic Representation (SR) component mitigates this problem and restores semantic interoperability between systems and IoT platforms by allowing the definition and enforcement of the semantics of the data. The SR component typically offers two functionalities

- Data semantic definition
- Data validation

As shown in Figure 13, systems implementing the SR component enable interoperability by defining a data semantic which can be used by other entities to exchange data. The interoperability is achieved with a *data model*, which can be defined using different data

[7] https://tools.ietf.org/html/rfc6749#section-4.4

representation standards, such as JSON schemas and W3C WoT Thing Description, but proprietary models can also be implemented when required by the application.

This data model is then managed by the SR component, which allows users to define the accepted data models in the system. An example could be the definition of security rules in the schema stored in the SR component: defining the security rules forces third parties' schema implementations to satisfy them if they want to communicate with the system. Another example is the definition of a schema which defines the main properties and interfaces of the IoT devices that can interact with the system.

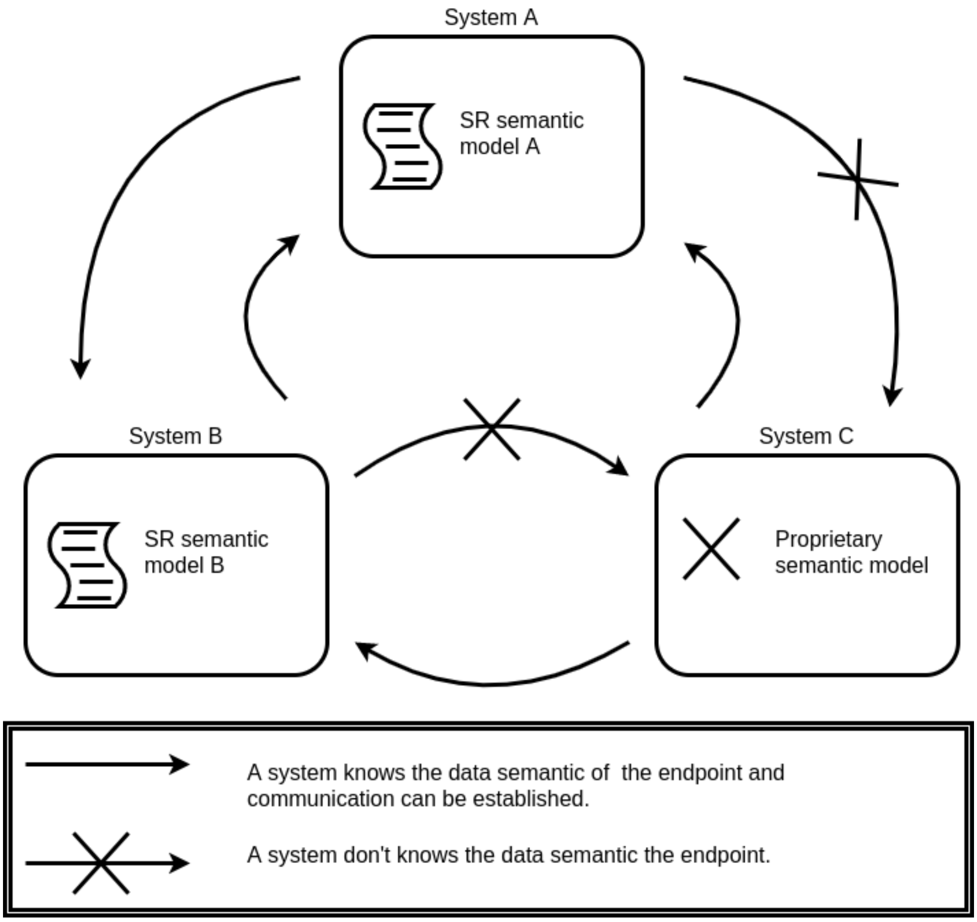
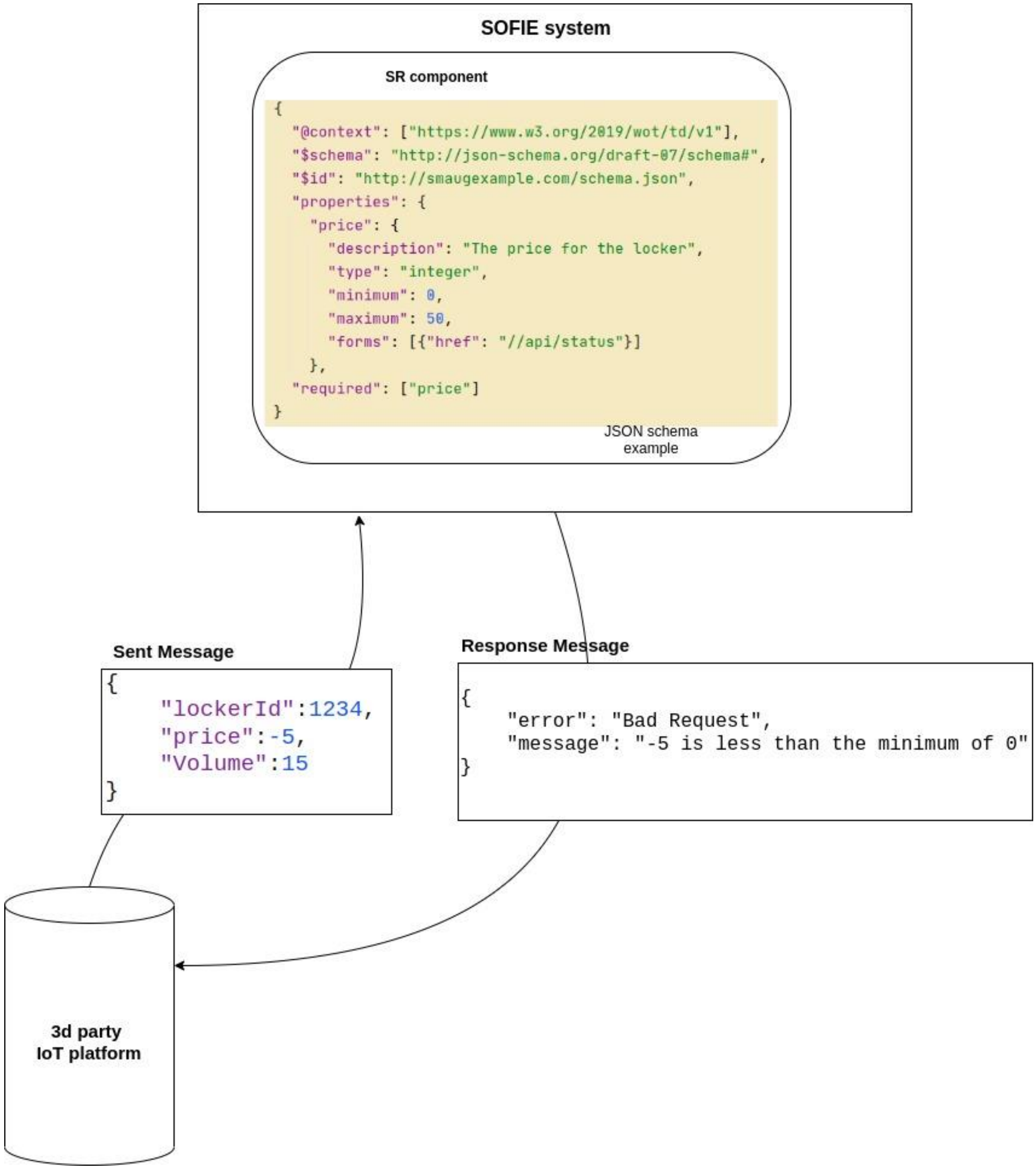

