SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D2.2

Federation Architecture, 1st 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	31.8.2018
Author(s)	Santeri Paavolainen (AALTO), Tommi Elo (AALTO), Mikael Jaatinen (LMF), Giuseppe Raveduto (ENG), Vasilios Siris (AUEB-RC), Henri Lakk (GT), Dmitrij Lagutin (AALTO), George C. Polyzos (AUEB-RC), George Xylomenos (AUEB-RC), Nikos Fotiou (AUEB-RC)
Responsible person	Santeri Paavolainen (AALTO), santeri.paavolainen@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

1. Introduction

This deliverable outlines the first version of the SOFIE federation architecture. This document will be updated during the project, thus until late in the project it will remain in draft state.

1.1 Overview

Federation in information technology means the ability to connect two or more systems (of some kind) together so it is possible to operate (in some way) across these multiple networks. Federation implies the existence of some common standards and interoperability between the different networks. In the case of IoT systems, the "networks" are organizational "silos", and the goal of federation is to enable operation across these silos. Also note that in federation the different networks (or IoT silos) retain their internal control and are able to determine the policies they enforce for federation with other systems.

When discussing "organizational silos", they encompass all kinds of technologies, including legacy mainframes, databases, enterprise message buses, customer databases etc. SOFIE's goal is not to enforce any change into the internal structure of these organizational silos. The only relevant operations that SOFIE can address is what happens **between** different organizations and entities, the purpose of those operations, and the mechanisms and technologies those operations are based on.

For SOFIE, there are some relevant constraints:

- **Internet of Things devices**. While more or less any device can be connected to the Internet, within the scope of SOFIE we are specifically interested in devices that perform a function that is primarily something else than interacting with humans or other systems (e.g. they perform a real-world function and/or interact directly with other IoT devices).
- **Distributed Ledger Technologies** (DLTs). DLTs, including blockchains, offer a way to define a neutral technological platform. The neutralness can allow new kinds of business practices to take place, which previously have not been possible without expensive setup costs (e.g. contracts).

While the IoT environment can be used in myriad ways, for SOFIE the focus is initially on a few operations that are to be federated:

- **Data transfers.** The main problem is not how to transfer data (FTP, HTTP and other protocols have existed for a long time), but the various other problems associated with what happens *before* actual data transfer takes place:
  - Locating the data, e.g. discovery. The scope of the discovery may be also local (local network, local geographic area), national or even global.
  - Identifying the data. Description of the data, formats, access methods etc.
  - Data governance and provenance.
  - Identification and authorization of the destination. Often the requestor needs to be identified and have appropriate access privileges.
  - Contracts and payments. Setting up the data transfer may require the requestor accepting legal terms, and performing a payment, or committing to a payment done later, to a subscription model of payment, or to a pay-as-you-go billing.
- **Remote actuation**. New applications may require or permit the requestor to change the state in an IoT system or device. The same problems as with data transfers apply to this case, too.

The subtlety here is that actual transfer of data, or actual operation of remote actuation can be performed easily, for example adopting HTTP, MQTT or CoAP - all existing solutions - using GET and PUT primitives, but that there are various steps that need to occur **before** transfer or actuation can take place such as establishing of trust, payment, or other similar prerequisites.

Thus, SOFIE's focus is on defining these preceding steps, and identifying the mechanisms that DLTs are applicable to. Very often, this leads to definitions of *business platforms*, e.g. platforms that define certain business processes, logic, entry and exit conditions and so on.

One important goal is to have the setup and associated transaction costs to be low or practically nonexistent. All of the operations described above are achievable today - albeit often at a high cost due to the number of parties consulted and manual operations required. This limits the possibility of inter-organizational IoT operations to high-value operations. The goal of defining a federation architecture is to significantly lower these costs, and decrease drastically the time for new inter-organisational interactions, to practically nil.

1.2 Structure

In this Introduction section, the next subsection below provides more concrete goals defined for this document (e.g. what questions this document answers), followed by a description of concrete constraints that apply. In this section also a short review of existing solutions in the problem space is given.

The following sections proceed with more detailed description of the SOFIE federation architecture and the various systems and components it is comprised of.

Please note that this architecture document **does not mandate any specific technology**. For that, see the upcoming deliverable *D2.3 - SOFIE Federation Framework: Part A: Interfaces and Protocols* document that details the set of specific technologies and deployment configurations that follow this SOFIE Architecture specification. Also see the D2.3 - *SOFIE Federation Framework: Part B: Business Platforms* document which has information on specific business platforms that are within the SOFIE project scope (e.g. pilots) and how they are designed and how they conform to the overall SOFIE architecture. Please see the figure below for a visual description of the relationships between different SOFIE documents, implementations, and pilots.

