Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Final Report on System Requirements and Architecture

The symbIoTe Consortium

Intracom SA Telecom Solutions, ICOM, Greece
Sveučiliste u Zagrebu Fakultet elektrotehnike i računarstva, UNIZG-FER, Croatia
AIT Austrian Institute of Technology GmbH, AIT, Austria
Nextworks Srl, NXW, Italy
Consorzio Nazionale Interuniversitario per le Telecomunicazioni, CNIT, Italy
ATOS Spain SA, ATOS, Spain
University of Vienna, Faculty of Computer Science, UNIVIE, Austria
Unidata S.p.A., UNIDATA, Italy
Sensing & Control System S.L., S&C, Spain
Fraunhofer IOSB, IOSB, Germany
Ubiwhere, Lda, UW, Portugal
VIPnet, d.o.o, VIP, Croatia
Instytut Chemii Bioorganicznej Polskiej Akademii Nauk, PSNC, Poland
NA.VI.GO. SCARL, NAVIGO, Italy

For more information on this document or the symbIoTe project, please contact:
Sergios Soursos, INTRACOM TELECOM, souse@intracom-telecom.com

Document Control

Title: Final Report on System Requirements and Architecture

Type: Public

Editor(s): Pavle Skočir (UniZG-FER)

E-mail: pavle.skocir@fer.hr

Author(s): Ivana Podnar Žarko (UniZG-FER), Konstantinos Katsaros (ICOM), Pavle Skočir (UniZG-FER), Matteo Pardi (NXW), Matteo Di Fraia (UNIDATA), Gabriel Kovacs (AIT), João Garcia (UW), Szymon Mueller (PSNC), Joaquin Iranzo (ATOS), Mikołaj Dobski (PSNC), Nemanja Ignjatov (UNIVIE), Pietro Tedeschi (CNIT), Daniele Caldarola (CNIT)

Doc ID: D1.4-v2.1.doc

Amendment History

Version	Date	Author	Description/Comments
v0.1	03/04/2017	Pavle Skočir (UNIZG-FER)	Initial Table Of Contents
v0.2	05/06/2017	Pavle Skočir (UNIZG-FER)	Updated communication diagrams in Section 5.5
v0.3	12/06/2017	Mikołaj Dobski (PSNC), Nemanja Ignjatov (UNIVIE), Matteo Pardi (NXW), Pietro Tedeschi (CNIT), Gabriel Kovacs (AIT), Konstantions Katsaros (ICOM), João Garcia (UW)	Updates in Sections 4, 5.2, 5.3, 5.4, 5.6, 6
v0.4	16/06/2017	Pavle Skocir (UNIZG-FER)	Updates in Sections 5.4, 5.6
v0.5	19/06/2017	Szymon Mueller (PSNC), Joaquin Iranzo (ATOS), Pavle Skocir (UNIZG-FER)	Updates in Sections 2.3, 5.1, 5.5, 6.1.2
v0.6	26/06/2017	Pavle Skočir (UniZG-FER), João Garcia (UW), Pietro Tedeschi (CNIT), Matteo Di Fraia (UNIDATA)	Added Sections 5.7, 5.8, 6.1.9; updates in Section 6.1.2
v1.0	28/06/2017	Ivana Podnar Žarko (UniZG-FER), Pavle Skocir (UniZG-FER), Mikołaj Dobski (PSNC), Pietro Tedeschi (CNIT)	Updates in Sections 1, 2, 3, 4, 5, 7
v1.1	30/06/2017	Ivana Podnar Žarko (UNIZG-FER), Jose Antonio Sanchez (ATOS), Christoph Ruggenthaler (AIT)	Updates in Sections 5.2, 5.6
v1.2	06/07/2017	João Garcia (UW), Marcin Plociennik (PSNC), Pavle Skočir (UniZG-FER)	Added content in Sections 5.5, 5.6, 5.7, 5.8 Updates in Section 5.3, 6.2.5 Updates throughout the document after internal review
v2.0	10/07/2017	Ivana Podnar Žarko (UniZG-FER), Pavle Skočir (UniZG-FER)	Updates in Sections 5.1, 5.2, 5.4, 6
v2.1	14/07/2017	Ivana Podnar Žarko (UniZG-FER), Matteo Di Fraia (UNIDATA), Pavle Skočir (UniZG-FER) Daniele Caldarola (CNIT), Nemanja Ignjatov (UNIVIE)	Major updates in Section 5.7, minor updates throughout the document

Table of Contents

Table of Figures

Table of Tables

1 Executive Summary

The aim of Deliverable 1.4, entitled “Final Report on System Requirements and Architecture”, is to document the final collection of the symbIoTe system requirements and report the final version of the system’s functional architecture, with the respective components, entities and interfaces. It reports the technical work performed in two tasks, T1.3 (System Requirements, M4-M12) and T1.4 (System Architecture, M4-M18). symbIoTe system requirements have been derived based on symbIoTe use cases reported in deliverable D1.3, while the architecture is built in accordance with the initial architectural sketch proposed in the Description of the Action (DoA).

symbIoTe addresses a challenging objective to create an interoperable Internet of Things (IoT) ecosystem that will allow for the collaboration of vertical IoT platforms towards the creation of cross-domain applications. Thus, it designs an *interoperable mediation framework* to enable the discovery and sharing of connected devices across existing and future IoT platforms for rapid development of cross-platform IoT applications. symbIoTe allows for *flexible interoperability mechanisms* which can be achieved by introducing an incremental deployment of symbIoTe functionality across the platform’s space, which will in effect influence the level of platform collaboration and cooperation with other platforms within a symbIoTe-enabled IoT ecosystem. *Syntactic and semantic interoperability* represent the essential interoperability mechanisms in the future symbIoTe-enabled ecosystem, while *organizational/enterprise interoperability* has different flavors within symbIoTe (platform federations, dynamic Smart Spaces and roaming IoT devices) to enable platform providers to choose an adequate interoperability model for their business needs.

The document lists the final collection of the symbIoTe system requirements covering the following domains: Application Domain, Cloud Domain, Smart Space and Smart Device. The first two domains (Application and Cloud) interconnect applications with platform-managed devices by a set of services enabling application developers to identify and use devices from different platforms in a uniform way. Devices are exposed to third-parties as services, where the management of devices and associated services, both *sensors and actuators*, as well as access control stays on the platform side. Furthermore, symbIoTe needs to enable organizational interoperability in the Cloud Domain so that platforms can interoperate directly and securely to share/trade devices and associated services at the level of virtualized cloud-based IoT services. Smart Space is a local environment hosting either one or multiple collocated platforms. symbIoTe’s goal is to seamlessly connect, dynamically configure and automatically register devices in a Smart Space while facilitating local interactions between collocated platforms. Furthermore, symbIoTe also relates to the device level with a concept of a Smart Device, which can interact with surrounding visited environments enabling the concept of a roaming IoT device.

symbIoTe offers flexible interoperability mechanisms enabled by an incremental deployment of symbIoTe functionality across the previously mentioned architectural domains. This approach enables platform providers to choose an appropriate level of integration of symbIoTe-specific services within their platforms, which will in effect influence the level of platform collaboration and cooperation with other platforms within a symbIoTe-enabled ecosystem.

This document reports an extensive set of requirements that have been derived from five symbIoTe use cases and have served as input for identifying the symbIoTe software components and associated features. The main task of these components is to facilitate syntactic and semantic interoperability of IoT platforms so that platforms offer an open API

with a mapping of platform information models to the symbIoTe information model[1]. In this document, we include communication diagrams depicting component interaction for syntactic and semantic interoperability. Since security-related requirements play a vital role in symbIoTe, security-related components implementing *Attribute Based Access Control* (ABAC) have also been defined. These components are providing authenticated and authorized access to platform devices. In addition, the Cloud Domain specifies components for platform-to-platform interaction for bartering and trading of devices. The document also reports an initial view on components and envisioned functionality needed for the Smart Space and Smart Device domain. The aim in those two domains is to offer dynamic reconfiguration of devices in environments hosting a number of platforms, and to support roaming devices that can blend with visited environments not operated by their home platforms.

Finally, the document analyzes relevant work in the area of IoT interoperability, with focus on reference architectures by standardization bodies and their mapping onto the symbIoTe architecture, projects with similar goals as symbIoTe, and platforms by symbIoTe partners aiming to become part of the future symbIoTe-enabled IoT ecosystem. We can conclude that the proposed functional architecture and its layered stack with four domains (Application, Cloud, Smart Space and Smart Device domain) is in accordance with the AIOTA reference architecture. It is motivated by the oneM2M architecture, but symbIoTe extends the scope by identifying features which go beyond the oneM2M functional architecture: These are related to platform federations, bartering and trading as well as device roaming.

[1] The symbIoTe information model will be specified in deliverables D2.1 “Semantics for IoT and Cloud Resources” and D2.4 “Revised Semantics for IoT and Cloud Resources”.

2 Introduction

In a world of smart networked devices, wearables, sensors and actuators, transparent and secure access to and usage of the available resources across various Internet of Things (IoT) domains is crucial to satisfy the needs of an increasingly connected society. However, the current IoT ecosystem is fragmented: a series of vertical solutions exists today which, on the one hand, integrates connected objects within local environments using purpose-specific implementations and, on the other hand, connects smart spaces with a back-end cloud hosting often dedicated proprietary software components. The symbIoTe project steps into this landscape to facilitate the creation and management of hierarchical, adaptive and dynamic IoT environments, and to devise an **interoperability framework** across existing and future IoT platforms for seamless networking and rapid cross-platform application development.

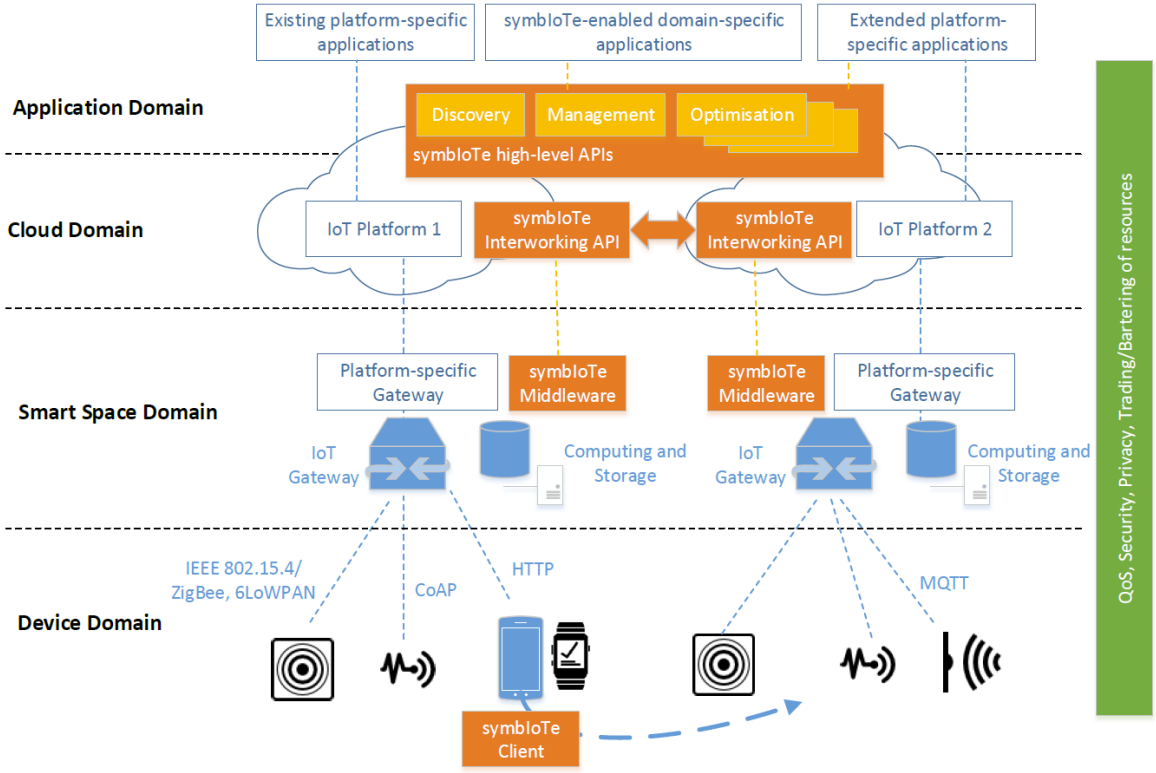

Figure 1 Sketch of the symbIoTe architecture, as proposed in Description of the Action (DoA)

Figure 1 sketches an environment of the future symbIoTe ecosystem which is built around a hierarchical IoT stack and spans over different IoT platforms. We assume that various IoT devices are connected to IoT gateways within Smart Spaces, representing physical environments with deployed “things”, while being operated by one or more collocated IoT platforms. Those platforms provide IoT services locally in Smart Spaces and share the available local resources (connectivity, computing and storage). Smart Spaces and local platform services are connected to platform services running in the cloud (e.g. resource discovery and management, data analytics). Thus, in addition to their local representation, deployed physical things are also mapped to their virtual representations within the cloud, and exposed as IoT services to third parties, e.g., mobile or web applications. The

symbIoTe architecture spans over the following four layered domains, as depicted in Figure 1:

- **Application Domain:** enables the creation and management of cross-platform virtual environments to allow unified view on various platforms and their resources;
- **Cloud Domain:** enables platform interoperability and creation of platform federations or associations between two platforms so that platforms can securely interoperate, collaborate and share resources;
- **Smart Space Domain:** enables dynamic discovery and configuration of resources within local smart environments, even those already connected to different platforms;
- **Smart Device Domain:** relates to heterogeneous IoT devices representing things in smart spaces; IoT devices can be discovered in visited Smart Spaces and use their resources in a controlled way (*device roaming*).

2.1 Purpose of this document

The purpose of deliverable D1.4 “Final Report on System Requirements and Architecture” is to document the **final collection of system requirements** and a **final version of the system’s functional architecture**, based on the results of the completed tasks T1.3 and T1.4. System requirements have been carefully derived from the use case descriptions reported in deliverables D1.1 and D1.3 through an iterative and collective process. Identified system requirements have served as input to create an initial, but also quite comprehensive, list of components, their features, and interfaces for the Application Domain, Cloud Domain, Smart Spaces Domain and Smart Devices Domain. Furthermore, this document reports the symbIoTe security mechanisms which are incorporated into component descriptions and corresponding communication diagrams.

The functional architecture reported in this document is used as input to design and implementation tasks in WP2, WP3 and WP4. In particular, task T2.2 designs and implements components for the Application Domain and Cloud Domain to enable semantic and syntactic interoperability of IoT platforms. Tasks within WP3, focusing on IoT Platform Federation, were running in parallel with T1.4 and used inputs reported in this deliverable to specify in more detail trading and bartering mechanisms (T3.1), security and access scopes (T3.2), and the design of components enabling IoT federations (T3.3). Work within WP4 further elaborates on the Smart Space and Smart Device Domains presented in this document to design and implement Smart Spaces Middleware.

Deliverable D1.4 is an upgrade of deliverable D1.2 “Initial Report on System Requirements and Architecture” that focused on Application and Cloud Domains. In this deliverable, component features and interaction descriptions between the aforementioned domains are further refined, while the requirements and architecture for the Smart Space and Smart Device Domain are elaborated.

2.2 Terminology and definitions

IoT-related terms and concepts used in this document are based on the AIOTI Domain Model [3] which is derived from the IoT-A Domain Model [5]. However, we do not use the term **resource** as it is defined in the IoT-A Domain Model, but rather base our definition on the one proposed by oneM2M [6].

IoT-related terms:

- **Thing**: represents a physical entity in the physical world with which a generic user interacts indirectly via an IoT Service. It usually has sensing/actuation and communication capabilities, as well as data capture, storage and processing capabilities.
- **IoT Device:** interacts with a thing and exposes the capabilities of the actual physical entity. Typical devices are sensors, actuators, tags or gateways (referred to as intermediary devices in IoT-A).
- **Virtual Entity**: represents a thing (physical entity) in the digital world.
- **IoT Service**: is associated with a virtual entity and can interact with the corresponding thing via its IoT device.
- **Composite IoT Service**: is associated to one or a group of virtual entities managed by a single or multiple IoT platforms, but appears to the outside as a single IoT service.
- **Resource**: is a uniquely addressable entity in symbIoTe architecture and, as a generic term, may refer to IoT devices, virtual entities, network equipment, computational resources and associated server-side functions (e.g., data stream processing). This definition is on purpose highly generic and abstract to allow its unified, recursive use across all layers of the envisioned symbIoTe stack.
- **System**: the set of APIs, interfaces, services and, in general, all components of the software realization of the symbIoTe architecture.

Stakeholders:

- **IoT Platform Provider**: offers IoT services managed by an IoT platform (reside within symbIoTe Cloud Domain).
- **Application Developer**: build IoT applications based on the IoT services exposed by various IoT platforms (reside at the symbIoTe Application Domain).
- **End User**: an individual user of a symbIoTe-supported IoT application.
- **Infrastructure Provider**: physically deploys the necessary hardware and software infrastructure within smart spaces.
- **Prosumer**: is a stakeholder (e.g. platform or end user) which at the same time produces/provides resources/goods but also consumes resources/goods provided by other producers or prosumers.
- **Consumer:** Does not provide any resources/goods. Can participate only in trading transactions which allow him/her to gain access to resources registered within the symbIoTe Core Services.
- **Producer:** Provides resources/goods within the symbIoTe ecosystem. He/she can engage in trading and bartering transactions.

Security-related definitions:

- **Public IoT service**: IoT service without access restrictions.
- **Restricted IoT service**: IoT service to which only specific users have access, platforms grant access rights to their IoT services.
- **IoT device metadata**: resource description maintained within symbIoTe core to perform the search functionality; access to resource metadata may also be restricted to a selected group of users.
- **IoT service access policy**: rules for accessing an IoT service defined by a platform per each IoT service or a set of services, can be used to filter out search results to which an app/enabler does not have access rights.