System A
SR semantic model A
System B
SR semantic model B
System C
Proprietary semantic model
A system knows the data semantic of the endpoint and communication can be established.
A system don't knows the data semantic the endpoint.

Figure 13. System interoperability

The second functionality, message validation, shown in Figure 14, assures the quality of the data exchanged between systems. This can be done by validating the messages an external system sends to the SR component. The functionality can be implemented by validating the messages against the schema defined with the definition functionality. An example is JSON messages validation: this functionality acts as a filter for the data exchange to the system, informing the users that sent the data whether the process is successful or what problems occurred. The functionality can be used only if a schema is stored in the semantic representation component. Third party users, such as IoT platforms, can send JSON messages to the system that implements the component, specifying to which schema the message conforms. The SR component then validates the message against the defined schema to check that the data conforms to the schema's rules.



```
{
  "@context": ["https://www.w3.org/2019/wot/td/v1"],
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://smaugexample.com/schema.json",
  "properties": {
    "price": {
      "description": "The price for the locker",
      "type": "integer",
      "minimum": 0,
      "maximum": 50,
      "forms": [{"href": "//api/status"}]
    },
  "required": ["price"]
}
```

```
{
     "lockerId":1234,
     "price":-5,
     "Volume":15
}
```

```
{
    "error": "Bad Request",
    "message": "-5 is less than the minimum of 0"
}
```

Figure 14. Message validation representation

The SR component can be used with other components of the SOFIE Framework to enhance the functionalities of a system. An example is the P&D component that uses the SR component to enforce rules and requirements for the provisioning of devices.

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 vehicles) in an automated, decentralised, and flexible way. In this context, a decentralised 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 marketplace can be partially decentralised, when e.g. a group of independent agriculture producers and retailers are managing it, or fully decentralised where anyone can join and use the marketplace.