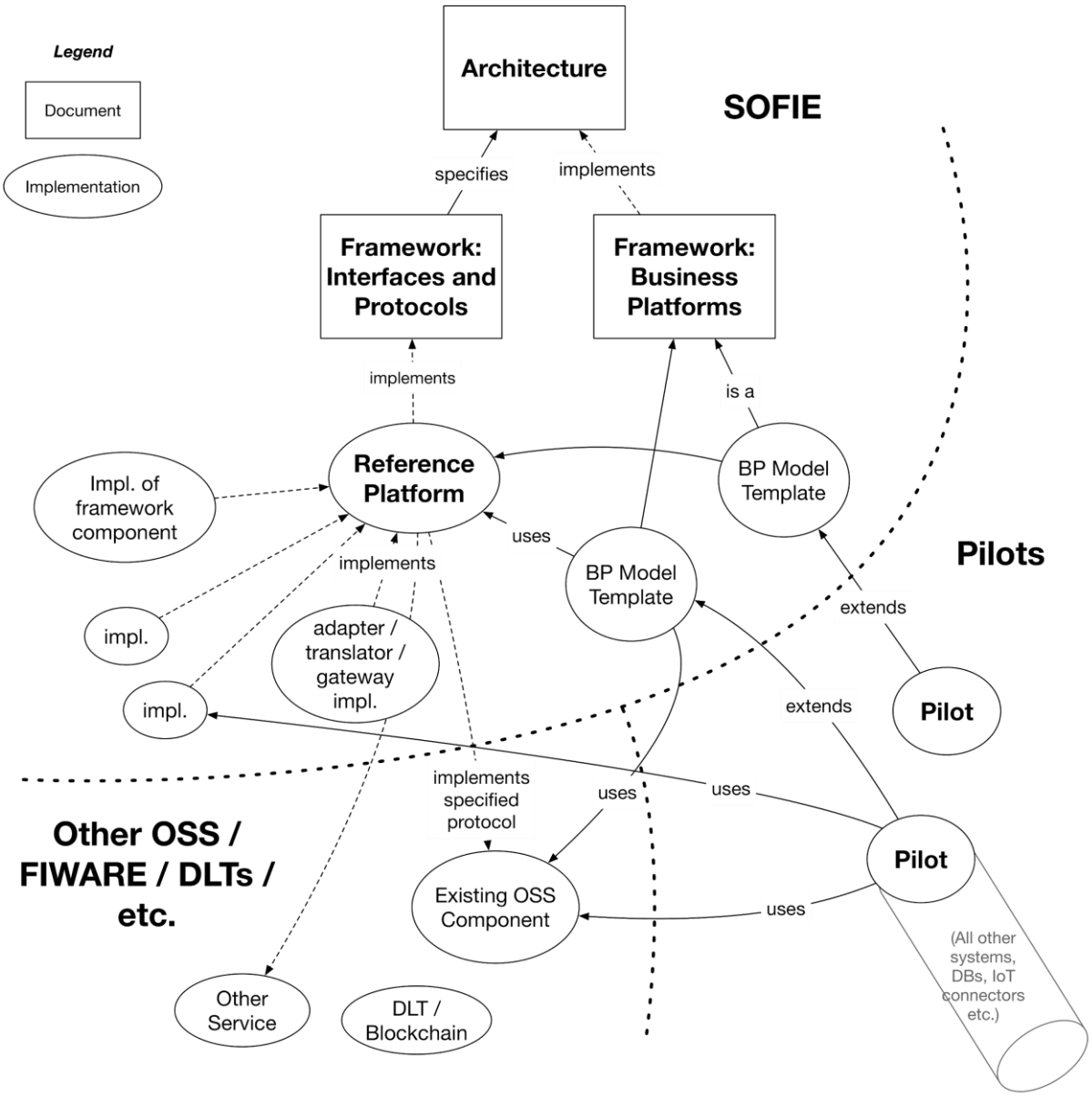

Figure 1: Relationships between different SOFIE architectural elements, documents, implementations and pilots. Rectangles signify documents and ovals signify implementations.

It is important to realize that while the SOFIE project in itself can produce implementations for specific needs, SOFIE is not a **particular** implementation or any specific deployed environment. This document lays out requirements for SOFIE (the architecture) and the framework document gives out specific protocols and interfaces that conform to these requirements. These protocols and interfaces typically are either fully or partially implemented by existing open source projects, but if they are specific to SOFIE, then a reference implementation is provided as part of the SOFIE project results. The business platform model templates represent generic implementations of a specific business platform. These represent implementations of the requirements and specifications provided in this and the framework document.

1.3 Goals

This document answers the following questions:

1. What are the different entities and roles that can occur within the context of SOFIE-federated systems?
2. What kinds of operations do SOFIE systems support? What are the roles and responsibilities taken by different entities during these operations and what are their sub-steps?
3. What types of interfaces are required for these operations and steps and are they part of the SOFIE Framework specification scope?
4. How are the results from the above tied to a cohesive architectural framework?
5. What kinds of deployment / integration / implementation configurations are possible and/or anticipated?

1.4 Constraints

The architecture itself is constrained by both what is feasible in the real world, and by the overall project goals. Some of these constraints are listed below.

- No mandated central governing body for deployed environments -- if an organization wants to deploy a “SOFIE system”, use a SOFIE component, or interact with another organization’s “SOFIE-enabled” system, there is no centralized governing body that has access or can mandate anything on the environment they have deployed. (The only restriction should really be in licensing of the term of “SOFIE Compatible” etc. if something like that ever comes along to uses of SOFIE interfaces that conform to the specifications.)
  This does not exclude different entities from establishing a governing body for the environments they have chosen to deploy and integrate. The requirement applies only to the deliverables of the SOFIE project, e.g. they do not form a control mechanism for the use of, or access to, components defined, used, or distributed by or in the SOFIE project.
- The state of distributed ledger research is moving rapidly forward, thus it is infeasible to put down hard requirements on any particular technology (possibly to be superseded in the near future) as well as setting specific goals on DLT capabilities (potentially not reached in the near future).
- (This list will be elaborated in later versions based on concrete understanding from pilot deployment and testing.)

1.5 Role of distributed ledgers

1.5.1 Types of ledgers

While the use of IoT devices, standard protocols etc. would probably be quite straightforward for most technical people, perhaps a few words about the role of distributed ledgers within SOFIE is warranted. Distributed ledgers encompass a wide variety of individual technologies, ranging from public blockchains supporting cryptocurrencies such as Ethereum, to permissioned DLTs such as Guardtime KSI Blockchain and Hyperledger Fabric, and even to non-blockchain DLTs such as Hyperledger Indy (e.g. Sovrin). These offer also widely varying types of service, levels of security and performance guarantees.

The use of DLTs in SOFIE is based on three tenets: 1) they offer a ledger (e.g. append-only data structure) secure against tampering[1], 2) they are decentralized and distributed, meaning no single node in the DLT network has authority over others, and the data in the ledger is distributed against single points of failure, and 3) they enable the deployment of distributed

[1] Internal and external tampering can have different security requirements. For example, permissioned ledgers are secure against external tampering provided the actual permissioned nodes remain secure, but may not be able to provide non-tampering guarantees against an insider attack.

applications that are either directly evaluated by the DLT in deterministic manner, or evaluated externally in a way that allows third parties to verify the correct behavior of different parties. These allow distributed ledgers to be used for cases such as these:

- Public ledgers as a neutral platform for establishing agreements between distrusting parties. Since the ledger is machine-readable, this could for example be a statement for mutual access rights to specified IoT devices, claim about a state of an internal system (a private ledger, for example, useful for potential dispute settlements).
- Public ledgers as publishing a permanent record, for example, of a description of an internal state of a system.
- Public ledger as a globally reachable neutral platform.
- Consortium ledgers, where (potentially distrusting) consortium members operate a permissioned ledger with consortium members operating their own nodes. This would allow members to use the ledger, while having certain guarantees about its integrity. The ledger could be open for access by everybody to allow the state to be inspected, or access could be restricted so that data confidential within the consortium could be stored.
- Various other combinations are also possible, where some functionality of the ledger (such as adding transactions) are permissioned, while others may be permissionless and open for anyone to join (such as verification).

Using a public ledger supporting smart contracts, for example, may allow an organization to set up a marketplace for trading and creation of agreements --- a customer could perform a payment using a cryptocurrency for IoT system access, or a third party could join the marketplace. Note that when spanning the digital domain of a distributed ledger to the physical domain of an IoT system, it is generally not possible to extend the integrity and security guarantees of a DLT without placing a level of trust to the other participant (after all, the IoT system provider can just take the payment and run, or provide a mock system providing useless data values.)

1.5.2 Inter-ledger operations

A final piece in this puzzle is the concept of **inter-ledger transactions**. These may take different forms depending on the situation, with the core commonality being that they form a protocol that in one form or another links the state of two or more ledgers together:

- Private state (of a private or consortium ledger) is published in a public ledger, for the purpose of establishing a fixed point of private system state (potentially useful in dispute resolutions).
- Link IoT records on different blockchains, where an IoT record on one blockchain depends on one or more IoT records on one or multiple other blockchains.
- Inter-ledger payments (see Interledger Protocol, for example), where a payment is initiated in one ledger and results in the payment of funds in another ledger, transgressing different real or virtual currencies.

Please note that in *general*, it is not possible to perform inter-ledger operations "transactionally" in the ACID[2] sense of "transactions" between non-cooperating ledgers. Consider for example two smart-contract enabled ledgers, where a payment in one is assumed to result in a state change in the second one. Since these ledgers would be self-contained, performing a transaction from one ledger to another requires a third party that confirms the payment in the first ledger, and then performs an operation in the second one. This implies trust in the third party. If such trust mechanisms can be created (such as in ILP

[2] Atomicity, Consistency, Isolation, Durability - these are a set of properties generally desirable for data manipulation operations. Conventionally, databases are able to provide full ACID properties for transactions, although often these constraints are relaxed for performance and availability reasons.

relying on peering contracts between connectors), then mechanisms such as two-phase commit can be employed.