- **Token**: represents a digital object used as a container for security-related information. It can be used for authentication and/or authorization purposes. In general, it appears as a list of elements. Each element contains an assertion that further specifies properties assigned to the owner of the token or to the token itself (i.e., issuer of the token, issuing time, expiration date, subject and so on). Moreover, a token contains one or more attributes, assigned to the owner of the token itself. Finally, it also contains an element (generally stored at the end) that certifies its authenticity and integrity, namely a sign.
- **Attribute**: it is a specific property assigned to an entity, i.e. role, permission or feature, which can be assigned after a successful authentication procedure. An entity in the system is characterized by a set of attributes, potentially assigned by different platforms at different time instants. Tokens are composed of one or more attributes and are used when accessing resources in the system.
- **Multi-Factor Authentication:** Users/devices/applications are authenticated by presenting two or more different type of evidence factors. Types can be: knowledge (something you know), possession (something you have) and inherence (something you are) or location.

symbIoTe-specific definitions:

- **symbIoTe Compliance Level** (CL): four interoperability aspects covered within the symbIoTe project.
- **Level-1 Compliant IoT Platform** (L1 Platform): offers an open symbIoTe-defined platform interface within the Cloud Domain; platform resources and IoT-related services are searchable within the symbIoTe Core Services.
- **Level-2 Compliant IoT Platform** (L2 Platform): implements functionality needed for platform federations and direct platform to platform interworking for bartering/trading of resources.
- **Level-3 Compliant IoT Platform** (L3 Platform): supports dynamic smart spaces.
- **Level-4 Compliant IoT Platform** (L4 Platform): supports device roaming in visited domains; a Smart Device can use services in a visited Smart Space.
- **IoT Platform Federation:** an association of two or more platforms enabling secure interoperation, collaboration and sharing of resources.
- **Smart Space**: physical environment (e.g. residence, campus, vessel, etc.) with deployed things where one or more IoT platforms provide IoT services.
- **Smart Device**: a device that can directly interact with a Smart Space.
- **symbIoTe Core Services**: services implemented by symbIoTe components at the Application Domain which enable the interaction between third-party applications and L1 platforms.
- **Interworking Interface**: symbIoTe-defined interface which offers platform resources as IoT Services in the Cloud Domain.
- **Core Information Model** (CIM): central information model of symbIoTe. It defines all the terms (classes and their relations) that symbIoTe components can understand. Additionally, it serves as a shared vocabulary between all symbIoTe-Compliant IoT Platforms.
- **Platform specific Information Model** (PIM): custom extension of CIM defined by symbIoTe Compliant IoT Platforms. It contains additional classes and predicates that are necessary for describing the data provided by that specific platform.
- **Best practice Information Model** (BIM): a special kind of PIM designed to cover the domains of the use cases within symbIoTe project.

Performance indicators:

- **Registration response time**: the time required for the completion of the IoT service registration process.
- **Search response time**: the time required for the system to return the results of IoT service search.
- **IoT service access latency**: the time required for the first use of an IoT service, including initial authentication/authorization processes; it does not include access to data (or instruction to an actuator).
- **Domain Enablers**: domain-specific back-end services which provide added value features on top of IoT platforms and their resources. They are placed within the symbIoTe Application Domain with a goal to ease the process of cross-platform and domain-specific application development, and even cross-domain application development (specifically for mobile and web applications).

Bartering and Trading related definitions:

- **Bartering**: refers to economic mechanisms whereby prosumers get access to desired external resources and grant access to own resources requested by other prosumers without monetary compensation.
- **Trading:** refers to economic mechanisms (e.g. auctions, direct negotiations) whereby producers come to an agreement with consumers about providing goods/resources for which they are paid.
- **B2B**: platform to platform, prosumers (have IoT services to share).
- **B2C**: platform to consumer/application that don't have IoT services to share (only buy access).
- **Forward Auction:** The *producer* initiates the auction by creating an *offer*. Other producers or consumers can start bidding. In this auction the price keeps increasing. The winner will be the entity which bids the highest price.
- **Reverse Auction:** The *consumer* (or producer) initiates the auction by defining a *need. Producers w*ho wish to fulfil the need can start bidding. In this auction the price keeps going down. The winner will be the producer which bids the lowest price.
- **Vouchers:** digital objects comprised of:
    1. The Service Level Agreement (SLA) which contains the available commodity/resources/goods (e.g., air quality sensors in Vienna, access for one week, beginning with a date, etc.).
    2. The authorization token with access rights mapped in accordance to the SLA.
    3. The issuers desired SLA (e.g.: willing to exchange own voucher for a voucher containing humidity sensors and other attributes).
    4. A validity date (expiration date).
    5. A unique Identifier.

2.3 Relation to other deliverables

System requirements and architecture reported in this document have evolved from deliverable D1.2 “Initial Report on System Requirements and Architecture.” The process of defining system requirements and architecture followed a two-step approach: Initial requirements and architecture were created in the first step with focus on Application Domain and Cloud Domain (reported in the deliverable D1.2). The second step reported in

this deliverable D1.4 “Final Report on System Requirements and Architecture” focuses on Smart Space and Smart Device requirements and architecture. Additionally, D1.4 documents updates of the Application Domain, which has been in the implementation phase at the time when the report is published, and a more detailed specification of the Cloud Domain.

This deliverable **presents the functional architecture of symbIoTe**. The symbIoTe Information model, which has been developed in T2.1 in parallel with tasks T1.3 and T1.4, is presented in D2.4 (M18) “Revised Semantics for IoT and Cloud Resources.” Domain-specific enablers mentioned in this deliverable have been elaborated and specified within T2.3, and reported in D2.3 (M12) “Report on symbIoTe Domain-Specific Enablers and Tools.” Bartering and trading mechanisms, as well as security aspects specified in this document were documented in more detail within D3.1 (M11) “Resource Trading Mechanisms and Access Scopes”. Smart Spaces and Smart Devices presented within this deliverable have been investigated through activities in WP4 (Dynamic Smart Spaces) and reported in D4.1 (M12) “symbIoTe Middleware Tools, Protocols and Core Mechanisms.” This deliverable documents an updated version of Smart Spaces and Smart Devices architecture from D4.1.

Implementation details of the symbIoTe system based on the system requirements and architecture are presented in separate deliverables. D2.5 (M18) “Final symbIoTe Virtual IoT Environment Implementation” describes symbIoTe Core Services and the components for enabling IoT platforms to become Level-1 Compliant. D2.6 (M21) “symbIoTe Domain-Specific Enablers and Tools” will document implementation details of generic Enabler components, and specific Enabler components for selected use cases. D3.2 (M22) “Resource Trading, Security and Federation Mechanisms” will present the design and implementation of symbIoTe security mechanisms as well as other mechanisms enabling IoT platforms to become Level-2 Compliant. D4.2 (M23) “symbIoTe Middleware Implementation” will document details of the Smart Space Middleware design and implementation enabling IoT platforms to become Level-3 and Level-4 Compliant.

2.4 Document structure

Section 2 presents the purpose of this document and its relation to other deliverables within the project. Section 0 elaborates on symbIoTe goals and challenges, and provides details about symbIoTe domains and their relation to the four symbIoTe-specific interoperability aspects referred to as *Compliance Levels* (CLs). System requirements are presented in Section 4, while system architecture with all respective components, entities and interfaces is introduced in Section 5. Section 6 presents a state of the art overview, with focus on reference architectures by standardization bodies, projects with goals similar to symbIoTe, and platforms by symbIoTe partners aiming to become a part of the future symbIoTe-enabled IoT ecosystem.

3 The symbIoTe Vision

The symbIoTe vision is to participate in the IoT ecosystem of the future by providing an interoperable mediation framework integrating heterogonous IoT platforms that will allow for the rise of next-generation cross-platform IoT applications as well as local interactions between smart devices managed by different platforms. Considering the fact that there are many different IoT platforms already on the market, focusing on different domains while tackling various requirements and business goals, symbIoTe proposes a flexible interoperability concept (translated into four interoperability-related CLs) so that platform providers can choose an appropriate one to match their business needs.

3.1 symbIoTe's goals and challenges

In a highly fragmented IoT ecosystem which is faced with an increasing number of new IoT platforms on the market[2], there is a need for a viable interoperability framework enabling platform collaboration and cooperation. This would allow for a new generation of cross-domain IoT services and applications to be built on top of various platforms managing huge numbers of heterogeneous devices. symbIoTe aims to devise an interoperability framework which will provide an abstraction layer for a "unified view" on various platforms and their resources so that platform resources are transparent to application designers and developers. The IoT resources will be organized in a hierarchical manner with Smart Spaces interacting with an IoT platform back-end running in the cloud environment. Dynamic discovery, reconfiguration and migration of IoT devices will be supported by Smart Spaces to provide dynamicity and adaptability. In addition, symbIoTe also chooses the challenging task of implementing IoT platform federations so that IoT platforms can securely interoperate, collaborate and share resources for mutual benefit. Moreover, symbIoTe envisions the implementation of use case-specific high-level APIs (enablers), which will further foster a simplified IoT application and service development process over interworking IoT platforms.

Finally, satisfying security requirements also occupies an important place in symbIoTe's design principles. symbIoTe designs and develops mechanisms assuring secure IoT platform interworking, to offer resource access schemes based on security scopes and to devise an identity management solution for IoT resources which integrates security features and avoids potential attacks. To this end, specialized security components are identified in various architectural domains.

Main technological challenges:

- **Unified and secure access to physical and virtualized IoT devices**: such access is required for the next-generation of cross-platform and cross-domain applications.
- **Device discovery across various IoT platforms**: symbIoTe needs to offer search mechanisms to efficiently identify devices across platforms that are adequate and accessible to third parties.
- **Security**: access scopes and identity management represent the key requirements for authenticated and authorized access to various resources across platforms.

[2] Beecham Research estimates around 300 IoT platforms to be on the market by the end of 2016 [25].

- **Platform federation for collaborative sensing/actuation tasks**: platforms should be enabled to securely interoperate for the trading/sharing of resources as well as to control the terms under which roaming devices are allowed to use resources in visited domains.
- **Hierarchical, adaptive and dynamic IoT environments**: it should be possible to dynamically reconfigure devices in Smart Spaces so that they connect to different gateways, or even to different platforms collocated in the same environments.
- **Seamless roaming of Smart Devices across Smart Spaces**: roaming devices should be enabled to use resources from a local surrounding environment and in accordance with Service Level Agreements (SLAs) between a platform managing the visited domain and the platform that operates the roaming device.

3.2 Architecture overview

The symbIoTe approach is built around a layered IoT stack connecting various devices (sensors, actuators and IoT gateways) within Smart Spaces with the Cloud. Smart Spaces share the available local resources (connectivity, computing and storage), while platform services running in the Cloud enable IoT Platform Federations and open up the Interworking Interface shown in Figure 2 to third parties. The architecture comprises four layered domains, 1) Application Domain, 2) Cloud Domain, 3) Smart Space Domain and 4) Device Domain, as depicted in Figure 2. Hereafter we list the main functional objectives for each of these domains:

1. **Application Domain (APP):** enables platforms to register IoT resources which they want to advertise and make accessible via symbIoTe to third parties, while symbIoTe Core Services can search for adequate resources across platforms. It also hosts domain-specific back-end services (Domain Enablers) which are designed to ease the process of cross-platform and domain-specific application development (specifically for mobile and web applications).
2. **Cloud Domain (CLD):** provides a uniform and authenticated access to virtualized IoT devices exposed by platforms to third parties through an open API (Interworking Interface). In addition, it builds services for IoT Platform Federations, enabling close platform collaboration in accordance with platform-specific business rules.
3. **Smart Space Domain (SSP):** provides services for the discovery and registration of new IoT devices in dynamic local smart spaces, dynamic configuration of devices in accordance with predefined policies in those environments, and uniform interfaces for devices available in smart spaces.
4. **Smart Device Domain (SDEV):** relates to smart devices and their roaming capabilities. We assume that devices have the capabilities to blend with a surrounding smart space while they are on the move. In other words, smart devices can interact with devices in a visited smart space managed by a visited platform, in accordance with predefined access policies.

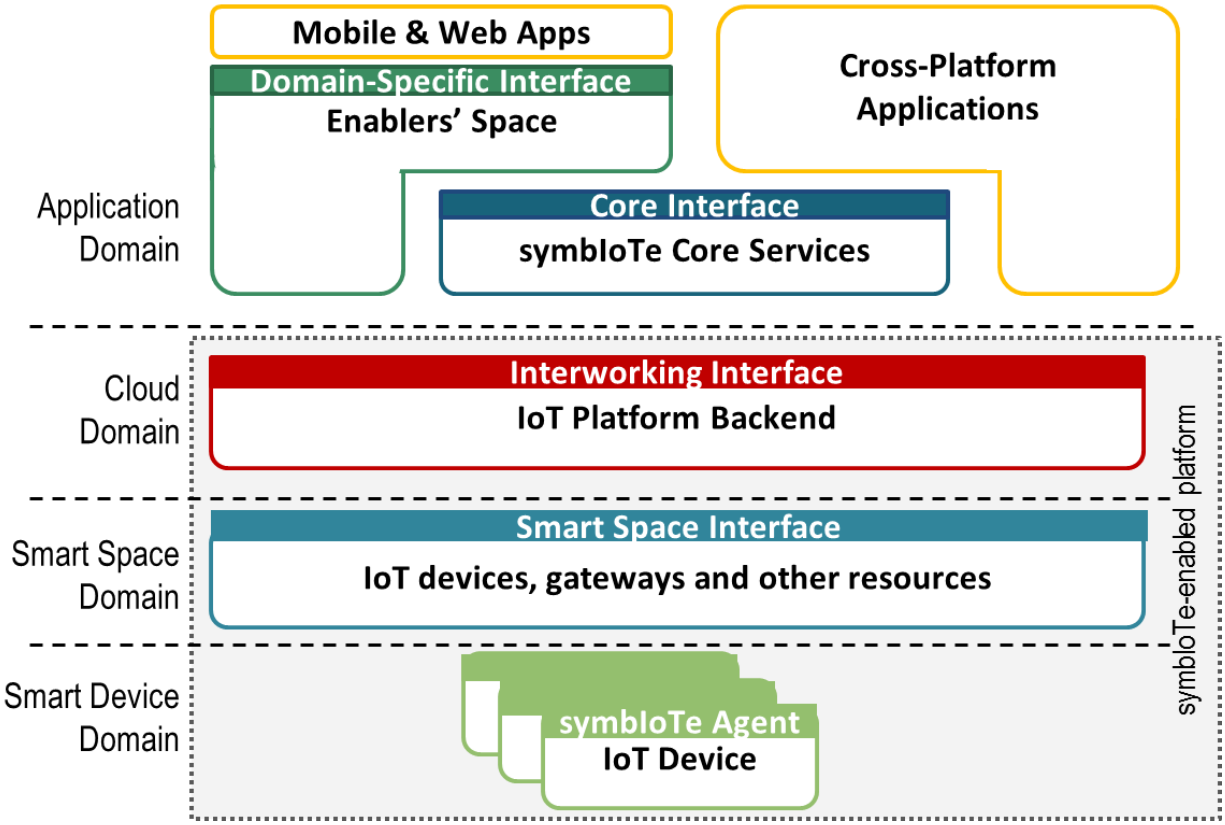

Figure 2 The symbIoTe high-level architecture

APP is designed to offer a unified view on different platforms to create an environment enabling **a new generation of cross-platform and cross-domain IoT applications**. This is achieved by the symbIoTe Interworking Interface providing a uniform view on platform backend services and Core Services enabling applications to search for IoT devices across platforms. It relies on a common semantic representation of IoT resources (services or devices) which uses an expressive yet minimalistic information model[3] (symbIoTe Core Information Model, CIM). Note that the Core Services store and manage only IoT resource descriptions (i.e. resource metadata), while the access to those resources (e.g., sensor data and actuation primitives) is provided by the underlying platforms through the Interworking Interface. Thus, the symbIoTe Core Services are in close interaction and collaboration with the services provided within the Cloud Domain which offer the actual access to virtualized IoT resources.

In addition to Core Services, we also envision domain-specific enablers to be placed in APP. Enablers offer value-added services on top of IoT services which are managed and offered by "native" IoT platforms. For example, an enabler can gather and process all air quality related data for a certain country and provide data analytics on top of the data set acquired from various sources and administrative domains. In principle, an enabler can be regarded as a virtual IoT platform since it does not possess the actual hardware, but rather offers value-added services on top of the IoT services and devices being accessed through symbIoTe Core Services. For symbIoTe Core Services an enabler thus plays a dual role: 1) it is an application using symbIoTe Core Services to find adequate IoT services, and 2) it acts as another IoT platforms offering domain-specific IoT services to applications.

[3] The symbIoTe information models are specified in deliverable D2.4 "Revised Semantics for IoT and Cloud Resources".

CLD hosts components that enable a unified and secure access to underlying platform resources as well as closer collaboration between platforms, i.e., platform federations. The symbIoTe Interworking Interface is defined and implemented to expose platform resources to third parties, which have previously been registered with the symbIoTe Core Services. CLD services also implement specific functionality for the exchange of resource metadata between two collaborating IoT platforms, e.g., for bartering and trading purposes in accordance with predefined SLAs.

SSP comprises various IoT devices, IoT gateways as well as local computing and storage resources available within, e.g., a home environment or campus building. We assume that IoT platform-specific gateways are setup in a SSP. To enable dynamic sensor discovery and configuration in SSP as well as dynamic sharing of the wireless medium, symbIoTe adds a new software component, symbIoTe middleware, to SSP, at the gateway level. The symbIoTe middleware exposes a standardized API for resource discovery and configuration of devices within a SSP, and implements a sensor-discovery protocol for a simplified integration of sensors with platforms hosted in particular Smart Space Domains. After the initial interaction with the symbIoTe middleware, an IoT device is connected to and configured with the platform gateway serving the domain. Note that the device may be located either in a home or visited space. This protocol will also enable that an IoT device entering a visited space becomes part of a new SSP, enabling thus device roaming. An SLA needs to be in place between the platforms serving home and visited spaces, which also specifies services exposed to the roaming device in a visited space.

SDEV spans over heterogeneous devices which may use proprietary link layer protocols, or ZigBee and 6LoWPAN, while it can be expected that future IoT devices will also support application-layer protocols such as HTTP, CoAP and MQTT. Devices should be capable to dynamically blend with a surrounding space and get discovered by the symbIoTe middleware which performs the initial "introduction" of devices within a Smart Space. Smart Devices can self-organize and can be configured on the fly to be integrated with different IoT platforms hosted within the Smart Space, preventing thus the lock-in of customers to a specific IoT platform and IoT provider. We envision that device-specific symbIoTe clients will be running on, e.g., smartphones, to realize these features.

3.3 Compliance Levels (CLs)

symbIoTe allows for flexible interoperability mechanisms which can be achieved by introducing an incremental deployment of symbIoTe functionality across the listed architectural domains. This approach will enable platform providers to choose an appropriate level of integration of symbIoTe-specific services within their platforms, which will in effect influence the level of platform collaboration and cooperation with other platforms within a symbIoTe-enabled ecosystem. For example, a platform may only choose to expose its Interworking Interface and selected IoT services to third parties in order to advertise them by using the symbIoTe Core Services, or it may opt for a closer collaboration with another platform by forming a platform federation. Platform federations require additional symbIoTe components to be included and integrated within a platform space in CLD.