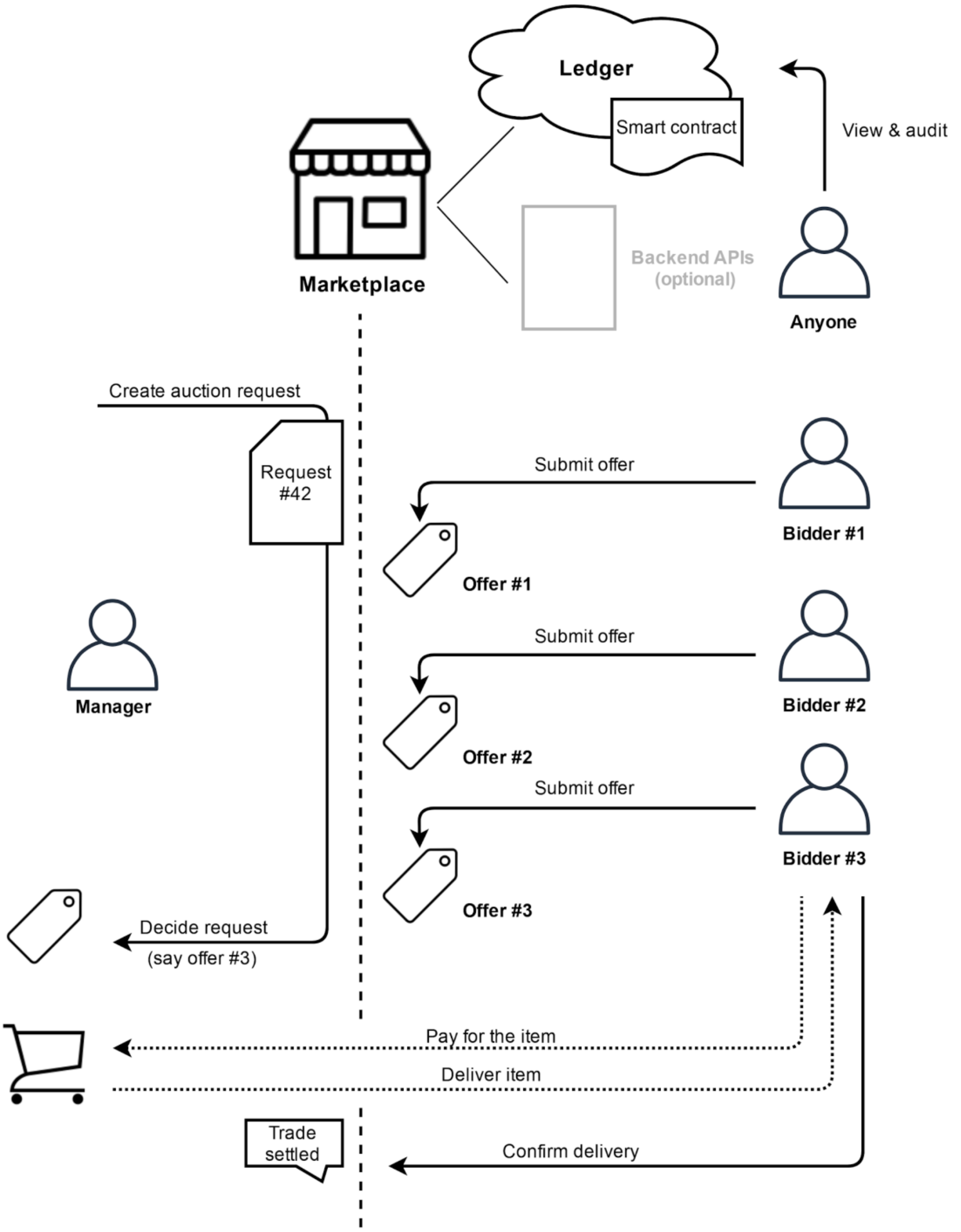


Figure 15. An example flow of the marketplace process.

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 actors (buyers or sellers) on the marketplace must be able to negotiate trades using different pricing models, perform payments, and verify that the trade has been carried out successfully with as little user interaction as possible. The marketplace must also provide auditability to help with potential dispute resolutions. The example flow of the marketplace process is shown in Figure 15.

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 marketplace functionality can be implemented e.g. on top of an Ethereum blockchain utilising smart contracts, though the marketplace may also interact with other kinds of DLTs (the SOFIE Framework provides an implementation built on top of Ethereum). 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 contracts, and affords security, transparency, and auditability.

The decentralised nature of the Marketplace component opens the possibility of trusted and auditable trading of resources among parties who do not necessarily trust each other. Such trading enables the flow of data, value, and resources across originally isolated IoT silos. Many of those IoT silos are connected to different DLTs, which remains an obstacle for trading on the marketplace. This issue can be resolved by combining the Marketplace component together with the SOFIE Interledger (IL) component, which allows the passing of information, value, and state of assets across different ledgers in a secure and atomic manner.

4.6 Provisioning and Discovery

The goal of the Provisioning & Discovery (P&D) component is to manage the IoT resources in the system by provisioning existing the IoT device to a working state with the platform and by enabling the discovery of new IoT resources along with their related metadata, the interaction of which is illustrated in Figure 16. Using this functionality, it is possible to e.g. decentralise the process of making new resources available to systems and to automate the negotiations for the terms of use and the compensation for the use of these resources. This component works together with the SOFIE Semantic Representation component to provide meta-data for the IoT devices.

The SOFIE Framework's P&D component provides the following functionalities

- Provisioning of IoT resources including
  - Configuration of Devices
  - Enrolling new devices to system
- Discovery of the new IoT resources using e.g.
  - Bluetooth Low Energy discovery
  - DNS-Service Discovery
- Licensing of the resources