Future developments on cross-ledger interoperability may provide developments in this area, however, for currently widely deployed permissionless ledgers in the general case, “ACID transactionality” is not feasible in inter-ledger operations. Thus, with “transaction” we refer to some form of inter-ledger operations that usually result in state changes that can be verified later to have occurred correctly, but cannot be ensured to be atomic or isolated natively in the current generation of distributed ledgers. While ledgers do not generally support native inter-ledger operations, these are possible using other techniques that rely on external communications (e.g. an approach a bit like using an external transaction monitor) such as timed hashlocks that can be used to gain transactional atomicity under very general ledger requirements and operating conditions. Inter-ledger operations are very much an area under research and active development.

2. Architecture

This section provides an overview of the architecture of a SOFIE system. As described earlier, this architecture is more of a descriptive than normative architecture. However, here we provide an architectural framework that provides the structure and terminology that is used later for describing concrete SOFIE systems with their own unique features and functionalities. In general, the SOFIE architecture's goal is to be modular, with clean separation of concerns between different service features. Thus, while in an implementation many of these features and components may be integrated into a single application, service or device, here they are handled as if they were separate implementations (somewhat akin to microservice architectures).

In the conceptual SOFIE model, we define a **service** to contain a set of features. If a component does not conceptually contain all of these features, then we would define it as either an external service (e.g. not fulfilling all of the features listed here), or a gateway, a translator or an adapter component. The features that we would consider a (full SOFIE) service are:

- Discovery
- Service description enabling and supporting features such as privacy and data protection compliance, data provenance and governance, resource access endpoint description and referencing of other federated resources
- Explicit definitions of technical API endpoints for resource access

This is shown in Figure 2 as a high-level view. It is important to realize that here we do not define what a "service" is, how it is implemented (e.g. in a DLT, in conventional application server or as a serverless service), or what functionality it implements.

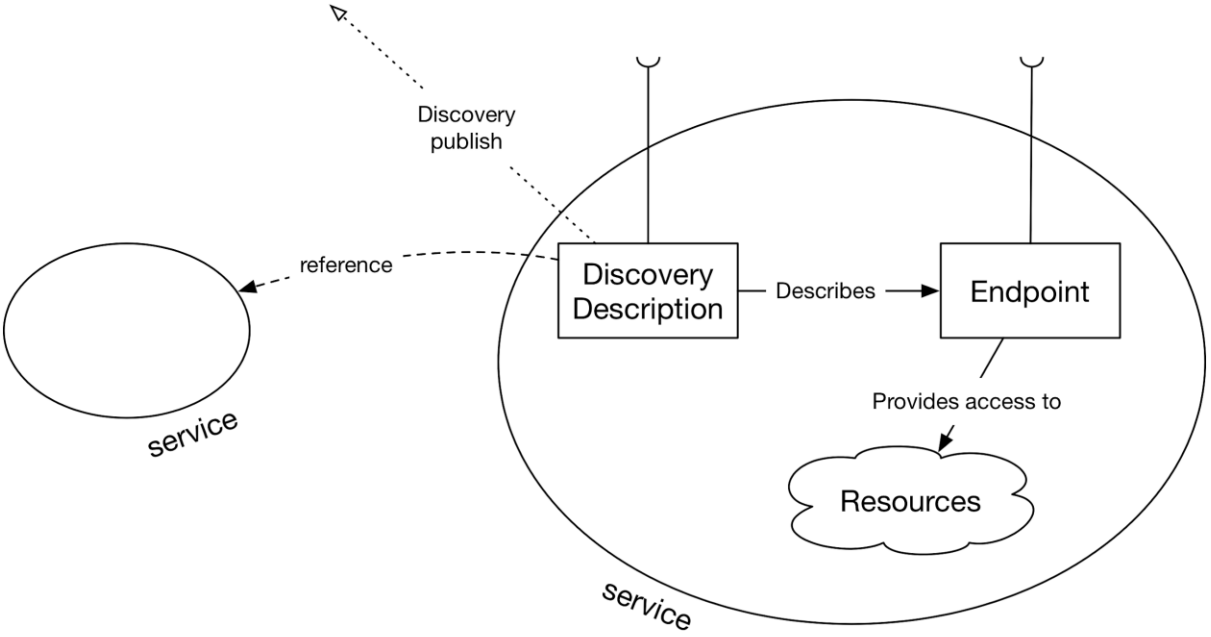

Figure 2: High-level view of the SOFIE system architecture.

A more comprehensive description is shown in the figure below. Here we introduce the concept of a **service ecosystem** that describes multiple different services (typically governed by the same entity) that form an interconnected system. In the diagram for example, we have a "catalogue" service which in itself does not implement access to resources (alternatively we can view it as providing access to a "null" resource, defining no access methods), but provides references to other services in the ecosystem. The different systems in the service ecosystem

can implement features such as a gateway for IoT device access (when those devices cannot be directly accessed), a service offering protocol translation, general resource access API service, a notary service residing in a DLT and an authentication service. These services in turn can refer to other service either within the ecosystem (authentication, for example) or external services (such as third-party authentication).

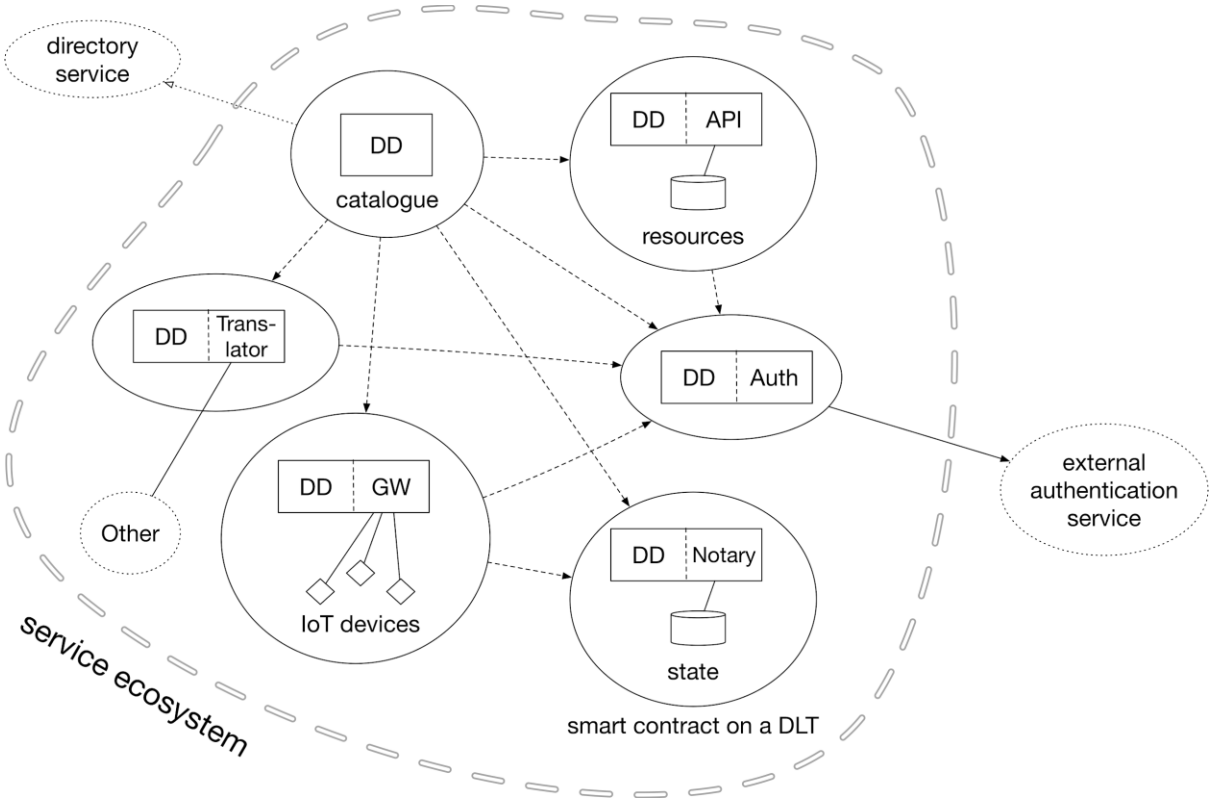

Figure 3: A collection of different services, potentially under governance of a single entity, that comprise together of a service ecosystem. Here "DD" refers to service discovery and description mechanism.