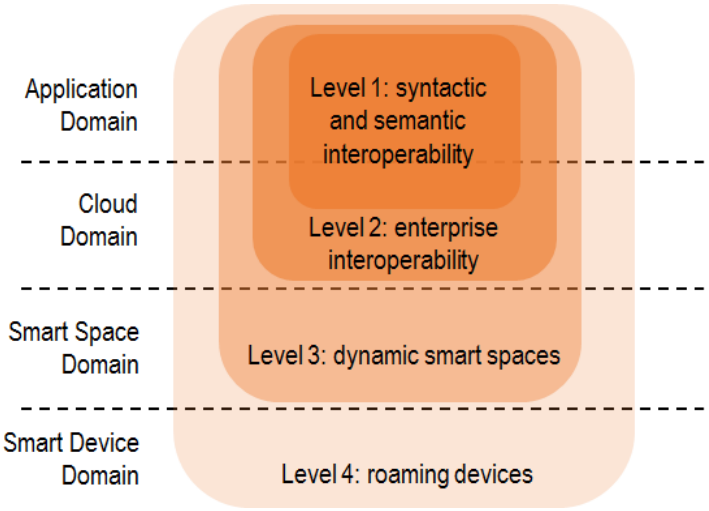

Figure 3 symbIoTe Compliance Levels (CLs)

We define four different CLs for IoT platforms, as depicted in Figure 3. They reflect different interoperability modes a platform can chose to support, affecting thus the functionality and corresponding symbIoTe components which have to be integrated on the platform side within different domains:

- **Level-1 symbIoTe Compliant Platform (L1 Platform)**: This is a "lightweight" symbIoTe CL since a platform opens up only its Interworking Interface to third parties in order to advertise and offer its virtualized resources through the symbIoTe Core Services. It enables the syntactic and semantic interoperability of IoT platforms in a symbIoTe ecosystem, and affects only APP and CLD.
- **Level-2 symbIoTe Compliant Platform (L2 Platform)**: This level assumes that platforms federate, which requires additional functionality to be included in CLD, for example for sharing/bartering of devices. The functionality provided at this level enables the so-called enterprise interoperability.
- **Level-3 symbIoTe Compliant Platform (L3 Platform)**: This CL assumes that platforms integrate symbIoTe components within their smart spaces to simplify the integration and dynamic reconfiguration of IoT devices within local spaces.
- **Level-4 symbIoTe Compliant Platform (L4 Platform)**: This level offers support for device roaming and can enable the interaction of smart objects with visited smart spaces. A prerequisite for this level is that a platform is already Level-1, 2 & 3 Compliant, so that smart spaces can discover new visiting devices and integrate them (e.g., grant access to certain local resources) in accordance with SLAs between platforms. Those platforms should thus be in a federation (Level 2), while smart spaces need the functionality for dynamic reconfiguration (Level 3).

L1 Compliance (CL1) can be directly mapped to semantic and syntactic interoperability, as identified in the ETSI Whitepaper [1], and subsequently adopted by IERC [2]. L2, L3 and L4 Platforms can clearly be categorized as systems supporting organizational interoperability. symbIoTe proposes here an original approach with finer granularity of organizational interoperability by placing specific interoperability concepts in the CLD for CL2, in the SSP for CL3 as well as in both SSP and SDEV for CL4. In particular, L2 Platforms form platform federations, L3 Platforms support dynamic and reconfigurable Smart Spaces, while L4 Platforms support roaming of Smart Devices which can use services in visited smart spaces. To achieve CL2, a platform should first adhere to CL1,

while an L4 Platform requires a full symbIoTe framework (i.e., an L4 Platform is also L1, L2 and L3 Compliant).

CL1 relates to services placed in two domains, APP and CLD. An IoT platform becoming part of the symbIoTe ecosystem needs to integrate the symbIoTe Interworking Interface with its existing components, e.g., with services exposing sensor-generated data or actuation primitives. This facilitates open and uniform access to virtualized resources across platforms. Note that a platform chooses which resources it wants to register and make discoverable via the symbIoTe Core Services. In addition, the platform issues access tokens to third parties and keeps control over access to its resources. symbIoTe plays here a mediation role and uses distributed and decoupled mechanisms for authentication and authorization, namely the Attribute Based Access Control (ABAC) with token-based authorization (more information is provided in Section 5.4).

Figure 4 shows the benefits of CL1 by an example depicting two platforms A and B using the symbIoTe Core Services. When an application searches for resources and identifies adequate ones, it accesses RESTful services offered by the two platforms through the Interworking Interface. In other words, cross-platform applications i) use the symbIoTe Core Services to find adequate resources across platforms and ii) access, integrate and use those resources through a uniform and open interface. Note that symbIoTe stores only resource metadata within the Core Services to provide adequate search mechanisms, while cross-platform applications access and use resources directly at the platform side.

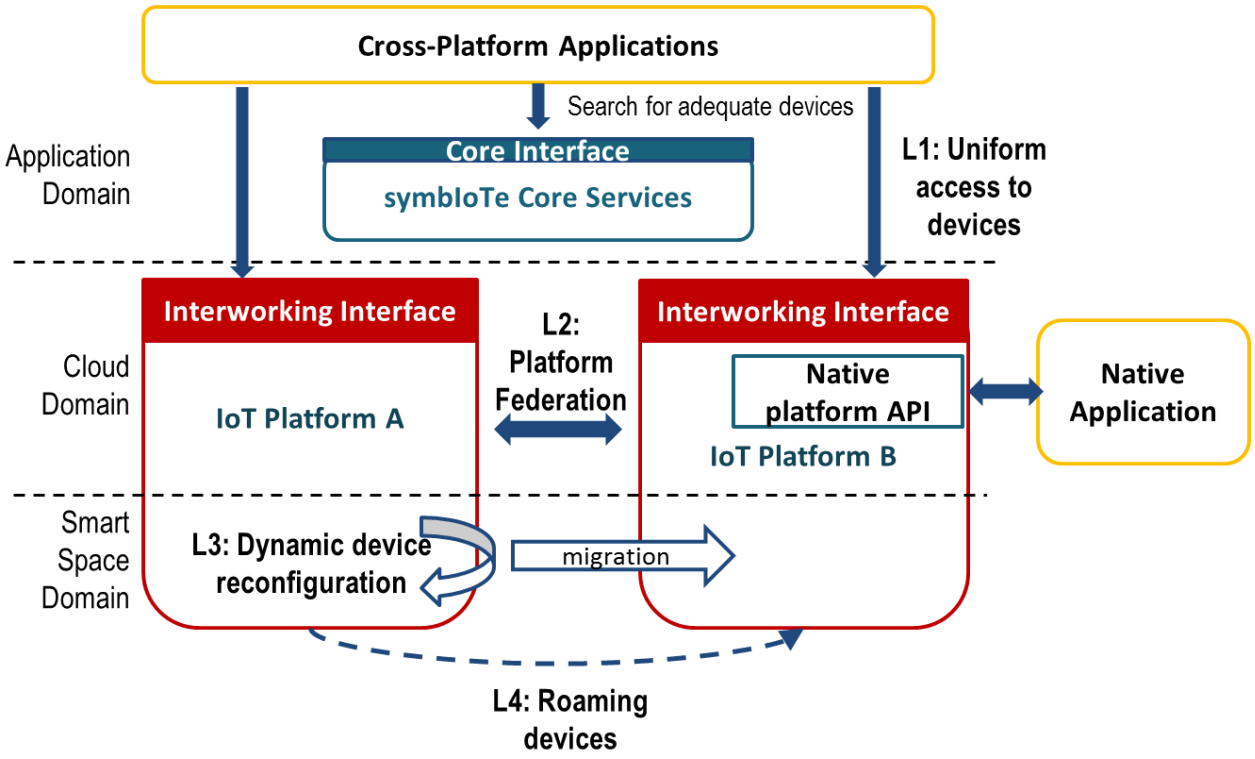

Figure 4 Illustrating symbIoTe CLs

CL2 involves components placed both in APP and CLD, but requires a significant extension of an existing platform deployment to enable a closer collaboration between platforms by adhering to specified SLAs. Figure 4 also illustrates an example federation where it is possible to expose certain IoT resources from platform A within the space of platform B. This creates an opportunity that an existing application (*native application*)

expands the set of resources available within platform B, since they appear as native services to an existing application built exclusively for platform B.

CL3 and CL4 mainly affect platform software which is deployed within a SSP, and may also require specific software at the IoT devices level. Features similar to the ones appearing in APP and CLD are needed within SSP, but with quite a different and reduced scope relating to local space resources. Since a number of platforms can occupy the same SSP, CL3 refers to dynamic reconfiguration of devices within a SSP, so that a device is reconfigured on the fly to become part another platform, thus preventing vendor lock-in. CL4 relates to interoperability at the SDEV level. An example is when a device registered in platform A visits an environment operated by platform B. The device can use the surrounding infrastructure operated by platform B in accordance with an SLA between the two platforms.

4 System Requirements

4.1 Framework

The specification of system requirements in symbIoTe aims at driving the design of the symbIoTe architecture, based on a thorough assessment of the considered use cases in the context of the identified project goals (see Section 3). In this effort, special attention has been paid in capturing the requirements posed by the various stakeholders in the symbIoTe landscape i.e., IoT Platform Providers, Application Developers, End Users, Infrastructure Providers, Prosumer. Towards these ends, the specification of the requirements has been structured according to the identified CLs thus resulting in the Application Domain, Cloud Domain and Smart Space and Smart Device Domain requirements. Paying particular attention to security, the project has separately focused on the specification of the corresponding requirements, by forming a team of security experts within the consortium.

The initial report on system requirements in D1.2 focused on the specification of the requirements at CL1 and CL2, with a preliminary investigation of the requirements for CL3 and CL4. Following the submission of D1.2, the consortium further focused on the finalization of requirements specification for CL3 and CL4 as well, resulting in additional requirements. At the same time, the concept of Enablers was further investigated, leading to the specification of additional CL1 requirements. Moreover, particular attention was paid to the feedback we received on D1.2, trying to improve the specification of the requirements. In this context, we reviewed existing and new requirements in order to elaborate on the selection of the importance level, wherever this was not assessed as obvious. Additionally, we attempted to identify potential risks emerging from the specified requirements, related to the uptake of the envisioned business models. Finally, we further hereby highlight the system requirements strongly related to business model-related aspects, as primarily introduced by the Bartering & Trading concepts in symbIoTe.

Requirements presented in this report document the final set of system requirements devised within symbIoTe project. In order to structure the specification of the system requirements, we have identified the following set of requirement attributes.

Compliance Level (CL): Following the structure of the symbIoTe stack, each requirement relates to one or more CLs, namely: CL1 (Application), CL2 (Cloud), CL3 (Smart Space), CL4 (Smart Device). In several cases, requirements may apply to more than one CLs. These are reported within the set of the highest CLs they apply to.

Type: Two types of requirements are initially defined:

- *Functional*: requirements describing the behavior of the symbIoTe system i.e., what the symbIoTe architecture should do.
- *Non-Functional*: requirements describing properties of the symbIoTe architecture and system operation.

Category: A set of thematic categories is defined with the purpose of assisting grouping requirements of similar nature and later guide the architectural design of the symbIoTe system. Each requirement may fall into more than one categories. Namely:

- *Interface*: refers to the methods employed to enable the interaction between different entities in the symbIoTe architecture, within and across domains, as well as between the symbIoTe system and end users and/or clients.

- *Monitoring*: refers to the collection of information describing the current and past status of resources in symbIoTe.
- *Management*: refers to all types of functional and non-functional requirements related to the handling/control of resources in symbIoTe. This does not include management aspects within each existing symbIoTe-enabled IoT platform.
- *Performance*: refers to non-functional requirements including Key Performance Indicators (KPIs) that will assist in establishing an evaluation framework for symbIoTe but also take into account potential performance bottleneck during implementation. This shall also include scalability aspects i.e., linking KPIs with (work)loads expected.
- *Security*: encompasses all security aspects of the symbIoTe architecture including authentication, authorization, privacy, etc.

Importance: Each requirement is characterized by its importance level with respect to its fulfilment by the symbIoTe architecture and system. The level of each requirement is expressed within the corresponding description text (see next) using the appropriate terminology. Following Best Current Practice to Indicate Requirement Levels [26] we consider the following levels:

- *MUST (SHALL):* this is an absolute requirement i.e., it is mandatory for the symbIoTe architecture and system to conform to this requirement.
- *SHOULD (RECOMMENDED):* there may exist valid reasons within particular circumstances to ignore this requirement.
- *MAY (OPTIONAL):* a requirement for a feature or a property of the symbIoTe architecture that presents low priority within the project and may or may not be fulfilled, subject to time constraints. Usually such features are selected by different vendors subject to their market positioning.

Note of Importance: Following the reviewers' feedback on the initial requirements reporting (D1.2) we elaborate on the grounds a certain Importance Level was selected on, wherever this is considered not straightforward.

Potential barrier for Uptake: Similarly, trying to capture potential implications of system requirement on the uptake of the business models, we augment the requirement specification with a related note, where applicable.

Use case: We report the use cases each system requirement applies to. These are indicated via the following indexes:

1. Smart Residence,
2. Edu Campus,
3. Smart Stadium,
4. Smart Mobility & Ecological Routing, and
5. Smart Yachting.

In some cases a requirement may appear to apply to none of the selected use cases; such requirements are specified for the broader, targeted environment of symbIoTe where additional use cases are envisioned to be supported.

4.2 Methodology

Based on the above framework, the specification of the system requirements followed an iterative process whose purpose was to derive the key requirements across the various CLs. In this process, special attention was paid to finding the common ground across the involved use cases, instead of merely identifying the key needs of a particular example case, so as to pave the way for the support of additional use cases by the symbIoTe system, not currently considered within the project.

Starting from CL1, the iterative process included the following steps:

***Step 1**: Revisit the previous level's requirements within the current CL (if applicable)*

The purpose of this step is to identify requirements that pertain across CLs. Such requirements bear the potential of leading to more efficient architectural design that identifies key functional components across the considered domains, further promising a modular design. It is important to note that this step does not only aim at an optimized modular design, but actually further realizes a continuous requirement assessment process as the project progresses through the various CLs. The selection of the Domain attribute for each requirement therefore constitutes the result of careful consideration of the specificities of each CL. This step is carried out asynchronously with the help of the Confluence collaboration tool.

***Step 2**: Introduce domain specific requirements*

The introduction of domain specific requirements starts with the preliminary input from the leader of the corresponding project task, with the purpose of identifying key areas that should be explored. This is followed by a more elaborate input from the partners leading the considered use cases. Upon the completion of this stage, all use case owners, along with all other partners, inspect the derived requirements providing additional input in the form of:

- Additional requirements;
- Assessment on whether a requirement derived by one use case also pertains to some other, this also includes comments on the generality of the introduced requirements;
- Any other comment, including comments regarding the precise specification of the intended meaning;
- This step is carried out asynchronously with the help of the Confluence collaboration tool; at this stage the Task Leader of T1.3 consolidated all comments and identified grey areas to be discussed.

***Step 3**: Finalize requirements*

This last iteration step aims at finalizing all requirement attributes, ensuring the description is precise, the associated set of use cases has been correctly identified and the importance level within the overall project efforts has been correctly and realistically specified. This step is carried out with a conference call, in which each individual requirement is assessed and potential disputes are resolved. Due to the different nature of the various types of requirements, the above iterative process is followed in three parallel instances, namely for:

- Functional requirements,
- Non-functional requirements, and
- Security requirements.

4.3 Specified requirements

Table 1 below lists the final set of specified requirements for the symbIoTe system, each appropriately annotated with its attributes values. Table 2 subsequently presents the security system requirements as specified by the symbIoTe security team.

Table 1: System requirements

Index	CL	Type	Category	Importance	Note on importance	Potential barrier for uptake	Description	Use Cases
1	1, 2	Functional	Interface	MUST	The registered information is absolutely necessary for underlying resources to be discovered and made available. IoT platforms must have control of what they expose to symbIoTe which is what they offer to applications and other platforms.	Application developers might be reluctant to use symbIoTe supported IoT platforms if unregister events result in application disruption.	IoT platform providers MUST be enabled to register the available (composite) IoT services to the symbIoTe system. The system MUST allow IoT platform operators to update and revoke (unregister) their registrations.	1, 2, 3, 4, 5
2	1	Functional	Interface	MUST	Obvious. Expressing a major symbIoTe concept.	IoT services/platforms might refrain from registering if search results, and especially the corresponding ranking, are not based on objective metrics.	The system MUST expose the available (composite) IoT services to application developers and other IoT platforms. Directory listings and text search are examples of potential interfaces to application developers and platform providers.	1, 2, 3, 4, 5
3	1, 2	(Non-)Functional	Interface	MUST	Fundamental in achieving interoperability.		The system MUST support a common information model for the description of available IoT services across IoT platforms.	1, 2, 3, 4, 5
4	1	Non-Functional	Interface	MUST	Important for interoperability.		IoT services MUSTappear to application developers in a homogeneous manner i.e., the interface for application developers	1, 2, 3, 4, 5

							should not differentiate across IoT platforms. Data source/identity shall be exposed to application developers.	
5	1, 2	Functional	Monitoring	MUST	Necessary so as to assure availability of advertised/exposed resources.		The system MUST monitor the availability of the IoT services registered by IoT platform operators.	1, 2, 3, 4, 5
6	1, 2	Functional	Monitoring	SHOULD	Useful in assessing the expected availability and/or quality of the exposed service. Some services may not support this though.	IoT platforms may not be willing to expose information on the utilization of their overall resources. This is critical information that reveals operational details of the platforms.	The system SHOULD monitor the load on the registered IoT services. Related information can be directly retrieved by IoT platforms (if supported). Additionally, the system can keep track of the IoT services assigned to applications/enablers during the mediation process e.g., when an application developer has identified, requested and has been granted access to IoT services for the intended application. The retrieved information can be used to estimate service load, service popularity (useful for ranking).	2, 3, 4
7	1, 2	Functional	Monitoring	MAY			The system MAY perform functional performance tests on registered IoT services.	2, 3, 4
8	1, 2	Functional	Management	MUST	Service availability is at risk of not satisfied.	How can this be guaranteed? Is such capability (of underlying platforms) a common feature? Could it possibly raise a barrier in registering with symbIoTe?	The system MUST not grant access to IoT resources and services to applications or other IoT platforms if they appear not to be available (at the time of allocation). This requirement also applies to IoT service types with exclusive access rights (e.g., actuators).	2, 3
9	1, 2	Functional	Management	MUST	Different from SHOULD as exclusiveness can be a vital requirement in these cases		The system MUST support the exclusive use of IoT services for underlying IoT platforms that support such feature.	2, 3, 4