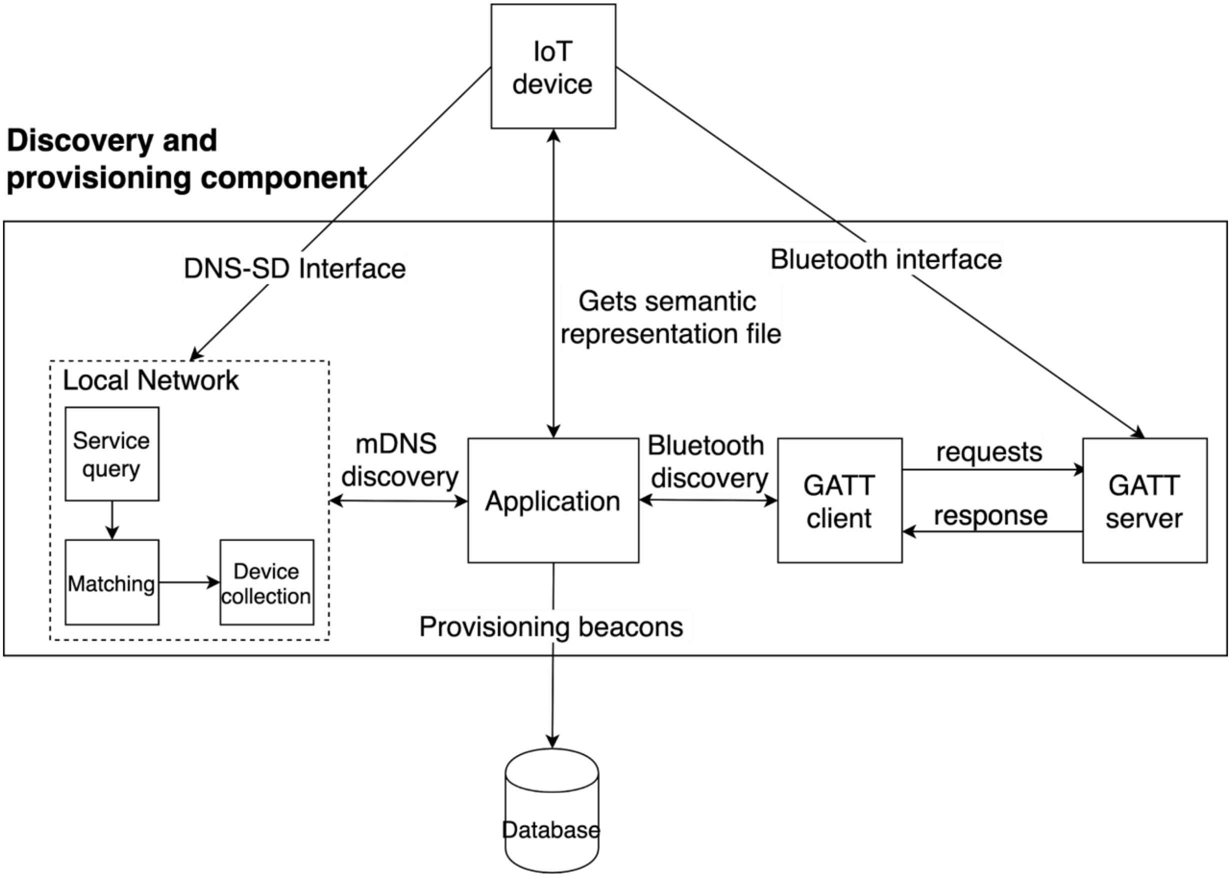


Figure 16. SOFIE Provisioning and discovery Internals.

The first functionality of the component is to provision the devices using the meta-data, which includes enrolling a device into the system, and getting each device configured to provide the required service and to send data to the right place on the network. The first part is establishing the initial connection between the device and the IoT solution by registering the device. In the component, the Provisioning interface goes through the meta-data and checks against the requirement before provisioning the device to the database. This also acts as the filter for either accepting or rejecting the newly discovered IoT resource. After enrolling the device, the interface provides for the configuration related information for the device to bring it to a working state, including defining the desired state of the device.

The second functionality of the component is the discovery of new IoT resources. This component's interface provides operations to perform a scan and discover open IoT devices nearby. It also provides an interface to discover devices published on the local (WLAN, etc.) network. The discovery interface lists newly discovered devices along with their related meta-data before enrolling them in the system.

The final functionality of the component is to license the device to automate the negotiations for the terms of use. The interface calls a smart contract on the blockchain and compensates the owner of the device for the usage of the provisioned devices.

As an example, a location-based game can discover new IoT devices usable for expanding the game world and automatically add them to the resource database if the resources are accompanied with the necessary metadata including the licence for using the device and the terms of compensation. This example was implemented using the following two existing protocols: BLE Discovery and DNS-SD with multicast. A mobile application is used to search for the new IoT devices on the WLAN or using the Bluetooth interface. The meta-data for the beacons is defined by the SOFIE Semantic Representation component. After retrieving the device information, the provisioning interface goes through defined rules and requirements before enrolling the devices. It also connects to the database and sends the IoT device meta-data to the database for provisioning. It also connects to the Ethereum blockchain and calls smart contracts in order to provide compensation for the device usage.

5 Federation Adapters

Two IoT platforms can be federated if they are interoperable, which means they understand each other's interfaces, and thus can exchange data and make sense of it. However good the SOFIE components are, they cannot accommodate the federation of every conceivable IoT platform on their own - a gap will always exist between the capabilities of existing IoT platforms and that which the SOFIE Framework can make use of. To bridge this gap, the SOFIE Architecture has a place for *federation adapters*. It is the role of these adapters to take into account the individual nature of every federated IoT platform, and adapt it to the SOFIE Architecture.

Federation adapters can take on this responsibility at different levels. For example, in the Decentralised Energy Data Exchange (EDEX) pilot, the federation adapters are completely in charge and take care of the secure communication between two IoT platforms, as well as the identity management and the service description format. The federation adapter makes use of the SOFIE components and calls them when needed. There is no central entity that the federation adapter could rely on to run the SOFIE components. This approach is described in more detail in section 5.2 below.

Another example is the federation adapter in the Decentralised Energy Flexibility Marketplace pilot, where it is used at the IoT device level and its sole purpose is to collect data from different IoT devices. The federation adapter knows nothing about the SOFIE components. A technical description of this federation adapter is in the following section.

5.1 Decentralised Energy Flexibility Marketplace

The DEFM pilot utilises a Federation Adapter[8] based on the FIWARE[9] platform. The adapter combines FIWARE components (Orion Context Broker, JSON Iot-Agent, and Comet Short-Term Historic)[10] with third-party components (MongoDB No-SQL database and Mosquitto MQTT broker). The main purpose of the adapter is to manage the lifecycle of the data retrieved from the IoT sources. Within the pilot, it is used to retrieve and store the readings from the IoT smart meters, but it can also be used for different kinds of data in different contexts.