This is a description of a meta-architecture: at this level it is not possible to specify implementation architectures of these services, since under the assumption of retaining organizational silos, it is not possible to mandate anything inside such a silo - only those parts that operate between silos e.g. external interfaces and protocols. Consider the IoT device gateway above, for example: the gateway may provide a limited set of actions for external users, then map these actions into multi-step queries sent to individual IoT devices over a local non-IP-protocol radio network. The gateway itself may internally be implemented on a cluster of server applications running in a containerized environment sharing a peer-to-peer data storage layer.

The sections below describe in more detail what kinds of entities we think are likely to operate within a SOFIE system, the roles they can take, and various lower level components that are relevant to further this understanding. Furthermore, a more detailed description of different core operations that we envision is provided, with a more concrete list of requirements that we see for different kinds of interfaces a SOFIE service, or a service ecosystem needs to implement.

Note: This section is expected to be expanded when understanding of generic patterns used in pilots and BPs is gained during the project lifetime.

3. Entities, roles and components

3.1 Overview

There are many different entities in a SOFIE system and several roles that different entities can work under. The entities and roles discussed below cannot be an exhaustive list for any practical deployment, which in turn requires the understanding of related business entities and roles. First, a short introduction to different entities and roles is given, and then, later, these are described in more detail.

On the highest level we are talking about **organizations**, organizations interacting with other organizations and individual **persons** interacting with an organization.[3] The interactions that are interesting here occur primarily via either policy-driven **software services**, software applications, or software **agents**. The software services may reside in **IoT devices**, IoT service **gateways**, as smart contracts in **distributed ledgers**, or other service environments (Cloud, serverless etc.), including mobile devices. Also, the difference between a human-controlled application, or a software agent performing operations either fully or semi-autonomously on behalf of a human and a software service operating under a policy should be made. Also note that a “service” in our context may also include capabilities for payment processing, identity validation etc.

Each of these different entities can take different roles such as **user** or **client**, **service provider**, **intermediary**, **payer** or **payee**. These may have further classifications such as a user of a service being either a customer, an administrator or a manager. In some cases a single entity can assume the same role for multiple different purposes, or assume both roles of a role-pair (such as both sending and receiving payments).

[3] While person-to-person interaction related to IoT devices is a possibility, it is not in focus in SOFIE.

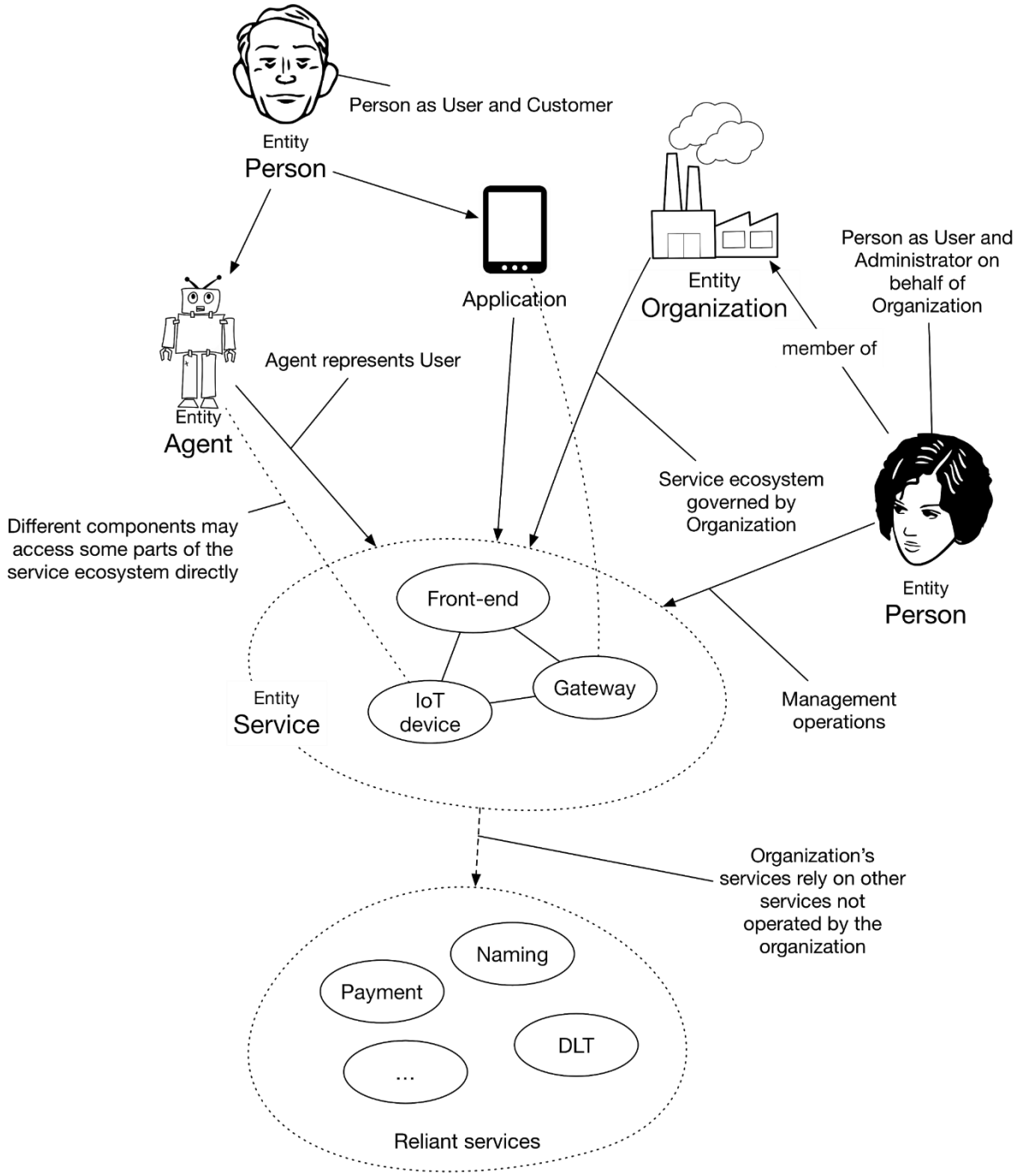

Figure 4: Example of different entities, roles and relationships in an organization's service ecosystem

3.2 Entities

The table below lists various entities that can be considered to be part of the SOFIE system in one form or another.

Table 1: Entities and their description

Name	Description
Person	(Todo in a later version.)