					that is supported by the platforms e.g., for actuators.			
10	1, 2	Non-Functional	Interface	MUST	Important for interoperability.		The information from IoT services and IoT devices MUST have the units in which the data is described associated to standard unit of the common information model (meters, kg, etc.). The encoding of units should adhere to a standard (e.g. UCUM).	1, 2, 3, 4, 5
11	1, 2	Functional	Interface	MUST			The common information model MUST support geo-reference information.	1, 2, 3, 4, 5
12	1, 2	Functional	Management	MUST	Some services may not support this.		The system MUST enable the control of access to the advertised IoT services according to fine-grained authorization policies and for reasons related to local legislation, security issues, etc., if the underlying IoT platforms support it.	1, 2, 3, 4, 5
13	1, 2	Non-Functional	Interface	SHOULD	Important for interoperability.		The information model of the system SHOULD comply to standardized ontologies where possible and SHOULD try to be compatible to the data model of the other IoT-EPI projects. An example ontology here is the Semantic Sensor Network Ontology (SSNO).	1, 2, 3, 4, 5
14	1, 2	Functional	Interface	MUST	Important for interoperability.		The information model of an IoT platform registering its IoT services to symbIoTe MUST be aligned to the symbIoTe information model.	1, 2, 3, 4, 5
15	1, 2	Functional	Interface	SHOULD	Useful for interoperability.		The system SHOULD provide best practices for the alignment of an IoT platform's information model with the symbIoTe information model, e.g., detailed examples documenting alignment procedures.	1, 2, 3, 4, 5
16	1, 2	Functional	Interface	MUST	Important for the correct operation of the system.		The system MUST provide unique identifiers of the (registered) IoT services within the system. Uniqueness MUST be enforced within and across IoT platform boundaries, including the case of mobile IoT devices.	1, 2, 3, 4, 5
17	1, 2	Func	Interface	MUST	Important for characterizing		symbIoTe MUST distinguish IoT devices	1, 3, 4,

		tional			the corresponding service. Applications/IoT platforms must be aware of mobility to assess the suitability of the corresponding resource to their purposes.		which are fixed (geo-location does not change over time) and mobile (their location changes).	5
18	1	Functional	Management	MUST	Obvious. Expressing a major symbIoTe concept.		The system MUST offer domain-specific enablers that hide from application developers the existence of multiple IoT platforms and resources targeted to a specific domain. (see Requirements 2, 5-12)	1, 2, 3, 4, 5
19	1	Functional	Management	SHOULD	The project will develop support for the creation of enablers. An enabler will be created as a proof of concept.		The system SHOULD allow application developers to create their own enablers (focusing on a single domain or be cross-domain), defining their own logic, etc. These "user-owned enablers" should be available at least to their creators.	1, 2, 3, 4, 5
20	1	Functional	Management	MAY	The main objective of symbIoTe is to support the development of enablers. The subsequent use of enablers is interesting but of low priority.		The system MAY allow application developers to share their custom enablers with other application developers. Trading mechanisms may be in place to govern the use of custom enablers.	1, 2, 3, 4
21	1, 2	Functional	Interface	MUST	This is a core symbIoTe functionality/concept.		The system MUST recommend IoT services based on the search criteria defined by an application developer or IoT platform provider.	1, 4, 5
22	2	Functional	Management	MUST	This is a core symbIoTe functionality/concept. The envisioned mechanism must not be application-specific so as to promote the generality of the symbIoTe approach.		The system MUST provide application-agnostic support for trading, bartering and cooperation of different IoT platforms. This MUST include an Auction System for businesses, customers and prosumers.	2, 3, 4
23	1, 2	Non-Functional	Performance	SHOULD	A desired feature, but it may not be fulfilled in particular cases of extreme system load.		Registration response time: SHOULD be in the order of minutes, depending on the volume of registered IoT services.	1, 2, 3, 4, 5

24	1, 2	Non-Functional	Performance	SHOULD	A desired feature, but it may not be fulfilled in particular cases of extreme system load.		Search response time: SHOULD scale with the volume of results data the volume of available data (with an upper limit). In the order of a few seconds.	1, 2, 3, 4, 5
25	1, 2	Non-Functional	Performance	SHOULD	A desired feature, but it may not be fulfilled in particular cases of extreme system load.		IoT service access latency: SHOULD have an upper limit of maximum 1-2 seconds.	1, 2, 3, 4, 5
26	1, 2	Non-Functional	Performance	SHOULD	A general desired feature. Satisfying this requirement is also subject to the available compute/storage resources.		Volume of IoT services supported: the system SHOULD target large volumes of meta-data (big data) provided by IoT platforms.	1, 3, 4
27	1, 2	Non-Functional	Performance	SHOULD	A general desired feature. Satisfying this requirement is also subject to the available compute/storage resources.		Number of IoT platform instances/enablers: the system SHOULD scale in the order of thousands of instances.	2, 3, 4
28	1, 2	Non-Functional	Performance	SHOULD	A general desired feature. Satisfying this requirement is also subject to the available compute/storage resources.		Number of applications/enablers: it SHOULD scale in the order of thousands of instances.	2, 3, 4
29	1, 2	Non-Functional	Performance	SHOULD	A general desired feature. Satisfying this requirement is also subject to the available compute/storage resources.		Volume of search queries: expressed as a search query rate. The system SHOULD support several hundreds of queries per second under the search response time requirement.	1, 2, 3, 4
30	1, 2	Non-Functional	Performance	SHOULD	A general desired feature. Satisfying this requirement is also subject to the available compute/storage resources.		Volume of monitoring information: refers to aggregated data collected and provided by the underlying IoT platforms. The system SHOULD target large volumes of data (big data) provided by IoT platforms.	4
31	1, 2	Functional	Management	MUST	An essential feature to promote healthy IoT services.		The system MUST support the ranking of the IoT service search results according to multiple criteria e.g., availability, performance, etc.	1, 3, 4, 5
32	1, 2	Functional	Management	SHOULD	The main target is to enable the filtering of the results subject to access rights.		The system SHOULD enable IoT platforms to control whether their IoT services appear in search results, subject to the access rights of	1, 2, 3, 4, 5

					Handling control of this feature to IoT platforms is of lower priority.		the query issued to these services i.e., whether the application developer or enabler is registered with the respective IoT platform.	
33	1, 2	Functional	Management, Interface	MAY	Useful but lower priority requirement.		The system MAY enable IoT platforms to define access rules to their IoT services . Such access rules refer to the intended availability of the IoT services to applications/enablers e.g., maximum 10 times per day, only from 7p.m. to 7a.m..	1, 2, 3, 4, 5
34	1, 2	Functional	Management	MAY	Useful but lower priority requirement. Basically targeting the more efficient utilization of available compute/storage resources of the symbIoTe system. Does not affect stakeholder facing functionality.	If not performed carefully this may result in reluctance of IoT services to register at first place. This points to some mechanism for contacting the operators/owners of the inactive services/platforms.	The system MAY periodically check the long term availability of registered IoT services with the purpose of purging or invalidating the corresponding registrations. The invalidation/purging should be preceded by communication with the owner/operator.	1, 3, 4
35	1, 2	Functional	Management	MAY	Not a core symbIoTe feature.		The system MAY support the registration of applications/enablers to underlying IoT platforms. This requirement pertains to cases where the search results contain IoT services that the query issuer does not currently have access rights for. An example mechanism for the intended symbIoTe support, is the redirection to the IoT platform registration interface.	1, 4
36	1, 2	Functional	Management	MUST	Important to be compatible with such underlying mechanisms. Push mechanism can substantially improve resource efficiency and		The system MUST allow applications to subscribe to IoT services to continuously receive the generated data/information, in addition to active requests for information from a used IoT service (when supported by the underlying IoT platform). In this mode of	1, 2, 3, 4, 5

					performance.		operation the application receives the data whenever this is pushed (published) by the corresponding IoT device.	
37	1, 2	Functional	Management	SHOULD	A useful feature that may not be necessary for platforms with hardly any updates/upgrades.		The system SHOULD support registration updates i.e., IoT platform operators/enablers should be able to update their registered IoT services with symbIoTe. For example, updating provided information upon sensor/actuator upgrades.	1, 2, 3, 4, 5
38	1, 2	Functional	Interface	MUST	Important for the correct operation of the system. (similarly to Req#18, but with a focus on the hierarchical structure).		The system MUST support hierarchically structured unique identifiers for the purpose of identification, trading, security and accounting. (E.g. the hierarchical information inherent in the domain names (cosy.computersciences.univie.ac.at) could be used)	1, 2, 3, 4, 5
39	1, 2	Functional	Management	MAY	Useful but not core symbIoTe feature.		The system MAY support notifications for updated results on past queries i.e., applications/enablers which have issued queries in the past, may be notified about the availability of new IoT services matching their registered past continuous queries.	2, 3, 4
40	1, 3	Functional	Management/Security	MUST			The system MUST support "multi-domain access rights composition", which means that an application registered in different IoT domains may be granted access to IoT services available in other domains. Specifically, the application is authenticated in each IoT domain where it has been registered, thus collecting a set of 'attributes'. Then, the application can combine attributes obtained in different contexts and be granted access to another IoT service exposed in a new IoT domain.	2, 3, 4
41	1, 2, 3	Functional	Management	MUST			The system MUST support an 'attribute mapping functionality' through which it is	1, 2, 3, 4

							possible to map attributes generated/released in one IoT domain to the same/similar attributes valid in different IoT domains.	
42	1, 2	Functional	Interface	MUST	Core functional requirement to enable keeping structured state about registered platforms.		The system MUST allow the registration of IoT platforms with the purpose of subsequently enabling them to register their IoT services (Req.1). The system MUST allow to unregister an IoT Platform.	1, 2, 3, 4, 5
43	1, 2	Functional	Interface	SHOULD			Enablers SHOULD be regarded as high-level IoT platforms that can register their domain-specific services to the system, similar to native IoT platforms.	1, 2, 3, 4, 5
44	1, 2	Functional	Interface	SHOULD	This is address the potential service disruption due to unregister capabilities. It is a useful but not core symbIoTe feature.	IoT services/platform owners would desire complete autonomy and therefore be reluctant to allow some "grace period" for applications to adapt.	symbIoTe SHOULD allow IoT services to gracefully unregister from the system. For instance, the process could foresee warning messages to affected applications/IoT platforms/enablers and/or some time period for them to adapt before completing the process.	1, 2, 3, 4, 5
45	1	Functional	Interface	MUST	IoT service/platforms would not participate if their presence is not equally treated.		The system MUST NOT favor any IoT service in the search results. The ranking of search results MUST solely based on objective criteria (see Req. 33).	1, 2, 3, 4, 5
Bartering and Trading (B&T) Requirements								
46	1, 2	Functional	Management	MUST			B&T MUST be available for B2B and B2C transactions.	2, 3, 4
47	2	Functional	Management	SHOULD			Businesses and consumers SHOULD be able to issue Vouchers (including predefined Service Level Agreements (SLA)), which they offer or require.	2, 3, 4

48	2	Functional	Management	MUST			Forward auctions MUST be available.	3, 4
49	2	Functional	Management	SHOULD			Reverse auctions SHOULD be available.	3, 4
50	1, 2	Functional	Management/Monitoring	MAY			symbIoTe MAY support the fine-grained monitoring of the availability of the IoT services engaged in established bartering/trading agreements (associated with SLAs), and the subsequent audit-proof archiving of the monitoring information.	3, 4
51	2	Functional	Management/Monitoring	SHOULD			Any Voucher consumer SHOULD be able to retrieve, from symbIoTe, the monitoring data associated with the acquired Voucher.	3, 4
52	2	Functional	Interface?	SHOULD			The search results in symbIoTe SHOULD indicate the possibility of accessing the chosen resources by means of B&T.	2, 3, 4
53	2	Functional	Management/Security?	SHOULD			The B&T transactions (B2C) SHOULD assure the anonymity of end users.	3,4
Smart Space & Smart Device								
54	3	Functional	Interface	MUST			The system MUST enable the discovery and registration of a new device that is willing to be registered with symbIoTe compatible platform middleware.	1, 2, 3, 5
55	3, 4	Functional	Interface	MUST			Any piece of equipment which needs to be integrated with symbIoTe is required to have a documented digital interface, providing either a standard or a properly described protocol.	1, 2, 3, 5
56	3	Functional	Management	SHOULD			The system SHOULD be able to prioritize the information sent to the platform (IMPORTANT information 1st)	1, 3
57	3	Non-Functional	Interface	SHOULD			The system SHOULD support the dynamic configuration of a subset of commercial sensors.	1, 3

58	3	Functional	Interface	MAY			Inside Smart Space multiple gateways MAY be used as an alternative fallback router for a given device.	1, 2, 3
59	3	Functional	Management	SHOULD			SymbIoTe smart spaces SHOULD be able to operate without a permanent Internet connection. (see Security Requirement 24)	1, 2, 3, 5
60	3	Functional	Management / Interface	SHOULD	Useful in case of limited connectivity		Different local IoT Platforms SHOULD be able to interact locally (i.e. without mediation from cloud-based L2 symbIoTe components).	1, 2, 3, 5
61	3	Functional	Management / Interface	SHOULD			Different collocated IoT platforms SHOULD (or even MUST) be able to interact locally with mediation from symbIoTe CLD components.	1, 2, 3, 5
62	3	Functional	Management	SHOULD	Useful in case of limited connectivity		A device running a symbIoTe app or a Smart Device SHOULD be able to access a Smart Space even if Internet connectivity is not available	1, 2, 3
63	3	Functional	Management	MUST	Important in case of limited connectivity (similar to #62, but the device is already associated)		A device running a symbIoTe app, when already associated to a Smart Space, MUST be able to access a Smart Device in that same Space even if Internet connectivity is not available.	1, 2, 3
64	3	Functional	Management	MUST	Important for identification of roaming devices		An L4 Compliant Smart Device MUST have a globally unique identifier.	1, 3
65	3	Functional	Management / Interface	SHOULD	Useful for roaming devices		An app/enabler SHOULD be able to receive a notification whenever an L4 Compliant resource it is using changes Smart Space association.	1, 2, 3
66	3	Functional	Management / Interface	SHOULD	Useful in case of limited connectivity		There SHOULD be a way for a local symbIoTe app to directly interface with the hosting Smart Space, that is by accessing it through the LAN rather than the Internet.	1, 2, 3, 5
67	3, 4	Functional	Management/Interface	MUST			SymbIoTe MUST accept visiting devices to be merged in the visited Smart Space.	1, 2, 3, 5
68	3	Functional	Management	MAY			The system MAY support IoT service /platform operators to alter the registration of their resources during runtime of applications.	1, 2, 3, 5

69	3	Functional	Interface	SHOULD			The symbIoTe on board gateway shall support the following digital interfaces: dry contacts, serial bus connections, Ethernet connections, other standard buses to be evaluated	5
70	3	Functional	Interface	MUST			The symbIoTe middleware components MUST be able to manage authentication and authorization functions.	1, 2, 3, 5
71	3	Functional	Interface	SHOULD			There SHOULD be a management interface to manage authN/authZ mapping between the local IoT Platform and symbIoTe core.	1, 2, 3
72	3	Functional	Management	SHOULD			The symbIoTe middleware SHOULD be able to interface with the local IoT Platform's functions to manage resource monitoring and accounting.	2, 3
73	3	Functional	Management	SHOULD			The symbIoTe middleware SHOULD be able to provide a mapping between potentially different metrics used across the Platform's border.	2, 3
74	3, 4	Functional	Interface	MUST			The symbIoTe middleware MUST be able to exchange information with the local IoT Platform regarding currently associated devices, as well as regarding devices leaving or requesting to join the local space.-	1, 2, 3, 5
Non-Functional / KPIs								
75	2	Non-Functional	Performance	SHOULD			The establishment of an SLA SHOULD complete within 1 sec, when an offering is already available.	3, 4, 5
76	1, 2	(Non-)Functional	Interface	MUST			The system MUST support arbitrary extensions of the common information model for the description of available IoT services across different IoT platforms.	1, 2, 3, 4, 5
77	1, 2	(Non-)Functional	Interface	SHOULD			The system SHOULD support mappings between two different extensions of the common information model.	1, 2, 3, 4, 5
78	All	Non-	Interface	MAY			The terminology used to describe the system	

		Functional					status must not be overly technical so that users can understand without having a technical background. symbIote MAY create best practices with unified terminology for developers of applications and enablers.	
Enabler specific requirements								
81	1	Functional	Monitoring	MUST			The system MUST monitor the quality of the offered services so as to make sure that the advertised quality of service is met e.g., number of aggregated sensors, accuracy of reported values, etc.	1, 2, 3, 4, 5
82	1	Functional	Management	MUST			The system MUST manage all the underlying resources so as to ensure the required quality, performance. For instance, an enabler that aggregates sensor readings throughout a country can search for new/alternative sensors in a certain area, if it experiences failures with already aggregated sensors there.	1, 2, 3, 4, 5
83	1	Functional	Management	MUST			The system MUST present a minimum application domain logic. In the simplest case this corresponds to the mere aggregation of IoT resources from multiple IoT services. More advances processing can be applied for the support of added value services.	1, 2, 3, 4, 5

Table 2 Security system requirements

Index	CL	Type	Category	Importance	Note on importance	Potential barrier for uptake	Description	Use Cases
1	1,2,3,4	Functional	Security	MUST	Important for interoperability and to control the access to the resources exposed by an IoT platform. It is needed for the authorization functionality.		The system MUST offer mechanisms for the **authentication** of symbIoTe entities/actors i.e., users/application developers, IoT platforms, developed applications and clients.	1, 2, 3, 4, 5
2	1,2,3	Functional	Security	MUST	Important for interoperability		The system MUST offer mechanisms for	1, 2, 3,