After the initial configuration and deployment, the FA is able to:

- define service groups
- define sensors and actuators
- communicate with the IoT devices via MQTT
- retrieve and aggregate historical data

Figure 17 illustrates the federation adapter’s architecture and software components.

[8] https://github.com/SOFIE-project/efm-federation-adapter

[9] https://www.fiware.org

[10] FIWARE components are discussed in more detail in Sections 6.2 and 6.3

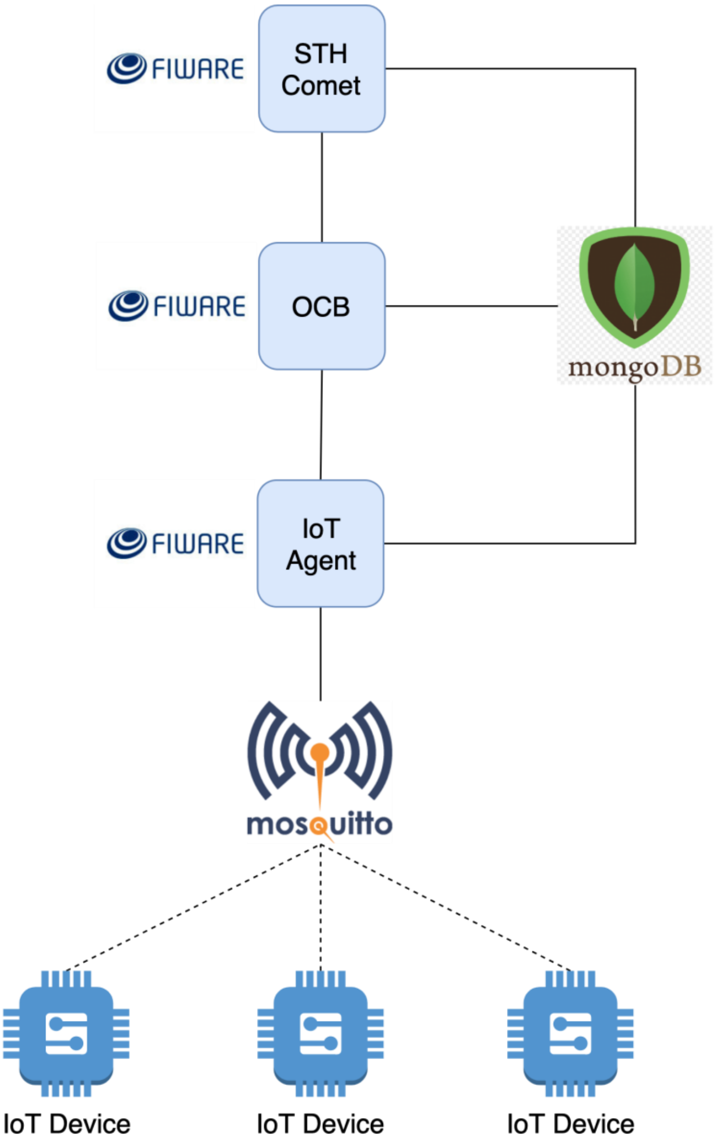


Figure 17. Federation Adapter of the Decentralised Energy Flexibility Marketplace pilot.

5.2 Decentralised Energy Data Exchange

The two main responsibilities of the FA in the EDEX pilot is to proxy messages and to manage the identity of the represented entity. The internal structure of the FA shown in Figure 18 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.

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 request signature. If the request is for a service that requires further authorisation, 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 ssi-agent receives such a request, it will send a proof request to the public interface of the data consumer ssi-agent. Once the data provider 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.