Organization	(Todo in a later version.)
Service	(Todo in a later version.)
Agent	A software, service (or a person or organization) acting *independently* on behalf of a person or organization within the constraints defined by the originating entity.
Device	(Todo in a later version.)
Gateway	(Todo in a later version.)
Distributed ledger	(Todo in a later version.)
Smart contract	(Todo in a later version.)
Oracle	Entity allowing smart contracts to interact with the Internet (e.g. call APIs). An oracle can also be viewed as a highly specialized form of an agent acting on behalf of a smart contract.

3.3 Roles

The table below lists different roles some of the different entities can take when interacting with a SOFIE system.

Table 2: Roles and their description.

Name	Description
User	(Todo in a later version.)
Client	(Todo in a later version.)
Provider	(Todo in a later version.)
Intermediary	(Todo in a later version.)
Payer	(Todo in a later version.)
Payee	(Todo in a later version.)
Payment processor	Separate role; consider a smart contract as a payment processor for example. A possible mechanism of use would be that 1) payer receives a signed datum from gateway (non-DLT) and a smart contract address, 2) payer sends a payment with the datum to the smart contract (or via ILP, for example) and the smart contract verifies the datum, and if valid, accepts the payment and logs the payment as received, 3) the payer can now take the transaction receipt back to the gateway which in turn can verify that the payment occurred, and can for example, hand off 4) an access token. Of course, a payment processor can be PayPal or anything… There are no universal standards though that these payment processors would conform to. (Web payments someday?)

Trusted third party	Entity that is *trusted* by all parties in a transaction.

3.4 Components

The table below lists some components that may represent an entity (acting out with some role) in this system. These are more like examples than normative descriptions.

Table 3: Some components in a SOFIE system and the entities and roles they represent.

Name	Description
N/A	(To be filled in later versions once more concrete understanding of the components is gathered through pilots.)

4. Orchestration and Governance

Note that this section is descriptive and not normative. It aims to provide rationale for specific requirements that come in later sections.

4.1 Overview

As noted before, the process of reading an IoT device value, actuating an IoT device, or performing some other intermediary operation often is a chain of distinct substeps.

First, one needs to locate the service. Often this is previously known either directly, or can be discovered through a known intermediary service. Sometimes even this information is not available, and a broadcast or multicast discovery operation must be performed. Regardless whether the **discovery** step is implicit or explicit, simple or multi-phased, requiring user interaction or not, it is the first step for federation.

Identifying the service and its capabilities is most often considered as part of discovery process, but we'll consider **service identification** as a potentially separate step. The rationale for this is that sometimes the endpoint is well-known, but it can change its behavior (new versions of the service etc.) that may change its capabilities while retaining old behavior in a backward-compatible manner. To allow more intelligent clients to use newer functionality, a mechanism to identify the service capabilities is needed. This may be explicit identification in the form of open-ended service description (service metadata), or implicit through service version numbering. Also while the service is identified, often the service itself needs to be authenticated for establishing a trust relationship in some manner - if the service is discovered through a trusted intermediary, this is implicit, but for broadcast type discovery, some form of trust in the fact that the discovered service really is offering the service it claims needs to be made. This could be accomplished through use of certificates of trusted service classification entities, or based on feedback from other users etc.

Also note that accessing service description may be an iterative process tied to identification[4] and access control - it is possible that the initial service identification will show only a rudimentary set of services, and describe the access policy, allowing the user of the service to bootstrap their access and then gain access to a more complete service description.

Apart from data accessible by open public, a form of **user identification and authentication** is present either explicitly (OAuth2, SAML or other credentials) or implicitly (source address for smart contract invocations). The actual access control is implemented by the organization responsible for the data or device, and not really part of SOFIE, although the metadata for describing the access and identification/authentication mechanisms could be. While external identification and authentication has several applicable standards, the underlying access control mechanism are usually highly specific to the business requirements of the controlling organization, and cannot be expressed in general terms, or at the SOFIE architectural level.

For business purposes, mechanisms that act as gates that require either legally binding contractual **agreement**, or some form of **payment** or payment escrow are critical. The transaction costs and speed of payments are more likely to be relevant for B2C (or C2C) than B2B. Agreements can be almost anything, but typically these would include EULAs or other liability risk mechanisms (for businesses). Sometimes it may be possible to automate these steps fully or partially, but often in the case of agreements, human interaction is expected and required.

[4] We use the term *identification* instead of *authentication*. Whether an organization requires a user to be *authenticated* is a policy issue, but almost always we need to *identify* the accessor, even if they are otherwise pseudonymous. In the later example the coffee dispenser would be interested in ensuring that all entities accessing it have accepted the EULA, but the actual identity of the person who is accessing would be required only when a dispute is handled.

4.2 Sample scenario

Let's take a hypothetical (and quite unrealistic, actually) scenario that demonstrates these steps:

> *A person has a "coffee dispenser application" in their mobile device. Wishing to get a shot of caffeine, they fire the application up. The application performs a discovery protocol on the local WiFi and 5G network neighbourhood, looking for services that advertise "coffee dispenser profile."*
>
> *It finds some services directly, and a few intermediary services. The application queries these services, using the defined minimum protocol version for further information. The intermediary services require EULAs to be agreed to before they provide further information. One intermediary has a standardized benign EULA that the user has pre-approved - the application demonstrates the pre-approval for the intermediary. The other has an agreement not yet confirmed and not on the pre-approved list. The application queries the user on how she wants to proceed - "Intermediary X wants consent before providing information - Ignore now / Decline permanently / View agreement?" The user decides to decline that particular intermediary, which the application will remember in the future.*
>
> *Now the application can query both the direct dispensers and the intermediary, and narrow down the dispenser selection to nearby geographical area. The user selects a nearby coffee dispenser - directly, not via intermediary, so the application will talk directly to the dispenser. The dispenser then asks for a payment. Since the application supports the required payment method, after verifying with the user, the application receives an electronic dispensing token.*
>
> *The user starts walking towards the dispenser, with the application tracking user's location relative to the dispenser. Once user is nearby, the application - as per user's preferences - automatically starts coffee dispenser with the given token, notifying the user of the time remaining until their brew is ready.*

In this scenario, the service discovery occurs both directly (broadcast or multicast), but also via intermediary (aggregator) services. Using an EULA requires identification - although in this case, a pseudonym is likely to be sufficient. The whole coffee dispensing profile can be a de facto standard, as well as the protocol used for dispensing. Please note that while the profile and actual dispensing protocol are most definitely not part of SOFIE, the supporting mechanisms potentially are. So, SOFIE is not interested in the actual wording of the EULA, for example, but about the capability to classify and present them during IoT discovery and agreement process.

One final step is often required - disputes. In real world, while everybody would be happy without any disputes, in practice they occur. While this field is largely unautomated (no APIs), it needs to be covered in some form. The minimal mechanism is to include an URL that a human can use for initiating a dispute, or alternatively provide other information required (a good business practice) to initiate a legal process. This generally applies to B2C situations, where it would be necessary for the business to prove its identity during the interaction to the consumer. The use of DLTs may offer improvement to current state of the art, as the transmission of electronic tokens and corresponding payments can be immutably linked and recorded on a DLT, and used if necessary for handling disputes.