	,4				and to control the access to the resources exposed by an IoT platform. Platforms want to control the access over the resources.		the **authorization** of symbIoTe entities/actors i.e., users/application developers, IoT platforms, developed applications and clients.	4, 5
3	1	Functional	Security	MUST	Important for interoperability.		The SymbIoTe ecosystem must offer mechanisms to establish trust relationships - and thus implicitly trust levels - prior to applying security mechanisms for the first time. (E.g. online or offline means for verifying the true identity of the respective user/platform/software/...? shall be defined and put in place, agreement digitally signed by the joining platform)	2
4	Smart Space	Functional	Security	SHOULD	Useful to ensure the authentication and authorization requirements for use cases that won't be online all the time.		The authentication and authorization to a smart space SHOULD work even if the smart space is disconnected from the Internet.	1, 2, 3, 5
5	1, 2	Functional	Security	MUST	Important to avoid undesired access to resources or correct possible mistakes of unwanted granted access.		The system MUST support the revocation of access rights to users/application developers, IoT platforms, enablers. *(Comment: Although in the Yachting use case it might only be revoked when the system comes online again.)*	1, 2, 3, 4
6	1, 2	Functional	Security	MUST	Important to avoid to leave granted access for certain users to some services. No revocation has to be done when the right is expired.		The system MUST explicitly support access rights expiration.	1, 2, 3, 4
7	2	Functional	Security	MUST	Simplify the access the resources in federated environment		The authentication mechanisms of the system MUST support identity federation (e.g. single sign-on).	1, 2, 3, 4
8	1, 2	Functional	Security	MUST	Important for privacy issues.		The system MUST preserve end-user privacy. (E.g. locations of end users / sent sensor data and their identity, e.g. via data	1, 2, 3, 4, 5

							anonymization)	
9	1, 2, 3	Functional	Security	MUST	Important to securely protect data and that anyone else to have access to it		The system MUST support encrypted data communication between all involved entities on level 1 and 2 (e.g. the SymbIoTe core, platforms, etc.).	1, 2, 3, 4, 5
10	3	Non-Functional	Security	MUST	Important for privacy issues.		The system MUST ensure privacy protection on each layer, do not publicly expose e.g., devices information or services used by applications.	1, 2, 3
11	1, 2	Functional	Security	MUST	Important for interoperability. Having access policies that are easier to write.		The system MUST support fine-grained and standardised access rights to registered IoT resources, including also aggregated resources e.g., resources provided by enablers. E.g. it must be possible to identify individual sensors (which also allows tracking their wearers) for the layer which interpolates the air quality from individual sensors. This functionality is done on a domain specific layer. The output of this will not give data from sensors away but for other entities (like street segments). E.g. In the smart stadium use case, the "normal" user should not be allowed to see locations of individuals. Certain personal security might need access to this information.	1, 2, 3, 4
12	1, 2	Non-Functional	Security	MAY	Enhance the security at the application level and to simplify the usage of security services.		The system MAY provide best practices guide for applications to set-up end-user security in order to function in a secure and privacy-preserving way.	2, 3
13	1, 2	Functional	Security	MUST	Important for privacy issues and protecting against illegitimate access to third		The system MUST provide the possibility to let users / entities choose where (enablers/IoT platforms) their data is being	1, 2, 3, 4

					parties. Also for legal issues to be compliant with local laws.		used and processed. The users/entities MUST be able to modify the privacy parameters regarding their data.	
14	1, 2	Functional	Security	MUST	Important for users to understand the behavior of an application and why it is not working.		symbIoTe MUST detect and propagate any security error notifications through the SymbIoTe system to application/enablers/end user. The error message MUST be propagated but it doesn't mean that the final user will have the full details of the error. Some details might be just available for administrators.	1, 2, 3, 4, 5
16	1,2,3,4	Functional	Security	MUST	To simplify the way the access rules are defined.		Access rules MUST be defined as an access policy.	
17	1,2,3,4	Functional	Security	MUST	Important for interoperability.		The system MUST allow entities to delegate access to specific resources to other entities (e.g. by the usage of bearer access tokens)	1, 2, 3, 4
18	1, 2	Functional	Security	MUST	Basic functionality. To know who is somebody/something does not automatically give them rights to access the resources.		The system MUST support the authentication of the user without implied authorization for a specific resource. (E.g. it must be possible for platform B to have a user of platform A authenticated (by platform A) in a secure way while roaming in platform B)	2, 4
19	1	Functional	Security and Usability	MAY	It is optional due to the fact that the underlying applications and platforms must provide such a functionality.		Symbiote MAY support Multi-Factor Authentication towards if the underlying platform supports it. (e.g. Authentication using password and PIN)	
20	1, 4	Functional	Security	MUST	To avoid to man-in-the-middle attacks and identity spoofing.		Mutual authentication must be supported by all security mechanisms. (I.e. NOT only the	4

							user/application/software/... must be authenticated against the platform but also vice versa in order to facilitate malicious platform detection) Mutual authentication must be provided also in the communication between smart devices	
21	1, 3	Functional	Security	MUST	Important for interoperability. Using ABAC it is possible to cover more options. ABAC allows higher level of flexibility.		The access to resource MUST be handled through 'Attribute-Based Access Control (ABAC)' schemes. An 'attribute' refers to a generic property/role/permission that the application grants during the authentication phases.	1
22	4	Functional	Security	MUST	Interoperability and security between smart devices.	Constraints on the device	The link-level communication between two smart devices MUST be authenticated, encrypted, and integrity-protected. To this end, security mechanisms MUST be properly designed by considering specific security needs, the set of requirements expressed in terms of latencies, bandwidth and energy consumption, as well as the used communication technologies.	1, (5)
23	1,2,3,4	Functional	Security	MUST	To detect security attacks and discover not security related malfunctions.		The system MUST detect anomalies that appear in the usage of the system for instance abnormal consumption of resources like temperature sensors that indicates an attempt of a DoS/DDoS attack. Supposing that a temperature sensor in Smart Home is polled 8 times an hour on average. Suddenly we observe that in a given time interval this sensor has been polled 100 times in 10 minutes. Anomaly detection modul should detect it and send a log to the Platform where the user that has polled the sensor was registered.	1, 2, 3, 4, 5

24	1,2,3,4	Functional	Security	MAY	To confirm or not the trust in the platform federation.		The system MAY detect anomalies that appear in the metadata provided by platforms and devices. (e.g. The system MAY provide secure mechanisms to provide trusted location/proximity information.)	3

5 symbIoTe Architecture

This section presents details of the symbIoTe architecture, with focus on its domains: Application Domain (APP), Cloud Domain (CLD), Smart Space Domain (SSP) and Smart Device Domain (SDEV), as well as the functionality required for CL1, CL2, CL3, and CL4. We introduce an extensive list of components and define features which have been identified for APP (Section 5.1), CLD (Section 5.2), SSP and SDEV (Section 5.3) based on the requirements presented in Section 4. The template for component descriptions is given in Table 3, in compliance with the template recommended in IEEE STANDARD 1016: Software Design Specification [27]. In this document we focus on component descriptions, list of features and related requirements, while detailed component design will be provided in respective design and implementation tasks. Note that we define the components based on the domain in which they are placed, rather than which set of features or Compliance Level (CL) they provide. Nevertheless, the APP contains mostly components for CL1 and some components required for CL2, while the CLD hosts mainly components required for CL2 and some for CL1. Moreover, the SSP hosts most of the components required for CL3, while SDEV is required for CL4.

Since security requirements play a vital role for symbIoTe, Section 5.4 describes and analyses the symbIoTe approach to security[4]. Sections 5.5, 5.6, and 5.7 go a step further towards component design for the APP, CLD, SSP and SDEV and put the functionality of the components as well as security-related technical decisions into the context of CL1, CL2, CL3 and CL4. We introduce interfaces and communication diagrams depicting component interactions to achieve semantic, syntactic, and organizational interoperability within symbIoTe.

Table 3 Template for component description

Component	**Name of the component**
Compliance Level (CL)	1, 2, 3 or 4
symbIoTe Domain	Application (APP), Cloud (CLD), SmartSpace (SSP) or SmartDevice (SDEV)
Description	Short description of the component
Provided functionalities	List of functionalities provided by this component
Relation to other components	How will this component interact with other components?
Related use cases	Use cases in which the component is applied.
Related requirements	List of requirements from T1.3 that are addressed by the component

5.1 Application Domain

5.1.1 General concepts

The Application Domain (APP) components enable symbIoTe to become a mediator between applications and IoT platforms so that applications can use platform devices exposed as IoT services by various platforms in a uniform manner. The basic functionality needed for this domain is that of a registry service which maintains a repository of platforms which are symbIoTe-enabled, their services and properties. In addition to native IoT platforms, we envision that the registry will also maintain enabler-related information

[4] Note that security-related components are identified and described in previous subsections (Section 5.1, Section 5.2, Section 5.3).

describing enabler value-added services. This will enable application developers to choose IoT services and devices adequate for their applications. In addition, APP components need to enable efficient search techniques. If we make an analogy to a web search engine, symbIoTe should act as an IoT search engine which finds and recommends adequate resources[5] to applications/enablers. There are already some relevant search techniques proposed in the Web of Things space [19], and commercial attempts: Thingful[6] and Shodan[7].

The general concept of symbIoTe is to maintain only resource metadata, i.e., descriptions, within the APP, while applications and enablers will be directed to native platforms and enablers when accessing the corresponding resources. IoT platforms and enablers have the power to select resources which they want to expose to third parties, and they control the access to those resources. To do so, platforms need to extend their existing system with an open API and to comply with certain symbIoTe requirements in order to create an environment with uniform open APIs across various platforms. The open API and components required on the platform side are further discussed in Section 5.2 while we further explain the details of APP and CLD interaction for CL1 in Section 5.5.

Since the quality of search results is vital for APP, as well as for the adoption of symbIoTe in practice, symbIoTe needs to ensure that resources which it recommends and lists in search results are indeed online and available, while its ranking function should take into account parameters such as QoS, resource popularity, etc. Thus, the APP components and their features are defined to enable continuous monitoring of registered resources.

Another important aspect is covered by APP components: semantic interoperability. symbIoTe chooses to follow an approach which requires that all registered resources are defined using a minimalistic Core Information Model which all platforms need to adhere to, while further resource details are can be described using platform-specific information models. This provides a lot of flexibility for platform owners, since they can accommodate their existing information models and vocabularies, while symbIoTe also understands the key concepts defining sensors, actuators, devices and services. Further details regarding the symbIoTe information model and approach to semantics is provided in deliverable D2.4.

The components deployed in APP shown in Figure 5 are the following:

- **Administration:** facilitates the control of symbIoTe Core Services via a web-based GUI;
- **Registry:** stores data about registered resources offered to applications or enablers by using the symbIoTe core information model;
- **Search Engine:** enables applications/enablers to find relevant resources registered within the Registry;
- **Semantic Manager:** stores information models used within symbIoTe and verifies if the resources conform to the information model they claim to be using;
- **Core Resource Monitor (Core RM):** tracks availability of registered resources in order to ensure their availability;

[5] We use the term resource hereafter to reffer to various addressable services offered by both IoT platforms and enablers, as defined in Section 0

[6] https://thingful.net/

[7] https://www.shodan.io/

- **Core Resource Access Monitor (Core RAM):** tracks information about resource popularity, i.e., which registered resources are being used by applications/enablers;
- **Core Anomaly Detection:** identifies malicious attacks and other security violations on symbIoTe ecosystem;
- **Core Authentication and Authorization Manager (Core AAM):** ensures that trusted platforms register resources with symbIoTe, while mapping resource access rights to proper credentials;
- **Core Bartering and Trading Component (Core B&T):** comprises all bartering and trading functionalities for CL2 that need to be centralized, and are thus deployed within APP;
- **SLA Engine:** manages the lifecycle of service level agreements (SLAs) for the platform federation (CL2).

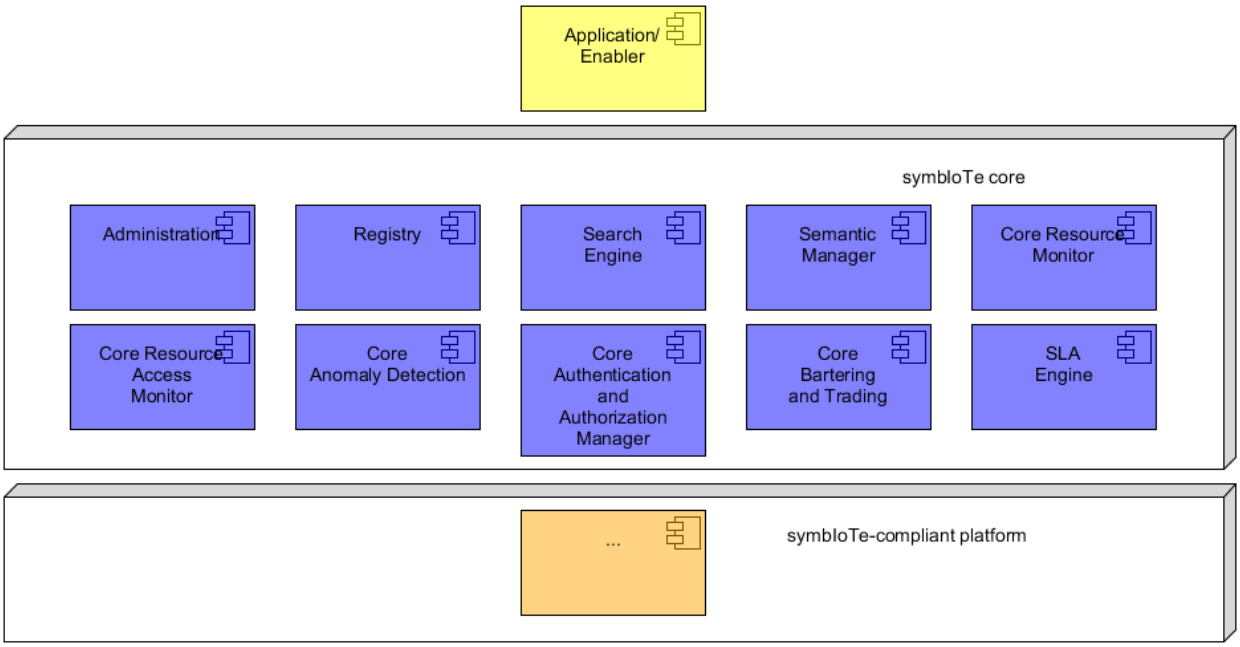

Figure 5 symbIoTe APP components

5.1.2 Component description

Hereafter we specify in detail the core components for the Application Domain (symbIoTe Core Services).

Table 4 Administration

Component	**Administration**
Compliance Level (CL)	1, 2
symbIoTe Domain	APP
Description	This component facilitates the control and administration of the symbIoTe Core Services by providing a web-based GUI. symbIoTe administrators will have access to a control panel that allows them to perform management actions and update parameters related to symbIoTe Core Services, such as removing specific platforms from the registry or changing the values of search engine variables to improve ranking. Furthermore, administrators will have access through the web-based interface to internal information,

	e.g. logs, IoT service popularity data, platform usage/status data or unauthorized user access attempts. This component will also provide features to non-administrator users. It will enable IoT platforms and applications to register with symbIoTe and to receive credentials which are required for the subsequent usage of symbIoTe services. We envision guest/trial, regular and premium registrations. We envision that enablers will also register with symbIoTe with two possible roles: 1) as platforms offering domain-specific IoT-related services and 2) as applications which use symbIoTe to find and access IoT services provided by symbIoTe-enabled platforms. The system should enable IoT platforms and enablers to control whether their resources appear in search results, subject to the access rights of the query issued to these services i.e., whether the application developer or enabler is registered with the respective IoT platform. The component is used in CL2 as a GUI for the management of federations. It provides federation-related information, and can be used to create a federation, join the existing one, or change federation affiliations.
Provided functionalities	• Provides a web GUI for administrators to manage platforms, resources, and other internal or database properties. • Presents logs and internal information to administrators. • Provides an interface for manual registration of IoT platforms, enablers and applications with symbIoTe. • Enables IoT platforms and enablers to control whether their resources appear in search results. • Supplies adequate credentials to IoT platforms, enablers and applications by the Core AAM. • Provides an interface for creating, joining, and updating federation affiliations for L2 Platforms
Relation to other components	Registry: manages platform/enabler/application-related data, handles requests for registry-related changes/actions. Core Resource Monitor: provides additional performance-related information for display. Core Resource Access Monitor: provides additional usage-related information and internal data for display. Core Authentication and Authorization Manager: Handles authentication of users and administrators.
Related use cases	All
Related requirements	34, 44, 45, S1, S5

Table 5 Registry

Component	**Registry**
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component must enable the registration and storage of resources which are offered by IoT platforms to be discoverable through symbIoTe. In addition, the component should enable the registration and storage of enabler domain-specific services which are offered through symbIoTe. Only IoT platforms and enablers which are symbIoTe Compliant and registered using the Administration component can register their resources. Additionally, only entities providing adequate credentials should be enabled to register resources. The component should support resource registration updates generated by entities providing adequate credentials, i.e., IoT platforms and enablers should be able to update their registered resources with symbIoTe. Examples are adding/removing resources, updating resource metadata upon sensor/actuator upgrades, etc. Registration response times i.e., the time required for the completion of the IoT service registration process should be in the order of minutes. The component must provide unique symbIoTe identifiers to all resources registered

	within the symbIoTe Core Services. Uniqueness must be enforced both within and across IoT platform boundaries, which is critical, e.g., in the case of roaming IoT devices.
Provided functionalities	• Handles requests for platform and enabler registration, unregistration and registration updates. • Handles requests for resource registration, unregistration and resource updates. • Assigns symbIoTe-specific unique identifiers to resources. • Stores resource metadata. • Receives resource availability information from the Core RM and updates resource status (e.g. online/offline).
Relation to other components	Search Engine: stores semantic representation of the data in Registry Semantic Manager: validates information models used to describe data, and validates if instances of data (resource descriptions) conform to the information model they claim to be using Core Resource Monitor: monitors availability of registered resources and pushes this information to the Registry Core Resource Access Monitor: monitors which application access which resources
Related use cases	All
Related requirements	1, 3, 11, 12, 15, 16, 18, 19, 25, 39, 45, S1, S2

Table 6 Search Engine

Component	**Search Engine**
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component enables application and enabler developers to search for resources available through symbIoTe Core Services. Search requests can be submitted in parameterised format, specifying metadata and relevance criteria for a search request, as well as application/enabler credentials. In addition, Search Engine also allows sending SPARQL search requests to query for meta information stored in the Core. When an application searches for resources in parameterised format, the component translates it into SPARQL query internally. After finding the resource descriptions in triple store, the component translates them from RDF to JSON, a format used to forward the response to the application. In case of a direct SPARQL query search provides output in format specified by the client. The component uses metadata associated to registered resources maintained by the Core RM and Core RAM to calculate resource scores with regard to a query. Resource metadata and annotations regarding their origin (i.e. particular platform or enabler) must be exposed in search results. Access policies associated to resources must be used to filter the results taking into account credentials provided together with the query. In search results, sensors which are fixed and mobile should be distinguished. The component uses a ranking function which takes into account resource metadata and relevance criteria defined in a search request, and should take into account various parameters (e.g. resource cost, availability, performance, offered service level).
Provided functionalities	• Searches for resources which match a specific query. • Stores meta information of entities (resources, platforms etc.) in the RDF store. • Finds the available resources and provides resource IDs which are subsequently used to access the selected resources. • Ranks resources relevant to a query in accordance with a symbIoTe-specific ranking model. • Filters resources for which a user does not have access rights. • Creates SPARQL query from a list of JSON query parameters
Relation to other components	Semantic Manager: provides query rewriting functionality. Core Resource Access Monitor: serves as a proxy to obtain URLs for resource IDs

	appearing in search results; provides information about resource usage used for ranking Core Resource Monitor: provides information about resource availability used for ranking
Related use case sections	All
Related requirements	2, 4, 19, 23, 26, 31, 33, S1, S2