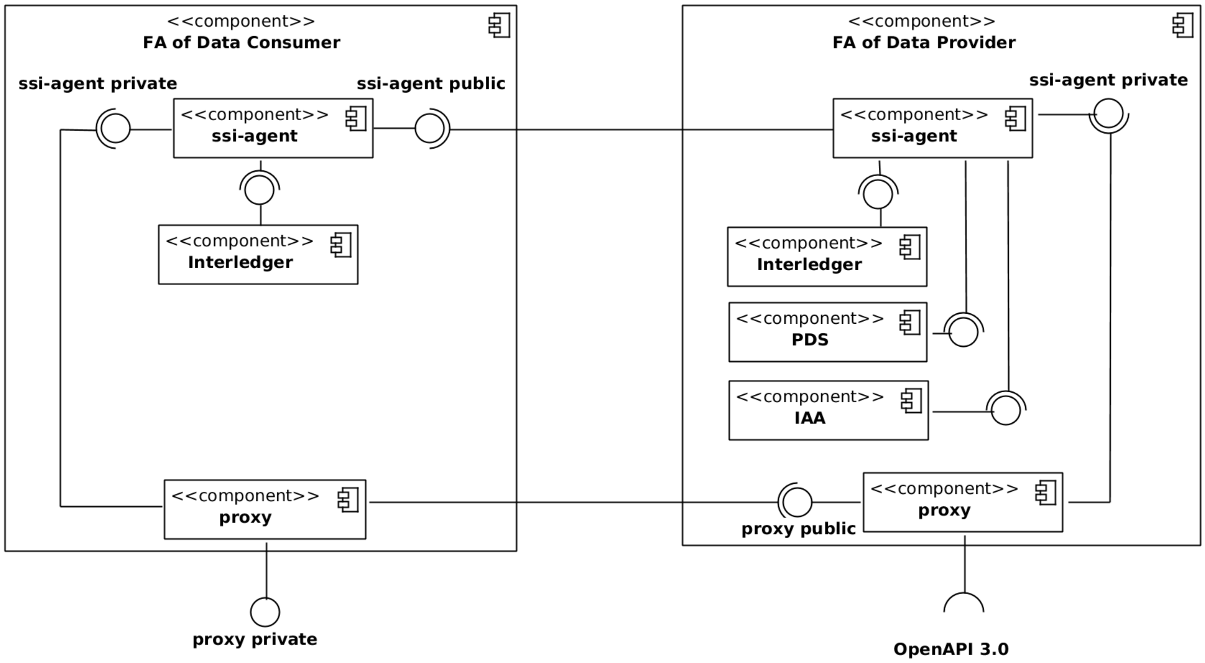


Figure 18. Internal structure of the FA

An alternative for the data consumer to proving its credentials on demand, is to send them together with the request, as a JSON Web Token issued by the SOFIE Privacy and Data Sovereignty (PDS) component. In this setup, it is the responsibility of the data consumer to acquire the token accepted by the data provider and include it in the request header. The data provider could itself be the issuer of such tokens. If the data provider proxy receives a request with a token, it will use the SOFIE Identification, Authentication, and Authorisation (IAA) component to verify the token and get the trusted values for the attributes that are required for the authorisation decision.

The ssi-agent on both sides uses the SOFIE Interledger component to periodically record the state of the Hyperledger Indy instance with KSI.

6 External Components and Interfaces

This section describes the key external components and interfaces used by the SOFIE Architecture implementation in the SOFIE Framework.

6.1 Web of Things (WoT) Thing Description (TD)

Web Of Things (WoT) is a set of standards developed by the W3C consortium to enable interoperability between IoT devices. The SOFIE Architecture focuses on WoT standards defining IoT devices: the WoT Thing Description (TD). The WoT TD allows the metadata definition of a Thing that can be any virtual or physical device that has to interact with other systems as shown in Figure 19.

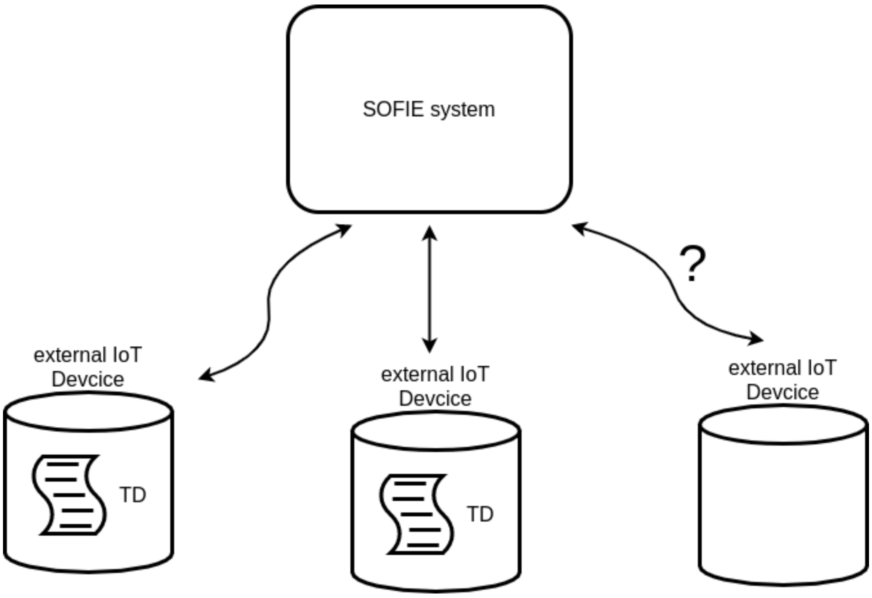


Figure 19. WoT Thing Descriptions enable interoperability.

With the definition of such a standard, the SOFIE system can be connected to any external systems, IoT siloes, and devices that can recognise and process the WoT TD. On the other hand, SOFIE systems can recognise external systems that implement the WoT TD. When an external system does not implement WoT TD, the SOFIE system must do an ad-hoc adaptation of that system, which requires more work.

WoT TD can be integrated in many ways, and in the SOFIE Architecture the Semantic Representation component helps the integration process in different ways. One example is to define the system data model: the SOFIE system can define its properties, interactions, supported actions, data schema, security vocabulary and more. Another functionality enabled by the SR component in conjunction with WoT TD is to validate the external systems, IoT platform, and devices to assure that the external devices are able to communicate with it: the SR component can check that the external system is congruent with WoT TD before enabling any communication.

6.2 FIWARE

FIWARE[11] is an open source initiative, which aims to facilitate the development of platforms for different domains, such as *Smart Cities* and *Smart Energy*, by defining a universal set of standards for context data management.

[11] https://www.fiware.org/

FIWARE offers a set of open source platform components, as illustrated in Figure 20, which can be assembled together to easily build smart solutions. The integration of components, their interoperability and portability are enabled by a powerful API (FIWARE NGSI). The common need for any different *smart platform* is the gathering and management of context information, making it possible to process that information and to react by altering or enriching the current context. FIWARE addresses the need for context information management with its Context Broker component, which enables software platforms to perform updates and access the current context state. The Context Broker is complemented by several optional platform components, covering different stages of the process from supplying context data to data processing, analysis, and visualisation.

FIWARE NGSI enables the portability of different applications across different FIWARE platforms, as well the extensibility of existing platforms by using additional components.