4.3 Elementary operations

The core operations that would be required for the scenario listed above are:

Table 4: List of elementary operations

Operation	Description
DISCOVER	How to discover that a service exists?
DESCRIBE	What endpoints does the service support, what are its requirements for authentication etc.?
ACCESS	Actually access the defined resource endpoints, which can be data retrieval, control operations, remote actuation, etc. as semantically defined by the service description
IDENTIFY	Provide identification of the user or client if required
AUTHENTICATE	Perform authentication of the user or client if required
PAYMENT	Perform a required payment
AGREE	Agree on a contractual term (EULA, etc.) required for resource access
WITHDRAW	Withdraw from a previous agreement

These are high-level operations that may be implemented in very different manner on different service implementations - consider that on a web service, service discovery and description occur over HTTPS, but on a DLT this may occur via calls to a smart contract.

In the subsections below, each of these elementary operations is described and discussed in detail. Please note that this is a **descriptive specification**, and any protocols or implementations mentioned are for illustrative purposes only. The actual specification of protocols is done in the SOFIE Federation Framework document. Also keep in mind that the operations described are categorical, e.g. there is not necessarily “a” REST API endpoint for IDENTIFY behavior, depending on the details this might be implemented entirely on the client side, or using other external services, or part of the service (or the related ecosystem) itself.

4.3.1 DISCOVER and DESCRIBE

The actual method of finding that a service even exists is a multi-faceted problem. Are you trying to perform service discovery on the local subnet, or local administrative network? How about services provided by the cellphone substation, or within a small geographic area? There are also existing protocols that provide support on specific contexts - consider UPnP for example, which is useful in a local network, but useless across network boundaries.

For DISCOVER, a SOFIE service needs to essentially “be discoverable”. What is relevant is highly dependent on the problem, and can cover the following scenarios:

- Local network: Either broadcasting, or replying to service discovery requests on the local network, if applicable. (For example, if one is creating an internet-facing service, then local network discovery is useless since any access crosses network boundaries, thus rendering local discovery useless.)

- Known service endpoint: Discovery is often implicit in this, e.g. a service *description* address is known beforehand, and if that is accessible and provides information, then the discovery has implicitly happened.
- Known catalogue endpoint: A catalogue endpoint can be something such as a domain name, in which case DNS service records or WebFinger would be appropriate. Similarly some specific problem areas have known locator addresses (consider registrar services in Ethereum blockchain, for example).
- Geographical area: … are there any mechanisms that actually support geographical service discovery … ?

For DESCRIBE, it is important that the service describes the following information:

- Service API endpoints and their requirements, on contractual and technical requirements such as the need of identification and authentication by entities accessing the resource
- Any references to other services and external resources, either as part of API endpoint requirements, or as a catalogue service
- Privacy and data compliance information, including things such as references to privacy policies
- Data governance and provenance information (this can be part of the data that is retrieved too, but potentially as part of the service description too)

Note that discovery and description can be a multi-step process. First, it may be required to query whether the system supports a specific kind of discovery protocol at all - for example, in Ethereum an ERC-165 compliant interface discovery process may be necessary, which requires the client to perform multiple queries to confirm that the queried smart contract conforms to a particular interface specification.Service metadata formats

4.3.2 ACCESS

The resource endpoints may support various actions as described with the DESCRIBE operation. Some endpoint actions may require authentication, or may result in different results depending on whether the client is anonymous, identified or authenticated. If identification or authentication is required, the service should return information that allows the client to perform the required action. The same applies for any other prerequisites such as contractual agreements.

In general, any action returning data should provide also information on its governance and provenance. If the actions or returned data are sensitive, then suitable protection methods must be employed (encryption, signing etc.). Similarly, if the ACCESS action adds data to the system, its provenance needs to be considered and kept up-to-date if necessary (e.g. distinguishing between information vouched by the system versus information provided by individual users of the system). Finally, an action may change the system state in a way that causes effects on the external (non-digital, real) world and is defined as remote actuation to distinguish from digital-only state-changing operations.

4.3.3 IDENTIFY and AUTHENTICATE

While the underlying mechanisms of IDENTIFY and AUTHENTICATE may be (in some cases) identical with the only difference being parameterization of the resulting data (pseudonymous vs. identity-encoding), they fulfill different requirements, and may sometimes be implicitly provided in the underlying data transfer protocol.

The reason for having IDENTIFY as a separate operation is that in many cases it is sufficient to be able to distinguish between users and clients without having the need to knowing the underlying personally distinguishing identity. Consider the coffee vending scenario from above - the coffee vending machine may be concerned only about the user accepting an EULA, and paying for the coffee. Yet, for reasons of potential dispute it is necessary for the client to be

able to provide a linkage of itself to a request, likewise for the vending machine to be able to show a specific (otherwise unidentified) entity agreed to the EULA. Given former DPD and current GDPR, it is often a sensible approach of not collecting personally identifiable information at all if avoidable.

In some situations, the IDENTIFY operation can be implicit: in a local network, a MAC address may provide sufficient distinguishing capability. In TLS, the use of self-signed certificates and their (presumably unique) public key may suffice. In other situations, a client-generated identifier (sufficiently large random string) may be acceptable, while in other situations a sufficiently secure token[5] generated by the service itself is necessary.

If the service requires an authenticated identity, e.g. a strong proof for the service that the remote entity is associated with a particular identity vouched by some other, trusted party, then AUTHENTICATE operation is required. In the simplest case authentication occurs with the service (or linked service under the same governance) itself using username-password pairs, in which case the service itself is providing the assurance of the identity to itself. In a more complicated situation the authenticated identity is given by a third party such as national registry or a commercial entity (think Google and Twitter). If TLS and the client-side certificates are signed by a trusted party, then they may provide the necessary authenticated identity information. Also, other protocols and services such as Sovrin may be applicable.

In all situations, the service should follow the best practice of minimizing security information exposure, e.g. the fact of being authenticated is separated from the process of authentication (this is the best practice of issuing secure access tokens in the authentication API, as in OAuth2 for example). Similarly a client must be ready to re-authenticate at any step - the service may provide short-lived tokens for less trusted entities, for example.

As a final note, remember that it is possible to have the client perform IDENTIFY operation followed by AUTHENTICATE operation of the user - meaning that the service is able to distinguish different clients (mobile devices, for example) while accessing the same underlying resource as authorized by the authenticated user.

4.3.4 AGREE and WITHDRAW

While in many situations, a resource may not require any kind of agreement between the service and the end user (just think of DNS). In other cases, the agreement may be implicitly assumed based on earlier actions of the user (for example, in an earlier step by signing up for the service via a web site, the service may assume the user has either explicitly or implicitly consented).

In the minimal level, a service can provide in the DESCRIBE operation, or as part of the metadata in ACCESS, information about the conditions and terms that apply to the service[6]. If explicit consent is required, one must consider the following cases:

- If an agent can perform agreement on behalf of user or organization, the agreement terms must be somehow standardized and uniquely classifiable by the agent. Consider the coffee vending example above, if the EULA is classified as an instance of a particular family of contracts (in some ontology), the agent may have been given permission to automatically agree for those on behalf of the user. (This kind of automatic agreement cannot really occur unless the user can beforehand understand the kind of consent they are providing for the agent.)
- Is non-repudiability required? In many cases, “sufficient proof” may not require that. In case of CC or OSS licenses, the default (of not agreeing) is more restrictive than the one provided by the license.

[5] Token that is generated and authenticated by the service so it cannot be forged by other parties.
[6] Whether this kind of implicit consent is sufficient in a particular case is beyond this document.