Table 7 Semantic Manager

Component	**Semantic Manager** (SM)
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component stores the symbIoTe Information Model (Core Information Model, CIM, and Meta Information Model, MIM). Furthermore, it stores platform specific information models (Platform Information Model, PIM) and best practice information model (BIM). These models are stored to enable PIM validation, i.e. to ensure that PIM uploaded to the symbIoTe Core is aligned with the CIM. When registering resources to symbIoTe Core by using JSON description, the component translates resource description to RDF. It also validates the resources described using RDF to ensure they conform to the information model they claims to be using (PIM/BIM/CIM).
Provided functionalities	• stores MIM, CIM, BIM, PIMs • validates PIMs • validates if the instances of data (resource descriptions) conform to PIM/BIM/CIM they claim to be using • translates resource descriptions from JSON to RDF format • translates resource descriptions from RDF to JSON format • provides SPARQL rewriting functionality. • stores mappings between models
Relation to other components	Registry: sends the resource description obtained during registration process for validation. Sends new PIM models being registered by the platforms. Search Engine: Provides mappings and SPARQL rewriting functionality needed by Search.
Related use cases	All
Related requirements	5, 6, 7, 8, 32, S1, S2

Table 8 Core Resource Monitor

Component	**Core Resource Monitor** (Core RM)
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component must monitor the availability of registered resources to regularly update resource status (online/offline/unavailable). This will ensure that symbIoTe search results do not contain resources if their availability cannot be guaranteed, e.g., IoT service types with exclusive access rights (e.g., actuators) are marked unavailable if being used and will thus not appear in search results. The component needs to provide appropriate credentials when checking resource status. Primarily the component should receive information from Monitoring component in the CLD. If this data is not available, it can use scheduled tasks to check resource status. Either resource registrations or application-generated monitoring requests can trigger the

	scheduling of a monitoring task per resource (or a group of resources). In addition, the component should monitor the load on the registered resources, either in a push or pull style, if IoT platforms and enablers support such functionality. The component may perform functional performance tests on registered resources. The component should be designed to target large volumes of monitoring information.
Provided functionalities	• Receives monitoring information from IoT platforms and stores the results • Checks the availability of newly registered resources. • Monitors the availability of all resources registered within symbIoTe according to a specified schedule • Monitors the load of registered resources
Relation to other components	Search Engine: uses performance-related data obtained by the Core RM for creating and annotating search results. Monitoring (CLD): reports resource availability data from underlying IoT platforms
Related use cases	All
Related requirements	5, 6, 7, 8, 32, 52, S1, S2

Table 9 Core Resource Access Monitor

Component	**Core Resource Access Monitor** (Core RAM)
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component must monitor which resources are accessed from provided search results and maintain resource popularity information within the Core Services. It also acts as a proxy that redirects applications and enablers to the actual resources offered by symbIoTe-enabled platforms. Furthermore, it should collect resource access statistics from Resource Access Proxy in order to maintain resource popularity information within the Core Services. Such information is used as input to the ranking algorithm used in symbIoTe Core.
Provided functionalities	• Keeps track of which application/enabler uses which resources in a best-effort fashion. • Redirects applications/enablers to the actual resource offered by an IoT platform. • Estimates resource popularity
Relation to other components	Search Engine: uses resource popularity information for ranking. Resource Access Proxy (CLD): Core RAM redirects resource access requests to a specific platform/enabler RAP.
Related use cases	All
Related requirements	6, 10, 13, 14, 27

Table 10 Core Anomaly Detection

Component	**Core Anomaly Detection** (CAD)
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component is responsible for detecting 0-day attacks and other types of security violations by using signatureless approach. It will help detect malicious users attempting to avoid making a payment and get illegitimate access to resources owned by other parties. Furthermore, it will help identify denial of service (DoS) attacks on symbIoTe

	components, and attempts for unauthorized data deletion, insertion or update. The component will analyze data from Core RAM, Platforms' RAPs and other logs within the system.
Provided functionalities	• analyzing logs and events from symbIoTe Core components • analyzing traffic between Core and the platforms • analyzing topology of the connection • anomaly identification • notifying about the detected anomalies
Relation to other components	Core Resource Access Monitor: provides resource access data to be analyzed Resource Access Proxy (CLD): CAD analyzes access to RAPs of different IoT platforms
Related use case sections	All
Related requirements	14, 34, 35, 37, 40, 42, 43, 52, 59, S1, S2, S3, S5, S7, S19, S20

Table 11 Core Authentication and Authorization Manager

Component	**Core Authentication and Authorization Manager** (Core AAM)
Compliance Level (CL)	1
symbIoTe Domain	APP
Description	This component must offer mechanisms for the authentication and authorization of symbIoTe entities/actors, i.e. users/application developers, IoT platforms, developed applications and clients. The component must provide functionalities for configurable trust relationships between symbIoTe system and symbIoTe applications, through the use of X.509 certificates. It must authenticate components belonging to a given IoT platform integrated with symbIoTe, that would like to use services offered by the core layer (i.e., resource registration, resource unregistration, resource update, monitoring of the resource availability, search, resource access). It must authenticate applications registered in both core layer or in an IoT platform integrated with symbIoTe, that would like to search and access resources. It must issue home tokens for components and applications that successfully complete the authentication procedure in the Core layer. Home tokens contain attributes (i.e., roles, properties, permissions) that are used to access to resources and/or services managed by the symbiote Core, according to the Attribute Based Access Control (ABAC) paradigm. It must also revoke home tokens when the “expiration date” indicated in the token expires or, asynchronously, when an abnormal or frequent unauthorized use is detected. When the authentication in the core layer is initiated by a component belonging to a given IoT platform federated with symbIoTe or by an application registered within a given IoT platform federated with symbIoTe, it MUST: • validate the “home token” previously generated in the aforementioned IoT platform, • perform the check revocation procedure • (optional) for explicitly agreed and defined mapping between that platform and the core - convert home tokens to foreign by translating the set of attributes assigned to an entity in a given platform to another set of attributes that characterize the entity in the core layer (this operation is called Attribute Mapping Function) The component manages a public keys revocation collection and a Token Revocation List (TRL), which contain platform and core users public keys and tokens that were revoked by the users, platform owners or symbIoTe Administrator.
Provided functionalities	• Authenticates platforms and applications (users) registered in symbIoTe core within symbIoTe. • Releases tokens storing trusted attributes, if the authentication process was successful. • Manages certificates issued for each registered user

	• Validates access tokens that it has released, by verifying that the token is authentic (by checking the integrity of its sign) and that the expiration date indicated herein has not expired. This procedure is called "token validation". • Verifies that a valid token (i.e., a token that successfully passed the token validation procedure) has not been revoked asynchronously, before the expiration date indicated herein, through the use of a TRL or by communicating directly with the AAM that generated this token. This is called "check revocation procedure". • Verifies that the entity using a given token is really the entity for which that token has been issued (application's authentication), by a “challenge-response procedure”. • Translates the set of attributes included in a platform's HOME token into a new set of attributes in accordance to the explicit rules between participating IoT platforms with the core and issuing the ROAMED token for entities that is not registered within the core layer but are in possession of a valid HOME token. This procedure is called “Attributes Mapping Function”. • Issues guest credentials for users that don't have an account in any symbIoTe AAM and want to try out symbIoTe services which have public access.
Relation to other components	All symbIoTe components and resources requiring authentication and authorization within the Core Services.
Related use case sections	All
Related requirements	14, 34, 35, 37, 40, 42, 43, 52, 59, S1, S2, S3, S5, S7, S19, S20

Table 12 Core Bartering and Trading

Component	**Core Bartering & Trading** (Core B&T)
Compliance Level (CL)	2
symbIoTe Domain	APP
Description	This component comprises all bartering and trading functionalities which need to be centralized and coordinated by the symbIoTe Core Services, including the following: • Auctioning: this subcomponent performs forward and reverse second price auctions according to the offers and requests of producers and consumers • Trading: Registration of producer offers which are subject to direct (non-auction) trading afterwards • Bartering: Registration and matching of prosumer vouchers + sending resulting authentication tokens to both prosumers after a bartering transaction has been concluded
Provided functionalities	Auctioning: • Publishes new auctions based on SLAs (Service Level Agreements) received from producers (forward auction) or consumers (reverse auction) • Accepts offer bids (forward auction) from consumers or request bids (reverse auction) from producers for a specific auction • Determines the winner of a second-price auction and the corresponding price (second-price) • Sends resulting authentication token to the winner after the payment process has been concluded Trading: • Producer offers (SLA) are registered and made searchable Bartering: • Registers prosumer vouchers (including offered SLA, desired SLA, authorization token, validity date, ID) • Matches vouchers (offered SLA1 = desired SLA2)

	• Sends out corresponding authorization tokens after completing a matching procedure
Relation to other components	Bartering and Trading Manager (CLD), Registration Handler (CLD), Federation Manager (CLD), Resource Access Proxy (CLD)
Related use cases	ALL
Related requirements	46-53

Table 13 SLA Engine

Component	**Service Level Agreement Engine** (SLA)
Compliance Level (CL)	2
symbIoTe Domain	APP
Description	This component manages the whole lifecycle of service level agreements for the federation. Therefore, it is responsible for defining the service level agreements between the federation members through the federated templates, and managing the one-shot negotiation in order to create the agreements between the federated platforms. These agreements cover the quality of the resources and services of the platforms inside this federation, and they don't follow up agreements between end-users and platforms. Hence, the SLA is responsible for guaranteeing the conditions and the qualities of the shared resources and services between the federated platforms, at the L2. The component has to interact with the monitoring component to obtain the measurements needed to enable assessments of the defined services level objective (SLO), which has been defined into the agreements. The interaction is managed by a publish/subscribe mechanism or a pull model to obtain only the measurements for those metrics that directly map the SLOs to be assessed. The defined SLOs are always mapped to the actual metrics to be collected and supplied by the monitoring components. If any violation arises, the SLA component notifies/broadcasts to all registered components that are interested to be informed about the violation of the agreements, like the administration component which manage the federation layer. Moreover, such violations would be communicated to other components, such as the accountability and billing components, which would manage the necessary actions including the enforcement of penalties, discounts and charges. The component is compliant with WS-Agreement, whose specification describes an XML schema for specifying service level agreements (both applicable to SLA Templates, Agreement Offers and Agreements). SLA Templates, Agreement Offers and Agreements are defined and described using the WS-Agreement schema.
Provided functionalities	• Management of the SLA federated templates. • Creation agreements based on the templates. • Obtain the metrics directly related to SLOs of SLAs, either through the subscriptions or pull requests. • Generate and store violation events following the boundaries associated to the guarantee terms/SLOs • Manage the agreements and violations • Access to the agreement and violation details. • Notify/broadcast to all registered components any violation of the agreements on which they have subscribed.
Relation to other components	Administration, Federation Manager (CLD), Monitoring (CLD)

Related use cases	All L2+ compliant
Related requirements	49, 52, 77

5.2 Cloud Domain

5.2.1 General concepts

The Cloud Domain (CLD) components enable IoT Platform Providers to register desired resources to symbIoTe Core Services. In addition, they provide the means for applications/enablers to access those resources, found through the symbIoTe Core Services, in a uniform and secure way. These functionalities are a prerequisite for achieving CL1. Furthermore, the CLD components implement specific functionality for the management of platform federations required for CL2, i.e. the exchange of information between collaborating IoT platforms as well as bartering and trading mechanisms.

The components within CLD are shown in Figure 6. Each IoT Platform Provider will need to setup and maintain these components on the platform side as well as integrate them with their platform to achieve either CL1 or CL2. The CLD components are the following:

- **Registration Handler (RH):** enables IoT Platform Provider to register resources to symbIoTe Core Services;
- **Resource Access Proxy (RAP):** receives requests for resource access from application/enabler who found resource metadata by using symbIoTe Core Services;
- **Monitoring:** monitors status of IoT Devices and records when applications/enablers access IoT Devices/Composite IoT services;
- **Authentication and Authorization Manager (AAM):** enables a common authentication and authorization mechanism for all L2 Platforms and applications;
- **Federation Manager:** manages all federation affiliations and signed SLAs per platform, handles SLA violation notifications and triggers optimization requests and trust calculation updates;
- **Bartering and Trading Manager (BTM):** manages bartering and trading actions in advance to establishing federations;
- **Platform Registry**: stores resource metadata for L2 resources shared within a platform federation;
- **Subscription Manager**: component used for publish/subscribe interactions between platforms within a federation, used to update records stored within Platform Registry;
- **Trust Manager:** computes trust levels for other platforms within a federation;
- **Optimization Manager**: suggests optimizations, such as usage of resources from another federated platform if resources are collocated in the same smart space with the objective of reducing overall energy consumption or costs.

The aforementioned components represent *an umbrella* to the underlying platform specific components. They provide a uniform access to symbIoTe Compliant Platforms and hide

their heterogeneity from third parties. The platforms might have their own monitoring system. Somehow, all these platform tools must be unified for symbIoTe. On the other hand, we do not want to force a platform to include all their devices within symbIoTe. These tools allow the platform provider to select a subset of devices to be used within symbIoTe and to set specific SLAs, bartering & trading and security rules just for symbIoTe.

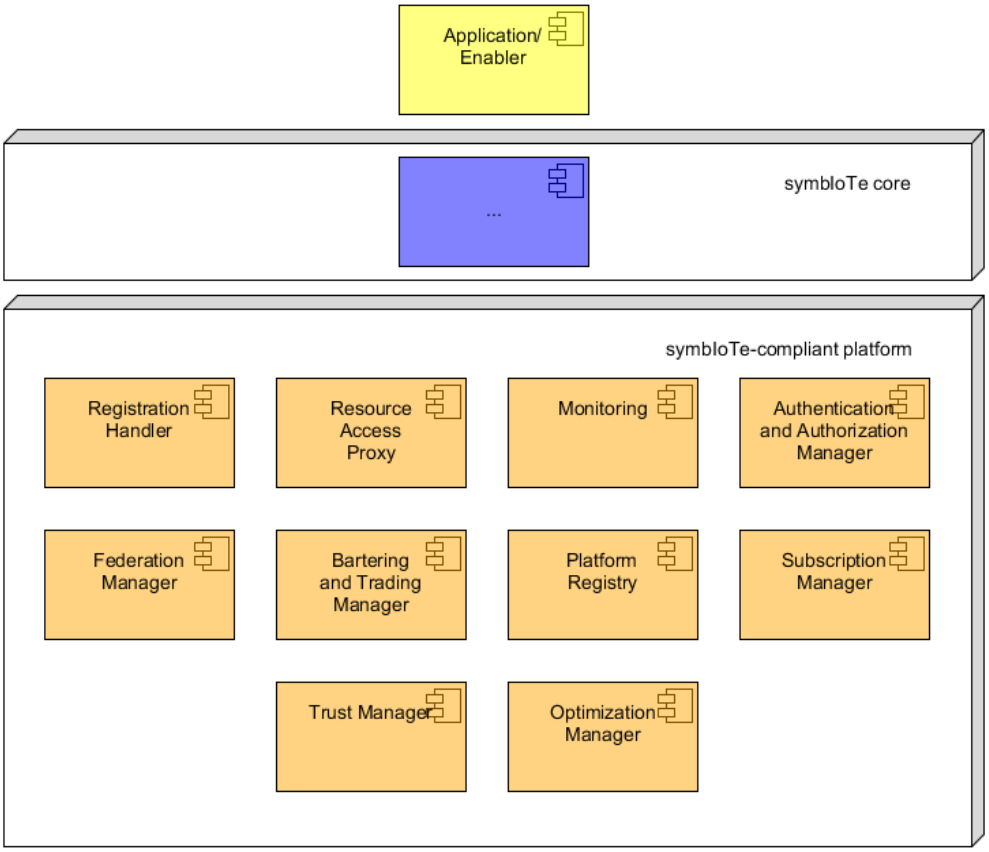

Figure 6 symbIoTe CLD components

Some of the tools, due to close relation with the platform, must be installed within the platform. We have, for example, the Resource Access Proxy which interacts intensively with an underlying platform and its devices. In this case, it makes sense to deploy such a tool within the platform near the devices. Other components placed in the CLD will mainly use data from the underlying platform and are very platform oriented, like the Bartering and Trading Manager or the Authentication and Authorization Manager.

5.2.2 Component description

Hereafter we specify in detail the components of symbIoTe Cloud Domain.