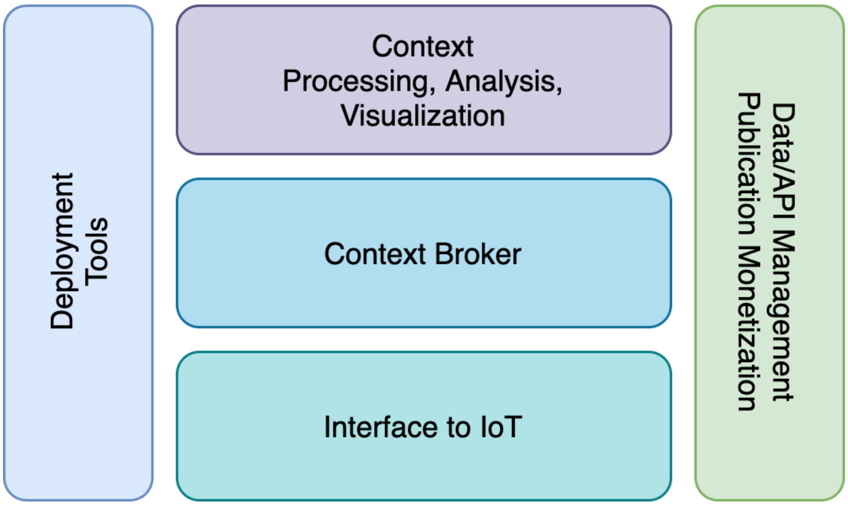

Deployment Tools
Context Processing, Analysis, Visualization
Context Broker
Interface to IoT
Data/API Management Publication Monetization

Figure 20. Architecture of a typical FIWARE platform.

The Federation Adapter used in the DEFM pilot and illustrated in section 5 leverages some of the most popular FIWARE platform components.

The FIWARE *Context Broker* Generic Enabler, in charge of the context information management, is the minimum and mandatory requirement to label a platform as “Powered by FIWARE”[12].

6.3 Integrating additional FIWARE components

The usage of the Context Broker as part of the DEFM adapter enables bi-directional connectivity between FIWARE platforms and the federation adapter:

As shown in Figure 21, the external FIWARE platform components will be able to *publish* context information updates to the Federation Adapter’s Context Broker, for example enabling the federation adapter to consume data gathered from external or new data sources.

[12] https://www.fiware.org/developers/catalogue/

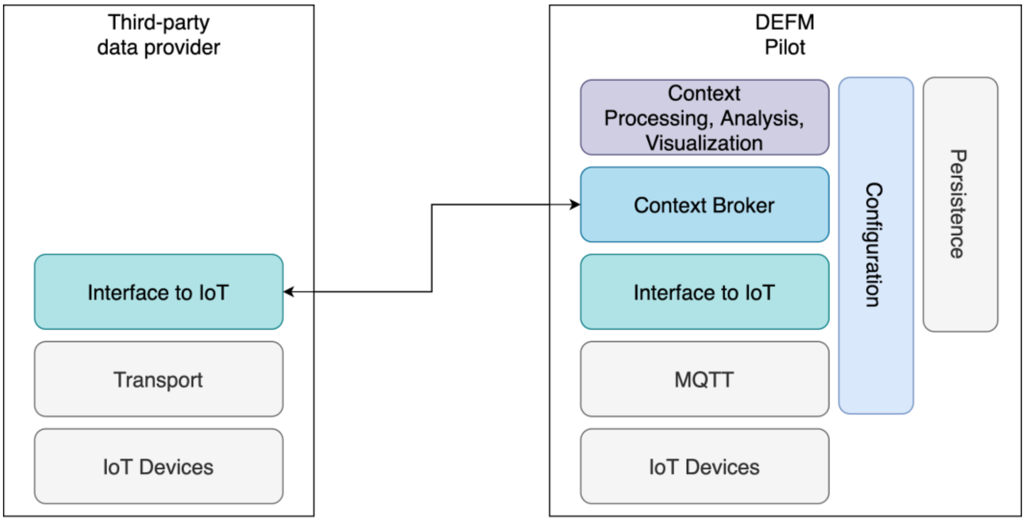


Figure 21. Connection of the DEFM pilot with a third-party data provider.

The external FIWARE platform components will be able to *subscribe* to the Federation Adapter’s Context Broker, as in Figure 22, being able to react to updates and process the data managed by the federation adapter.

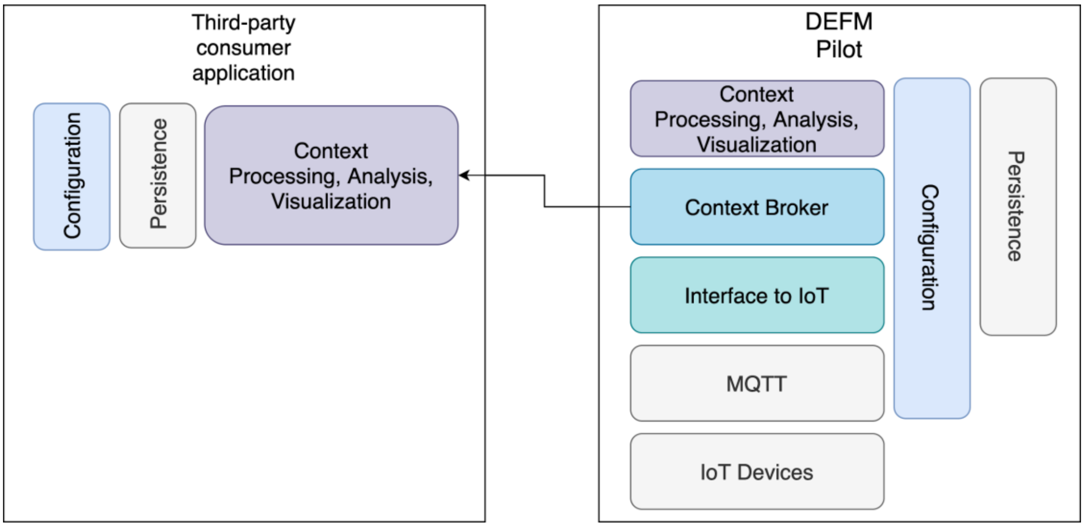


Figure 22. Connection of the DEFM pilot with a third-party consumer application.