- Agreement may be implicit in some other operation such as PAYMENT (when it occurs through a web site providing user with the delivery terms, for example).
- Agreement may be implicitly provided in some operations, in ACCESS for example. The service description may identify certain terms applying to the use of the ACCESS, with the assumption that these apply to any client (and by implication, entity) performing the operation.

AGREE implicitly requires either identification or authentication, as otherwise there is no guarantees provided for the service.

At this point it may be difficult to provide concrete specification for these operations in the framework documents, as it appears that many attempts in this area are dead in the water (such as WS-Agreement). Thus this section is attempting to describe the problem even it may need to be ignored in the SOFIE context.

4.3.5 PAYMENT

Payment is often required for access to a resource. Payments can take many forms, and while currency-based payments are most common today, other forms such as exchange of non-currency tokens or credits, CPU cycles or storage capacity etc. work as payments too. Payment is most often an asynchronous activity where the delay between a payment request to its fulfillment can be anything from seconds to days. Consider a situation where the payment can be fulfilled via a cryptocurrency payment on a fast payment network - here, if the agent is given permission by the user for automatic payments for certain services, the fulfilment may occur within a few seconds of the payment request. Conversely, some payment operations may require offline fulfilment and verification before being completed. In some situations, it is the payee who needs to be polled for completion, sometimes the payer needs to provide a proof of payment to the payee.

Regardless of the actual process involved, this is a very complicated field even within a single jurisdiction, and even more complicated across countries and continents, where different tax policies, currencies etc. may need to be addressed.

5. Interfaces and protocols

Within the SOFIE architecture, several different interfaces and protocols are identified as necessary to meet the interoperability and federation goals. The actual protocols used for these interfaces are defined in the SOFIE Framework document -- in here the discussion is on what requirements are put on the interfaces themselves.

Overall the interfaces discussed here are categorized into the following overall categories:

- Overall protocol and interoperability requirements
- Service discovery and service description interfaces and requirements
- Interface and protocol versioning and resilience towards future updates
- Data provenance and governance
- Identification, authentication, access control and privacy issues

The requirements are following the standard MoSCoW pattern of MUST, SHOULD and COULD classification.

5.1 Protocols and interoperability

Data transport and data encapsulation formats MUST follow specifications of reputable entities such as IETF or W3C.

Any existing interface or protocol that is used MUST have a freely-available, royalty-free specification available.

All protocols MUST use security mechanisms to protect the data in transit if personally identifiable information is transported. All protocols SHOULD use security mechanisms for data transport even in other situations.

Protocols and interfaces SHOULD support interoperability with older and/or newer versions of the protocols, and minimally they MUST decline working with the client if version incompatibility that cannot be resolved automatically is detected.

5.2 Discovery and service metadata

Services SHOULD support discovery and use discovery protocols that are applicable for the service.

Service MUST be discoverable (and describable) if its endpoint is known by the client (prior knowledge).

Services MUST provide service descriptions. The service description format MUST follow existing standards if they are applicable.

Services MAY require identification or authentication for detailed service description. If identification or authentication is required, then an unidentified or unauthenticated service description request MUST be replied with information about the identification or authentication requirement.

Services SHOULD include information that can be used by the client and its entity to contact the service operator. (For example, an URL to the company web page.)

Services MUST include privacy policy, GDPR information, links to them etc. if they process data under the provisions of relevant legislation.

Service description SHOULD support versioning of services and their access methods.

Service description SHOULD include information about any prerequisites for resource access such as authentication, contractual agreements etc.

5.3 Versioning and extensions

Interfaces and data formats SHOULD support explicit versioning.

Interfaces and data formats MAY support extensions, e.g. situations where the clients and services may be interoperable even if they do not support all of each other's feature support of the protocol. (Consider SMTP, IMAP, SSH, IKE and similar protocols where the parties may implement only partially overlapping extensions, and are able to continue operating while settling to use only the common subset of extensions.)

Interface and data format versioning SHOULD be explicit in client operations so that the service is able to detect situations where it can not support the request format.

5.4 Data provenance and governance

Data governance information SHOULD be provided with the data returned by the service.

Data provenance information SHOULD be provided with the data returned by the service.

5.5 Identification, authentication, access control and privacy

Existing standards SHOULD be used for centralized identification (centralized and federated authentication is a well-known problem area).

Identification and/or authentication, and access control SHOULD be separate steps if explicit authentication tokens are used (e.g. passwords, or federated authentication across different systems).

All identification and authentication based on shared secrets MUST be use secure transport methods when communicating over the Internet. Any form of identification and authentication MUST be secure against malicious third parties.

Note that on decentralized authentication mechanisms it is at this point to provide explicit requirements (under research).

5.6 Interledger transactions

Interledger operations MUST be verifiable by external entities. Any failure of complying with the interledger protocol MUST be attributable to a specific entity.

6. Deployment considerations

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 help with 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) and contain both the client and server capability, and translate one protocol to the 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 specifically talks to a specific type of system, for example, an IoT gateway. [this is a bit of semantic nibbling, but IoT gateways are a common term]
- **Native**, e.g. they implement a SOFIE-compliant interface directly.

It is important to realize that for most protocols, SOFIE specifies only protocols that are already widely used and already defined by other organizations than SOFIE. For these it is likely that widespread implementations for interfacing with them already exist. From SOFIE's point of view even if an organization uses these already 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). 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 providing a specific, more narrowly defined functionality (such as a specific business platform).

6.2 Extending existing systems

An important consideration for SOFIE architecture is how well it is suited for use in 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 5 below 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.