Table 14 Registration Handler

Component	**Registration Handler** (RH)
Compliance Level (CL)	1, 2, 3, 4
symbIoTe Domain	CLD

Description	This component will drive an IoT Platform Provider through the step of registering selected resources either to the symbIoTe Core (for CL1) or to an existing Platform Federation (for CL2). These resources can be either an IoT Device or a Composite IoT Service and must be described in accordance with the symbIoTe CIM. Some domain-specific properties of a resource can be reported during the registration process by using the PIM, but PIM needs to be an extension of the CIM. Registration Handler needs to provide the following information from the symbIoTe BIM: IoT Device or Composite IoT Service description, Location and its properties and Observed Property description and name. The Registration Handler will act as a proxy and broker between the Platform Owner that wants to share some resources and the Core or the Federation depending on the CL of the platform. An IoT Platform Provider must be able to define which of its resources are available to third parties (either to applications and enablers for CL1, or in federations for CL2), and under which conditions. A platform chooses a subset ot its resources to be exposes to symbIoTe. The IoT Platform Provider might also set the access policies for resources (e.g., a maximum of 10 times per day, only from 7p.m. to 7a.m., only to administrators of other platforms etc.). All this information will be received by the Registration Handler at registration or update time for a particular resource. After a successful registration in the Core (CL1) or the Platform Registry (CL2) it will notify interested parties with the resource data, so they can act accordingly. For example, the Resource Access Proxy might want to save the access policies for that particular resource (among other data) or the Bartering and Trading might want to save prices and access rules associated to it.
Provided functionalities	• Registers resources to the symbIoTe Core for L1 and Platform Registry for CL2, both virtual and physical, in accordance with the symbIoTe CIM • Updates resource status and unregisters resources • Notifies interested components about changes in the resources shared by the Platform Owner
Relation to other components	Registry (within symbIoTe Core Services): stores data about resource, assigns unique symbIoTe ID and keeps information about current resource status Bartering and Trading Manager: stores data regarding pricing of a resource Authentication and Authorization Manager: stores security information of a resource Federation Manager: stores data regarding the federation structure Monitoring and Core Resource Monitor (within symbIoTe Core Services): performs availability check/update
Related use cases	ALL
Related requirements	1, 3, 5, 14, 16, 17, 18, 19, 24, 25, 28, 34, 35, 36, 39, 42, 43, 54

Table 15 Resource Access Proxy

Component	**Resource Access Proxy** (RAP)
Compliance Level (CL)	1, 2
symbIoTe Domain	CLD
Description	This component enables symbIoTe Compliant access to resources within an IoT platform or enabler acting as a platform. It also provides access to resources shared through platform federation (L2 resources). Additionally for L2 resources, it accepts vouchers and sends them to BTM to check whether access to a resource can be granted. It must receive incoming access requests from applications/enablers using a symbIoTe-compliant communication protocol and data format. A request must contain a unique identifier assigned to a resource. It must check that a token included in the request is valid and that access to a particular resource can be granted (in accordance with specified access policies). The component must grant exclusive use of IoT resources, for underlying

	IoT platforms that support such feature. This applies in particular to resource types with exclusive access rights (e.g., actuators). It must enable the access control of resources according to fine-grained authorization policies and for reasons related to local legislation, security issues, etc., if the underlying IoT platforms support it. It may enable the prioritization access to IoT services if the underlying IoT platforms support prioritization, e.g., users with premium rights have priority over basic users when attempting concurrent access to some IoT services. Furthermore, the component will check with the Bartering and Trading Manager if the user has quota or will be able to pay for the access to L2 resources. Access to the resource is when all previous conditions are satisfied. The component may check the type of resource which is being accessed (e.g., whether it is a "simple" IoT Service or Composite IoT Service). In case of a Composite IoT Service RAP will retrieve the unique identifies for a set of IoT Services that are grouped under the umbrella of the composite service. The invocation to these underlying IoT Services will be done by this component. The data generated by IoT Services must be returned in a format which complies with the symbIoTe CIM and can include platform-specific definitions (PIM). The component must support subscriptions to resources so that applications or enablers can continuously receive the generated data/information. In this mode of operation the application/enabler receives the data whenever it is pushed (published) by the corresponding resource. When several access requests arrive at the same time, prioritization of requests may be supported.
Provided functionalities	• Enables authorized access to L1 or L2 Platform resources or to enabler resources and should enable request prioritization • Supports one-time requests for data delivery and subscription-based continuous data delivery • Ensures formatting of data generated by resources in accordance with the symbIoTe information model • Grants exclusive access to a particular resource (if the platform/enabler supports such feature) • Enables control of access to the advertised resources according to local legislation, security issues, etc. • Enables resource reservation (if the platform/enabler supports such feature) • Enables prioritized access to a resource (if the platform/enabler supports such feature) • Submits a received voucher to BTM to check whether access to resource from L2 Platform can be granted
Relation to other components	Bartering and Trading Manager Authentication and Authorization Manager (AAM) Federation Manager
Related use cases	ALL
Related requirements	8, 9, 12, 13, 14, 38, 56

Table 16 Monitoring

Component	**Monitoring**
Compliance Level (CL)	1, 2
symbIoTe Domain	CLD
Description	This component is responsible for monitoring the status and the load of the resources. The component will monitor periodically the status of the resources that are being exposed to symbIoTe. It will receive from Registration Handler the id of the resources that must be

	monitored. These ids can belong to IoT Devices or (Composite) IoT services. Prior to monitoring the Composite IoT Service, the component will need to get the ids belonging to the virtual entities that comprise this Composite IoT Service. When symbIoTe offers a group of IoT Devices in a single Composite IoT Service, the status of this Composite IoT Service must be determined. The information from the monitoring will be forwarded to the Core Resource Monitor. Additionally, the component monitors the load of the registered resources, if the IoT platform allows such monitoring. Related information can be directly retrieved by IoT platforms (if supported).
Provided functionalities	• Resource monitoring • Monitoring the load of resources
Relation to other components	Registration Handler Core Resource Monitor
Related use cases	ALL
Related requirements	5, 6, 32, 52

Table 17 Authentication and Authorization Manager

Component	**Authentication and Authorization Manager (AAM)**
Compliance Level (CL)	1, 2
symbIoTe Domain	CLD
Description	This component enables a common authentication and authorization mechanism for symbIoTe L1 and L2 Compliant IoT Platforms and applications. The AAM abstracts the native user and rights management functionality of each IoT platform and provides a uniform representation and structure of each user token. Each user token covers relevant user information, access rights per platform and common symbIoTe attributes with additional cryptographic entries to guarantee integrity and authenticity of the data itself. This approach enables an efficient and reliable access and policy validation workflow across all participating IoT platforms within the symbIoTe ecosystem. Another responsibility of the AAM is the management of the user token which include sign in, validation and verification, revocation and sign out features. If the current user token does not reflect the actual access permissions of the platform, the AAM will add the missing/changed attributes to it and return the modified token back to the application user.
Provided functionalities	• Map native user and rights management to common symbIoTe structure and format (user token) • Authenticate (sign in) users from symbIoTe Compliant Applications and issue respective user token • Validate/Verify symbIoTe Compliant user tokens plus integrity & authenticity • Enable sign out functionality for users from symbIoTe Compliant Applications • Check user token for any revocation/update in home and/or foreign platform • Enrich/modify valid user token with access rights for home platform • Issue tokens to users of foreign platforms and perform attribute mapping function based on presented attributes in home token • Provide management for required attributed set for the particular platform • Perform Challenge-Response to verify the user that is using a token is the entity for which the token has been issued for • Check certificate chain of emitted certificates to verify user and platform reliability

Relation to other components	Registration Handler Core Authentication & Authorization Manager Core Resource Access Monitor Resource Access Proxy Federation Manager Monitoring
Related use cases	All
Related requirements	14, 35, 37, 40, 42, 43, 52, 80, 81, S1, S2, S3, S5, S6, S7, S9, S11, S16, S17, S18, S19, S20, S22

Table 18 Federation Manager

Component	**Federation Manager**
Compliance Level (CL)	2, 3
symbIoTe Domain	CLD
Description	This component is responsible for managing all required federation information needed on platform level. It will receive federation and SLA updates from the core administration component and handles the execution of needed actions. These actions include updating the access policies on the platform level, providing up-to-date information to trust manager and monitoring components. It may also include triggering of optimization requests based on defined criteria. In addition, SLA violations will be received by this component and respective actions will be triggered in combination with the trust manager.
Provided functionalities	• Manage current federation affiliation and federation entities (joined federations, members) • Store all signed SLAs per joined federation for the individual platform • Received SLA violation notifications from SLA Engine relevant for the platform • Trigger and manage optimization requests • Mediate direct platform interactions within a Smart Space for collocated platforms
Relation to other components	Administration Authentication & Authorization Manager Trust Manager Monitoring Optimization Manager
Related use cases	ALL
Related requirements	33, 35, 61

Table 19 Bartering and Trading Manager

Component	**Bartering and Trading Manager** (BTM)
Compliance Level (CL)	2
symbIoTe Domain	CLD
Description	The component manages the bartering and trading between IoT platforms as far as this can happen in a decentralized way. Each IoT platform is able to set up bartering and trading rules for its resources (e.g. charges for resource usage or bartering options). It should be able to set different rules for the same resources depending on which platform is trying to

	access those resources, kind of access (prioritized or not), time that is accessed (daytime, night-time), how long the resource is used (camera used during half an hour or 5 minutes). The bartering and trading algorithm should only take into account the resources and services that will be exposed to 3rd parties, the information of the exposed resources/services can be retrieved from the Registration Handler. The module should be notified by RAP about the access to L2 resources and associate a price to this access. It needs the information about the access origin. Bartering algorithm is applied when a user from another federated platform tries to access a resource. For bartering it can exchange quota of access with 3rd party platforms. This component can access to the Bartering and Trading Manager from 3rd part platforms in order to request more quota in case there is no remaining one.
Provided functionalities	• Cost calculation from the resources • Access registration • Check quota of access
Relation to other components	Resource Access Proxy Registration Handler 3rd part Bartering and Trading Manager Central Bartering and Trading component
Related use cases	ALL L2+ Compliant
Related requirements	41, 42, 46-53

Table 20 Platform Registry

Component	**Platform Registry**
Compliance Level (CL)	2, 4
symbIoTe Domain	CLD
Description	This component must enable the registration of IoT Devices and (Composite) IoT Services which are offered by IoT platforms to be discoverable by other federated IoT platforms (L2 compliance). It should also support resource registration updates, like removing IoT services or updating their metadata. Moreover, Platform Registry must provide unique federation identifiers to all resources registered within the IoT federation. Uniqueness must be enforced both within and across IoT platform boundaries. A federation information model agreed by all IoT platforms participating in the federation must be supported by the Platform Registry to enable the description of available resources inside the federation. Furthermore, it should also provide search functionalities to discover resources either exposed by the home platform to the federation or by other federated IoT platforms which the home IoT platform has been subscribed to. It also maintains records about platform roaming resources in Smart Spaces (comparable to Home Location Register in mobile networks).
Provided functionalities	• Supports a federation information model for describing resources exposed in the federation. • Stores metadata of the home resources exposed to the federation • Stores metadata of federated resources which the home platform has been subscribed to. • Provides search functionalities • Maintains metadata about platform roaming resources in Smart Spaces
Relation to other components	Registration Handler Subscription Manager
Related use cases	All L2+ Compliant
Related	1, 2, 3, 11, 12, 15, 18, 19, 23, 25, 26, 31, 39, 40, S1, S2, 64, 65

requirements	

Table 21 Subscription Manager

Component	**Subscription Manager**
Compliance Level (CL)	2
symbIoTe Domain	CLD
Description	This component is responsible for sharing resource metadata between IoT platforms in a federation using the publish/subscribe communication mechanism. A platform that wants to access resources managed by federated platforms creates subscriptions for certain type of resources in which it is interested, and receives updates containing metadata (may include current resource availability or status) describing such resources from other Subscription Manager components in a federation. One Subscription Manager component is interacting directly with all Subscription Managers of federated platforms. One Subscription Manager component is used per platform. Using this component, an IoT platform publishes information to subscribed federated platforms about its new or deleted resources, and sends updates containing resource metadata. The component may also announce information about new resource types to federated IoT platforms. Platforms are able to define interest to receive resource metadata about certain resource types (e.g. mobile air quality sensors) in the form of subscriptions and can also define interest to receive updates about specific resources to store up-to-date metadata in Platform Registry.
Provided functionalities	• subscribing to resource metadata generated by other IoT platforms within a federation • publishing information about new or deleted resources to subscribed federated IoT platforms • notifying subscribed federated IoT platforms with updates about resource metadata • announcing information about new resources available in a federation to subscribed federated IoT platforms
Relation to other components	Platform Registry
Related use cases	All L2+ Compliant
Related requirements	1, 2, 3, 11, 12, 15, 18, 19, 23, 25, 26, 39, 40, 41

Table 22 Trust Manager

Component	**Trust Manager**
Compliance Level (CL)	2
symbIoTe Domain	CLD
Description	This component supports the platform in taking informed decisions by calculating the trust level based platform as well as resource properties (multi-layer approach). The platform trust level reflects the reputation of a platform within the symbIoTe ecosystem. The computation of the level is based on data collected during previous interactions with a given platform as well as available information provided by the Core and/or other platforms within the federation such as SLA violations or relevant Bartering & Trading history. A dwindling reputation should raise a red flag, signaling that the platform has not been fulfilling its contractual obligations at all or rather poorly. Consequently, a platform could be excluded

	from a federation or not even be allowed to join an existing one in the first place. Trust in resources is understood as the reliability of a resource in regards to its availability, accessibility and provided accuracy (efficiency) of the data (service). The main information source for calculating the reliability is the data supplied by the monitoring and anomaly detection components (as well as the reliability reports provided by the other federation members). Reliability has a major impact on the reputation of the members within a federation.
Provided functionalities	• Calculation of own resource trust values. • Calculation of foreign platform trust. • Calculation of "real" trust ratings for foreign resources.
Relation to other components	Search engine Core Anomaly Detection Monitoring Bartering and Trading Manager Federation Manager Monitoring
Related use cases	All L2+ Compliant
Related requirements	33

Table 23 Optimization Manager

Component	**Optimization Manager**
Compliance Level (CL)	2
symbIoTe Domain	CLD
Description	This component supports the suggestion of equivalent resources (from other platforms) if these resources are collocated in the same federation to optimize power/energy consumption, apply load balancing of equivalent resources, and enable global cost reduction within the federation
Provided functionalities	• Discover equivalent resources within the same Smart Space • Optimize resource allocation and power consumption • Increase resource availability by load balancing equivalent resources
Relation to other components	Federation Manager Monitoring Registration Handler
Related use cases	All L2 Compliant
Related requirements	8, 23

5.3 Smart Space Domain and Smart Device Domain

5.3.1 General concepts

Smart Spaces are environments (residence, campus, vessel, etc.) where one or more IoT platforms provide services. In order for such environments to be integrated into symbIoTe, we need to deploy proper software adapters (that we generically refer to as symbIoTe Smart Space Middleware, or **S3 Middleware**).

Some IoT platforms have both local and cloud architecture, whereas others have only local or only cloud components. Depending on each IoT platform's architecture, the S3 Middleware will need to be deployed either as a cloud component or as one of the Smart Space's appliances (or part thereof). Since the idea is to follow an approach as general as possible, though, the aim would be to design a software architecture that can be deployed in either way with no significant modifications. The functionality is duplicated in various domains, in CLD and SSP (e.g., device management in the CLD and SSP); what differs is the scope and the available hardware.

The Smart Space as a whole will expose (i.e. register, provide access to) the resources it contains, regardless of which "local" IoT platform they belong to; therefore, Smart Devices associated to the SSP will also be exposed directly, that is without being "mediated" by any of the local platforms. SymbIoTe Compliant local IoT platforms within SSP will be able to access all the resources associated to that SSP, provided that the required AA policies are in place. This includes both resources provided by any collocated IoT platforms, and those provided by the locally associated Smart Devices. In this way, the interoperability role played by symbIoTe will be fully functional also at the SSP level. When more than one IoT platform is active in a SSP, the S3 Middleware shall thus be acting as a local resource interchange.

Smart Spaces must be able to accommodate both incoming apps (a user with a smartphone or tablet running a symbIoTe app) and incoming devices (symbIoTe Smart Devices). In both cases, the incoming entity should be identified, authorized and given a way to access the Smart Space's facilities. This must be possible even in case of temporary failure or degradation of Internet connectivity.

In the context of symbIoTe, a Smart Device (SDEV) is a device that can directly interact with a Smart Space; according to this definition, any mobile device (smartphone, tablet) running a proper symbIoTe app can be considered an SDEV.

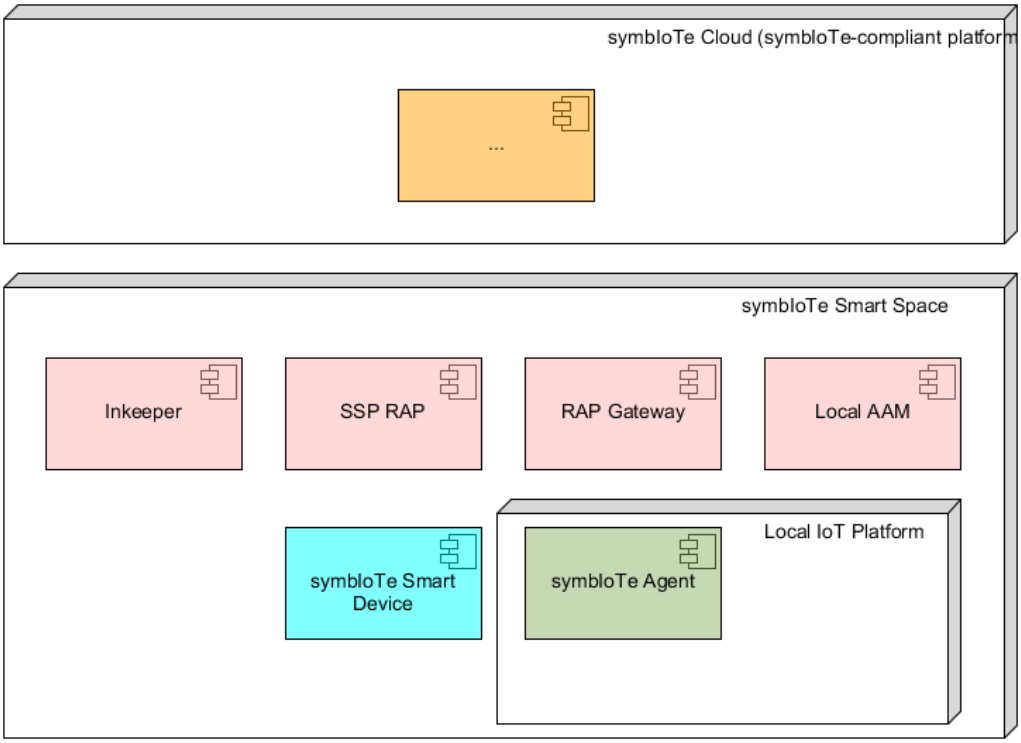

Figure 7 symbIoTe SSP and SDEV components

The components within SSP / SDEV are shown in Figure 7. Each environment considered as a Smart Space needs to be prepared with a deployment of the symbIoTe Smart Space Middleware. The SSP / SDEV components are the following:

- **Innkeeper**: it is used to connect app or Smart Device to the Smart Space, notifies a new app that it has entered a symbIoTe Compliant Smart Space, registers a new device to a local SSP keeps a register of locally registered apps and Smart Devices, and maps global URIs to local URIs;
- **SSP RAP**: allows direct access to the Smart Space's resources without going through the cloud components, so it is the local access point for symbIoTe applications running in the Smart Space;
- **RAP Gateway**: provides a resource access point for SDEVs to be addressable from outside the Smart Space;
- **Local AAM**: authenticates and authorizes symbIoTe applications and Smart Devices to allow access to the Smart Space even when no Internet connectivity is available;
- **symbIoTe Agent**: allows a symbIoTe-aware IoT platform to access Smart Devices registered in the same Smart Space, this component would allow a platform not only to expose its own devices to the symbIoTe ecosystem, but also to become a "client" (like an app or an enabler) and access foreign resources within the same Smart Space.

5.3.2 Components description

Hereafter we specify in detail the components for Smart Space and Smart Device Domains.