7 Summary

The key features of the SOFIE Architecture, as earlier discussed in 'Deliverable 4.3 - First Architecture and System Evaluation Report' [D4.3] and 'Deliverable 4.4 - Second Architecture and System Evaluation Report' [D4.4], are the following: 1) decentralisation, 2) open business platform support, 3) federation, 4) multiple ledgers support, and 5) combined support for trust, security, transparency, availability, and accountability.

The SOFIE Architecture is *decentralised* by design and involves multiple self-contained components, which can be combined to provide the service and satisfy the requirements of applications. This design allows, e.g. the distribution of the load of ledger operations to multiple entities to increase scalability and throughput.

SOFIE's architecture and framework components are open with clearly defined operations and interfaces, contributing to *open business platforms*, which allow anyone to join by conforming to the SOFIE Architecture and the practices of the business platform. The openness is further assured by the fact that no single party controls the platform, as SOFIE builds on the *federation* approach, where anyone can join the platform as an equal member.

The support for *multiple ledgers* may more accurately reflect the interaction between different parties, and allows different tradeoffs in terms of technical properties such as transaction cost and delay, smart contract capability, transparency, and privacy.

Security is supported by complementing existing privacy standards with revocation and immutability through the use of ledger technologies and smart contracts, and *privacy* is supported, e.g. with anonymous identifiers and privacy-preserving data sharing using local differential privacy mechanisms. Finally, *trust*, *transparency*, and *accountability* are supported through distributed ledgers and the interledger functionality, while *availability* is an inherent feature of distributed ledgers, and the different types of ledgers, such as public, private, and permissioned, feature different tradeoffs among these features.

Therefore, the added value of the SOFIE Architecture does not rely solely on novel component implementations but more fundamentally on the way they are used together to enable the *secure and privacy preserving federation of IoT systems* over organisational boundaries. So, the individual components can be implemented purely using existing standards and software thus supporting interoperability with legacy systems, promoting higher security, and speeding up deployment, but when existing standards do not suffice, implementing new solutions is also possible.

Both approaches are exemplified in the SOFIE Framework, which is an implementation of the SOFIE Architecture for use in the SOFIE pilots (it is described in SOFIE Deliverable 2.7 [D2.7] and available as open-source software in GitHub [Framework]). It provides an example implementation of all the components and several federation adapters, and it can be used as is, or the individual components and adapters can be extended or even replaced with alternative implementations to better suit the application requirements. This modular approach makes it easy to adapt SOFIE to different IoT use cases, and the use of federation and ledger adapters enables support for IoT devices *without requiring any changes to the devices themselves*.

References

[D2.4] Y. Kortesniemi et al. “SOFIE Deliverable 2.4 - Federation Framework, 2nd version”, December 2019. Available at: https://www.sofie-iot.eu/results/project-deliverables

[D2.7] Y. Kortesniemi et al. “SOFIE Deliverable 2.7 - Federation Framework, final version”, December 2020. Available at: https://www.sofie-iot.eu/results/project-deliverables

[D4.3] V.A. Siris et al. “SOFIE Deliverable 4.3 - First Architecture and System Evaluation Report”, December 2019. Available at: https://www.sofie-iot.eu/results/project-deliverables

[D4.4] V.A. Siris et al. “SOFIE Deliverable 4.4 - Second Architecture and System Evaluation Report”, April 2020. Available at: https://www.sofie-iot.eu/results/project-deliverables

[D5.2] I. Oikonomidis et al. “SOFIE Deliverable 5.2 - Initial Platform Validation”, June 2019. Available at: https://www.sofie-iot.eu/results/project-deliverables

[D5.3] I. Oikonomidis et al., “SOFIE Deliverable 5.3 - End-to-end Platform Validation”, July, 2020, Available at: https://www.sofie-iot.eu/results/project-deliverables

[Erl2014] U. Erlingsson, V. Pihur, and A. Korolova, “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response,” in Proc. of ACM SIGSAC Conference on Computer and Communications Security, 2014

[Framework] SOFIE Framework, an open-source software implementation of the SOFIE Architecture, available at: https://github.com/SOFIE-project/Framework

[Fot2020] N. Fotiou, I. Pitarras, V.A. Siris, S. Voulgaris, G.C. Polyzos," OAuth 2.0 authorization using blockchain-based tokens," Proceedings of the NDSS 2020 Workshop on Decentralized IoT Systems and Security (DISS), San Diego, CA, USA, 2020

[Ree2020] D. Reed, “Decentralized Identifiers (DIDs) v1.0 – Core architecture, data model, and representations”. W3C Working Draft, 12 October 2020. Available at: https://w3c.github.io/did-core/.

[Sir2019] V.A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, G. Polyzos: Interledger Approaches. In: *IEEE Access* Bd. 7 (2019), S. 89948–89966