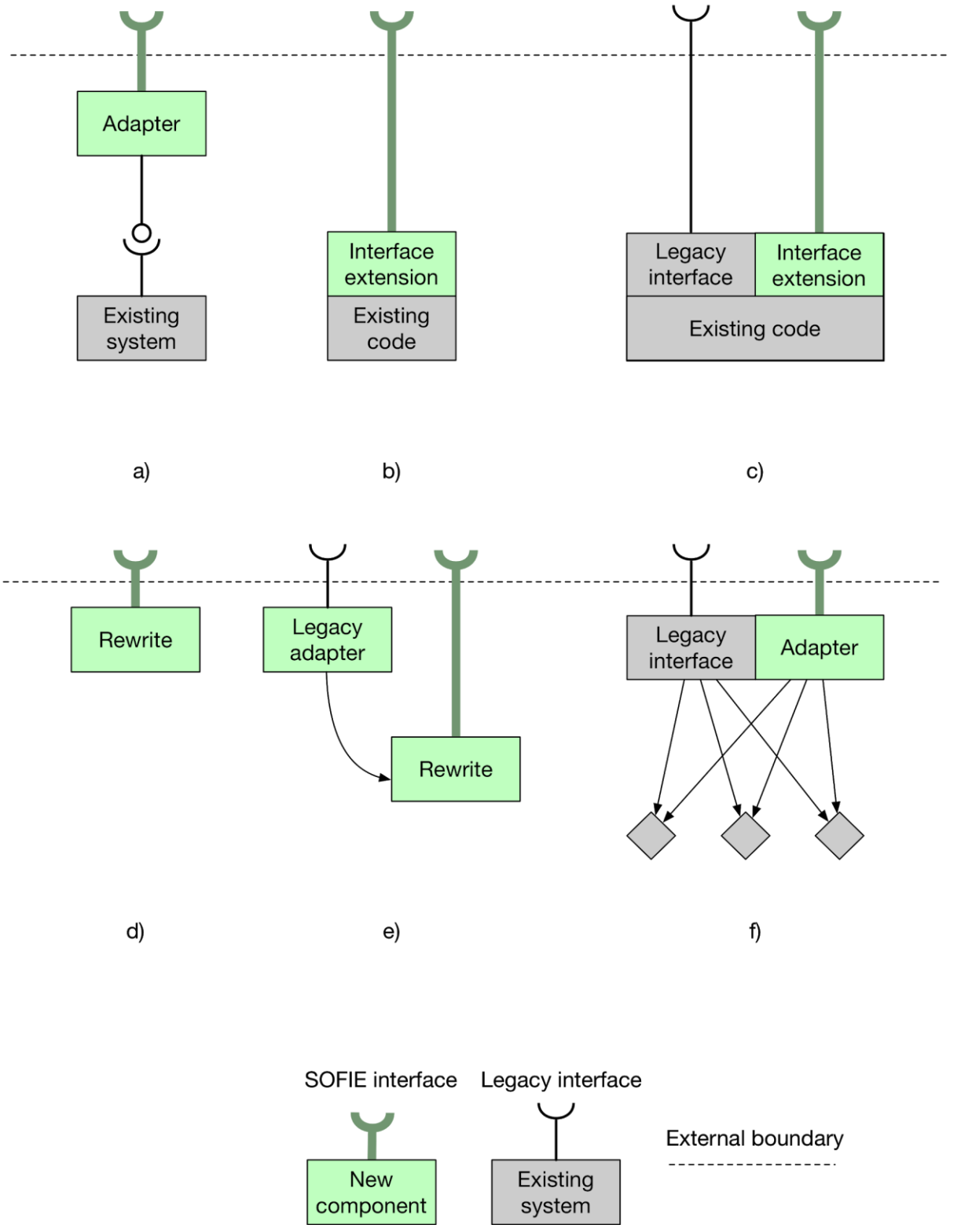

Figure 5: Different approaches to extending existing systems for SOFIE specification compatibility.

Case A: A separate adapter service (or a translator, depending on the complexity of the task) is developed that uses 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 co-existing 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 particular cases and no specific approach can be recommended or assumed.

7. Conclusions

This deliverable presented the first version of the SOFIE federation architecture. It outlines a general SOFIE architecture, describing architectural components and elementary operations. Furthermore, interfaces, deployment considerations, and KPIs are discussed. Based on this work, "D2.3 - Federation Framework, 1st version" will be delivered in October 2018. The architecture itself will continue to evolve, and the following two versions of this deliverable (due in 2019 and 2020) will extend and refine various aspects of the architecture.

Annex I: Architecture KPIs

For project purposes, the SOFIE architecture and framework needs to have measurable objectives (key performance indicators, KPIs). Specifically, this document is concerned only about *KPIs related to the SOFIE architecture and framework*, as other aspects of the project have their own KPIs as needed - for example, WP5 e.g. pilots will define relevant KPIs to evaluate the results of the pilots.

A good KPI is unambiguous, measurable and attainable. If a KPI is gradual (not a yes / no question), then "better" values of the metric should relate to improvement in the overall quality and success of the underlying system. Also, KPIs should try to be orthogonal, e.g. avoid duplicating the number of measures. Similarly, KPIs should not be conflicting, e.g. improvement in a KPI metric should not cause another KPI to decrease.

This annex is simple in structure, focusing on listing and defining the architecture KPIs in the next section.

KPIs

This section lists KPIs defined for the architecture. These partially overlap with project KPIs, as some of the results of architectural decisions can only be evaluated in the overall project scope including all outputs from WP2 (Federation Architecture & Framework), WP3 (Business Platforms Integration) and WP5 (Pilots). Thus, below the term "implementation" refers generally to any components from either WP2 (such as reference platform and business platform templates) and/or WP5 (pilots). Some of the KPIs may be measured in a WP3 environment, while some others may be measured from demonstrators or lab models.

Also, all these KPIs are under discussion and will be updated in the future versions of the architecture.

Table 1: Overview of the defined KPIs.

KPI	Goal	Description	Metric	Method of verification
1	IoT operability	Prove operability of the implementation with IoT silos	Number of IoT silos	Detection of data flow in silos during implementation use case
2	IoT interoperability	Prove interoperability across multiple IoT silos of the reference architecture	Number of IoT silo pairs	Implementation use case accesses data or actuates operations in different IoT silos
3	Ledger use	Validate SOFIE implementation capability with multiple ledgers	Number of distributed ledgers[7]	Ledgers have detectable data passing through SOFIE implementations
4	Inter-ledger use	Validate SOFIE implementation operating across multiple ledgers	Number of distributed ledger pairs	Implementation use case shown to result in operations across multiple ledgers
5	Ledger independence	Demonstrate capability of developing applications	Number of BP samples classified into *success*	Demonstrate that a BP sample can be deployed

[7] Across significantly different ledger technologies, e.g. Ethereum and Ethereum Classic are not considered different ledgers, as their differences are small enough to allow applications developed on Ethereum to be deployed on Ethereum Classic with only minor changes.

		using ledgers, where a sufficient abstraction can be provided to applications to allow them to be targeted simultaneously to multiple ledger technologies	or *partial success*	on two ledgers with only configuration changes, and the BP sample users are able to use either one with only configuration item changes
6	Privacy designed in as a fundamental requirement	Demonstrate GDPR compliance where relevant	Number of operational GDPR features referenced and supported.[8]	Final specifications have clear references to features implementing named GDPR requirements. Relevant pilot specifications also refer to the needed features
7	Device owner payments across ledgers	Ability of silo owners to send and receive payments or other value transfers	Number of ledger pairs supporting value transfer	Observation of value transfer as part of a use case in an implementation
8	Data sovereignty	Ability of data owners to reject or allow access, possibly for a specific time interval, to their data Each datum has an accompanying authorization list, which the data owner can modify	Number of pilot use cases utilizing data owner data sovereignty features, and data owner is from a different silo than the storage silo	Count the number of use cases
9	User responsiveness	Apparent responsiveness of system for end users	Number of seconds user gets response for an action initiated by the user	Measuring from the onset of user action until the user gets a response by the system (to the user interface he or she is using)
10	System performance	Overall system performance reflecting the diverse needs and requirements of different use cases	Acceptable system performance for users and pilots	Qualitative evaluation of system metrics (see below)

System metric

For system performance evaluation, multiple measurable metrics are often required in combination -- a system that precomputes all potential responses in advance may be very responsive, but conversely it has huge storage requirements making it expensive and infeasible to scale in practice. In some situations, the latency of response is critical (especially for user-visible operations), while in some situations long-running asynchronous operations are acceptable. Without specifying the particular use case in detail, it is not possible to provide a specific set of metrics and goals for a "performant system."

The approach for SOFIE is to perform a qualitative evaluation of the system, measuring the system using metrics that are relevant for the use case, and evaluating the holistically in combination. Note that some aspects of a system such as scalability or reliability, while

[8] Number of GDPR articles which lead to operational goals is generally thought to be about 10. See e.g. https://iapp.org/resources/article/top-10-operational-impacts-of-the-gdpr/

important overall, are not in the focus of this project, and are likely to be only evaluated based on design arguments instead of measurements or tests.

The list below specifies some of the **possible** measurements and aspects that can be used in a qualitative system evaluation:

- Response time
- Processing time
- Throughput
- Resource utilization (CPU, memory, network, etc.)
- Scalability
- Availability