Table 24 Innkeeper

Component	**Innkeeper**
Compliance Level (CL)	1, 3, 4
symbIoTe Domain	SSP
Description	This component is the main entry point for a SSP. It is in charge of notifying symbIoTe-compliant apps and resources that they have entered a symbIoTe-complaint SSP and for handling their initial local requests. In particular, it is addressing the registration requests from CL3/CL4 devices entering the space and CL1 apps/enablers which require direct interaction to resources within the SSP (without the need to contact a resource through CLD service). It acts as a local registry of resources currently residing within its Smart Space (comparable to Visitor Location Register in mobile networks). Once a new L3 device/gateway is being registered, the Innkeeper checks whether the device shall be registered as a new device (CL3 behavior) or as a roaming device (CL4 behavior). In case it is an L3 device, it is registered in the Innkeeper's Registry and subsequently it may be registered in the Core Registry if the device will be exposed to third-party applications directly from Smart Spaces (bypassing CLD). In case of an L4 device registration, the Innkeeper needs to associate its global URI to its local URI, and notify the platform which manages the L4 device (its Platform Registry component) about this association. It also includes a web GUI for administrators to manage platforms, resources and other SSP information, including logs.
Provided functionalities	• Notifies symbIoTe-compliant apps and resources that they have entered a symbIoTe-complaint SSP and handles their initial local requests. • Provides a web GUI for administrators to manage collocated platforms in a SSP, resources, and other internal or database properties. • Manages local Registry storing metadata about resources registered in a SSP • Maps global URIs to local URIs for L4 devices • Presents logs and internal information to administrators. • Provides an interface for manual registration of IoT platforms collocated in a SSP. • Supplies adequate credentials to IoT resources, enablers and applications from the Local AAM.
Relation to other components	Registry: manages Innkeeper registration requests towards symbIoTe Core, only for resources that want to be exposed to third-party applications directly from Smart Spaces. Platform Registry: receives notifications about its L4 devices to associate device's global URI to local URI Local AAM: manages authentication and authorization requests
Related use cases	Smart Residence, Smart Yachting
Related requirements	54, 55, 57, 59, 64, 67, 68, 74

Table 25 SSP Resource Access Proxy

Component	**SSP Resource Access Proxy (SSP RAP)**
Compliance Level (CL)	1, 3, 4
symbIoTe Domain	SSP
Description	This component enables symbIoTe Compliant access to resources within a SSP to

	Applications and Enablers (either within or outside the SSP). It is the counterpart of the Resource Access Proxy in the Cloud Domain, which acts within a Smart Space and checks tokens and access policies with each request for access to resources in its SSP. It allows a local access to resources and Smart Devices within a SSP, providing all the features described for the Cloud component (defined in Section 5.2.2), without going through the symbIoTe Cloud components. It allows authorized access to SPP resources from L4 Smart Devices. It enables different collocated IoT platforms to interact locally either without or with mediation from symbIoTe CLD components (with support from Federation Manager under the assumption that platforms are in a federation with a valid SLA).
Provided functionalities	• Allows direct access to the Smart Space's resources without going through the Cloud components to apps and L4 Smart Devices within the SSP • Supports direct interaction between collocated IoT platforms with and without mediation of CLD components
Relation to other components	Application and L4 Smart Device: sends requests for resources access Local AAM: manages authentication and authorization requests
Related use cases	Smart Residence, Smart Yachting
Related requirements	59, 60, 61, 62, 63, 66

Table 26 Resource Access Proxy Gateway

Component	**Resource Access Proxy Gateway (RAP Gateway)**
Compliance Level (CL)	3, 4
symbIoTe Domain	SSP
Description	This component acts as a gateway, allowing access to local Smart Devices for clients that are outside the SSP. It establishes a tunnel connection to the Smart Space local network.
Provided functionalities	• Provides a resource access point for SDEVs to be addressable from outside the SSP
Relation to other components	Application: accesses to SSP towards the RAP Gateway, when it is outside the SSP SSP RAP: provides actual access to SSP registered resources
Related use cases	Smart Residence, Smart Yachting
Related requirements	58, 61

Table 27 Local AAM

Component	**Local Authentication and Authorization Manager (Local AAM)**
Compliance Level (CL)	3, 4
symbIoTe Domain	SSP
Description	This component enables a local authentication and authorization mechanism within a Smart Space. It is the counterpart of the AAM in the Cloud domain, so it provides all the features in a local environment, even in case of limited connectivity (without a permanent Internet connection).
Provided functionalities	• Allows AAM mechanisms within a Smart Space and even without cloud component connectivity • Implements configurable fallback policies to allow each Smart Space to balance security risks with available functions
Relation to other components	Innkeeper: requests authentication and authorization permissions SSP RAP: requests authentication and authorization permissions

Related use cases	Smart Residence, Smart Yachting
Related requirements	59, 60, 63, 70, 71

Table 28 symbIoTe Agent

Component	**symbIoTe Agent**
Compliance Level (CL)	3, 4
symbIoTe Domain	SSP / SDEV
Description	This component resides on IoT devices, gateways and mobile devices to provide features which make them discoverable in SSPs, and enable their registration with SSP's Innkeeper to facilitate dynamic configuration of L3 and L4 devices. A device running a symbIoTe Agent becomes a SDEV. A subset of commercial devices will be supported. It facilitates local interactions in SSPs via SSP RAP. Platform (or gateway) to platform, SDEV to SDEV, and platform to SDEV interactions and direct access to registered devices within the same SSP will be enabled (without and with interaction to CLD components for checking the access policies and SLAs). Such access should be enabled even without Internet connectivity. It is also the way for a platform to expose its Smart Devices directly to the symbIoTe ecosystem. We distinguish agents built for L3 and L4 devices since L4 devices need to maintain their globally unique identifies and need to register their temporary URI within their Platform Registry.
Provided functionalities	• Allows devices, gateways and applications entering a SSP to become discoverable in the new space and perform initial registration with the Innkeeper • Allows a symbIoTe-aware IoT platform to access SDEVs registered in the same SSP • Allows a platform to expose dynamically its own devices to the symbIoTe ecosystem • Allows a platform to become a "client" (like an app or an enabler) and access foreign resources within the same visited Smart Space.
Relation to other components	Innkeeper: manages device discovery and registration within a SSP, maps global URI to local URI for visiting L4 devices SSP RAP: manages access requests to resources Local AAM: manages authentication and authorization requests to check whether a device can be accepted into a SSP Platform Registry: maintains up-to-date associations of L4 global URI with local URI for its roaming devices
Related use cases	Smart Residence, Smart Yachting
Related requirements	54, 57, 60, 61, 63, 64, 65, 67, 69, 74

5.4 symbIoTe approach to security

Provision of data and system security in distributed, hierarchical systems like symbIoTe requires sophisticated mechanisms of user authentication and authorization. Security requirements described in Section 4 stem from the main use case when Smart Devices are interconnected with applications, while devices are managed by different IoT platforms. Attribute Based Access Control (ABAC) fulfils these requirements unlike Role Based Access Control (RBAC). The latter method of authorization known from local computer networks, which assigns each user a role like 'administrator' or 'normal user', is

impractical in distributed IoT environments [17]. Security in a symbIoTe network of IoT platforms is achieved more effectively with ABAC, whose paradigm falls within a wide set of logical access control schemes. Their goal is the protection of sensitive data or services from unauthorized operations like discovering, reading, writing, creating files and so on.

ABAC is based on the assignment of *attributes* to each client application and resource in a system. An attribute is defined as a particular property, role or permission associated to a resource in the system, assigned after an authentication procedure by the system administrator.

In ABAC, in contrast to other access control methods, the access to resources is controlled by Access Control Policies. Resource owner assigns an access policy to a resource defining a specific combination of attributes needed to grant access to the resource. Therefore, a client application can access a resource only if it possesses a set of attributes that match its predefined access policy. In symbIoTe this policy can contain at the same time attributes assigned to users and objects as well as particular environment conditions related to an access request.

The prevalence of ABAC over traditional access control schemes like Identity Based Access Control or Group Based Access Control (GBAC) is due to the efficiency, simplicity and flexibility of the access rules. In fact, complex policies can be created and managed without directly referencing potentially numerous users, applications and objects. Moreover, the structure of the policy can be independent from the number of users within a system, with enhanced flexibility especially in distributed environments, where specific domains can avoid any form of synchronization to create consistent access control policies.

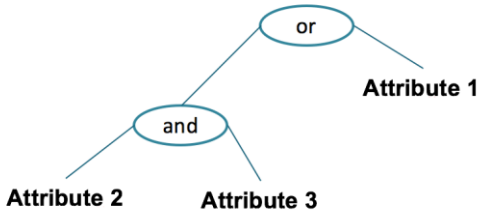

Figure 8 An example of access policy enforced by three attributes

For instance, with reference to the access policy depicted in Figure 8, an application may access to a resource if and only if:

- the list of its attributes contains at least Attribute 1;
- the list of its attributes contains at least Attribute 2 and Attribute 3.

In order to enable the ABAC logic in symbIoTe in a scalable, effective, and secure manner, proper security components, libraries and functionalities have been designed.

Security procedures are executed by introducing the following components: Core Authentication and Authorization Manager (CAAM), platform's Authentication and Authorization Manager (AAM, also refered to as PAAM), and Security Handler library (SH).

A user can be registered in Core layer or in any symbIoTe Compliant IoT platform. Indeed, it may perform a login procedure by delivering its own credentials (username and password). Without loss of generality, each user can be in possession of multiple resources. While the user is in possession of only one set of credentials, each resource is in possession of a unique public-private key pair and a valid X.509 Certificate. The X.509

Certificate is released by CAAM or PAAM, depending on where the user is registered to. Therefore, for each resource, the user generates a key-pair (made up by Public Key and Private Key) and sends a Certificate Signing Request to the reference AAM. Then, the AAM generates and delivers a valid Certificate to the user. Finally, the certificate is stored by the resource. In order to make the process for generating all the certificates scalable and effective, symbIoTe makes use of a certificate trust chaining architecture. Further details are reported in Section 5.4.1.

In line with the previously described ABAC logic, user properties are encoded by attributes stored within a dedicated data structure referred to as token. In symbIoTe, a token can be released by AAMs only. Moreover, it mainly contains the list of attributes assigned to (and valid for) a user within symbIoTe Core or any other IoT platform. In summary, three kinds of tokens are available in symbIoTe:

- **home token** released by the AAM where the Application/Enabler is registered;
- **foreign token** released by the AAM in exchange for home token when the AAMs (platforms, enablers, core) come into an agreement; and
- **guest token** released by any symbIoTe AAM and used to access public resources in symbIoTe.

When a user logins to CAAM or PAAM, a home token is provided, which stores all the attributes assigned to the user, the public key of the user's client and other important security parameters. The main structure of the token used in symbIoTe is reported in Section 5.4.2. With this token, the user may immediately start an access procedure to resources available in the domain where the user is registered.

With home tokens, the user may try to access resources exposed by other IoT platforms for which he/she does not have an adequate home token. In such a case, the user may request a foreign token from the AAM responsible for this other platform. During this procedure, the user must confirm the ownership of the token through a challenge-response mechanism, described in detail in Section 5.4.4. In case when the challenge-response procedure ends successfully, the AAM of the foreign platform validates (optionally offline) the home token provided by the user. Then, it generates a new, foreign token, usable according to the agreement details (e.g., in the foreign IoT platform only). Note that the generation of a foreign token implies the execution of an attribute mapping function that translates/maps/renews/updates the original list of attributes assigned to the user to a new set of properties valid within the foreign IoT platform. Further details are reported in Section 5.4.5.

Home and foreign tokens, issued for relevant clients are not[8] shareable: a user needs to acquire tokens for each of his/her clients from relevant AAMs.

The RAP is responsible to check access policies based on attributes provided in a token. Further details are reported in Section 5.4.6.

The revocation process invalidates access to the core or platform and its associated resources. It can be synchronous or asynchronous. It involves attributes (and therefore tokens) associated with an application. Further details are reported in Section 5.4.7.

[8] Technically they are shareable but sharing a token requires also sharing the matching private key store in a secure manner in the client application and users are encouraged to have a client+privateKey pair per each of their resources to provide fine-grained security

Note that symbIoTe also supports guest users: users can request public access credentials, i.e., guest credentials, for that they contact any AAM in symbIoTe by using a REST endpoint and get a token allowing them to use un-restricted resources.

5.4.1 Certificate Trust Chaining

A certificate chain is a list of certificates needed to certify an end subject. The certificate chain contains an end certificate, certificates of intermediate Certificate Authorities (CAs) and the certificate of a root CA.

An intermediate CA holds a certificate issued from root CA. The trusted root CA issues a self-signed certificate (or it may be released from VeriSign or other trusted CAs).

The certificate chain typically consists of three certificate types:

- **Root Certificate**: The certificate that identifies the CA. The root CA signs an intermediate certificate, forming a chain of trust.
- **Intermediate Certificate**: The certificate that identifies a Subordinate CA. This certificate is digitally signed and issued by a Root CA. An intermediate certificate authority is an entity that can sign certificates on behalf of the root CA.
- **End Certificate**: A certificate that links a public key value to a real-world entity such as a user or an application.

Certificate chain verification is the process that checks the validity period, and the signature of respective CA.

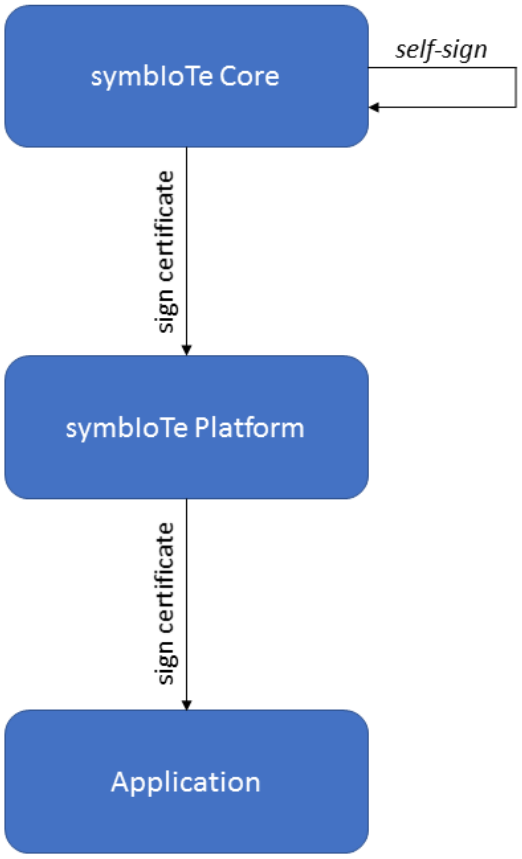

Figure 9 Certificate Chain

In symbIoTe, the *Root Certificate* is the **symbIoTe Core certificate**, the *Intermediate Certificate* is the **symbIoTe IoT platform certificate** and finally the *End Certificate* is the **Application Certificate** (generic Data Consumer).

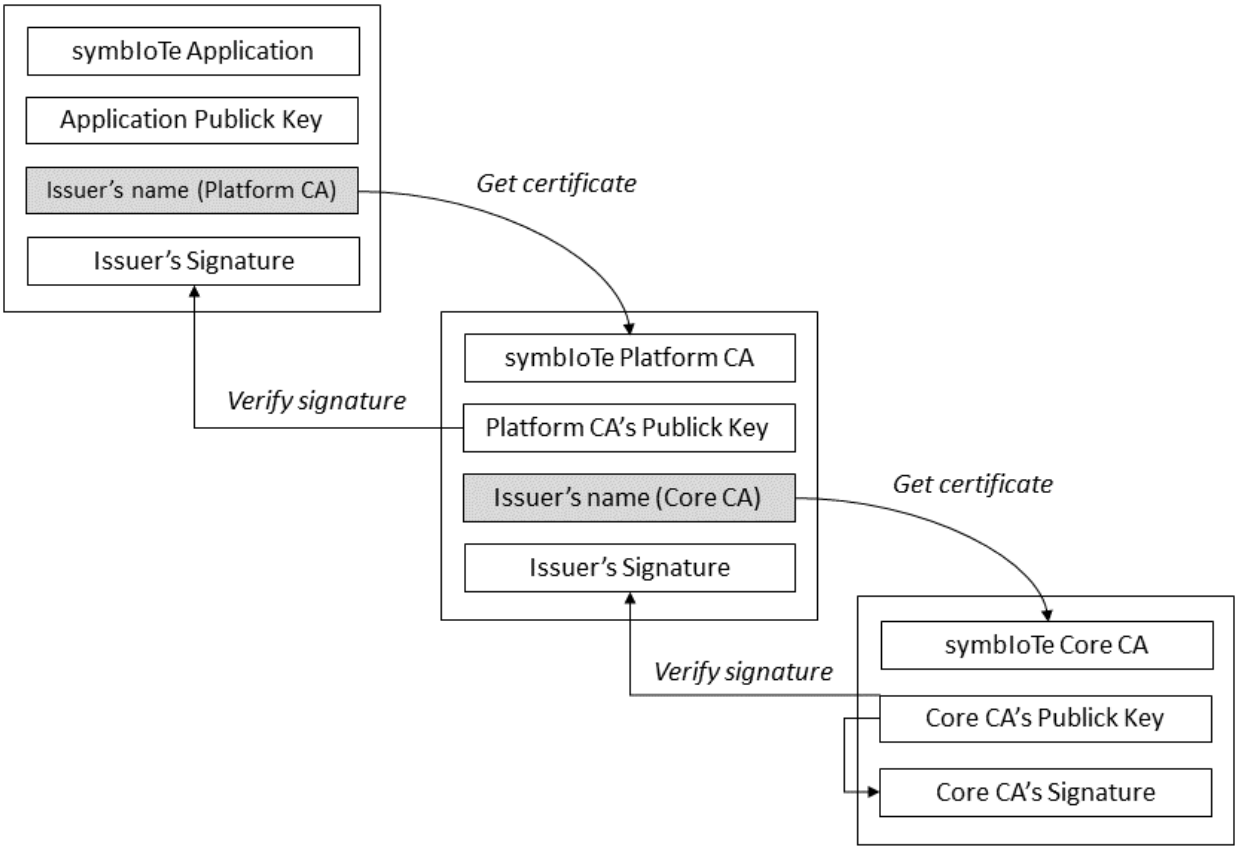

Figure 10 Certificate Chain Workflow

The main steps to create a certificate chain are the following:

- The Root CA generates a keystore for symbIoTe Core (Root CA) with certificate and private key, and afterwards exports symbIoTe Core certificate (Root CA self-signed certificate).
- An IoT platform (e.g. a generic Platform i.e. Platform P) generates a key pair for itself, generates a Certificate Signing Request (CSR) and submits the request to the Root CA (symbIoTe Core) that issues a certificate for Platform P.
- An Application (leaf, this isn't a CA) generates a key pair for itself, generates a CSR and submits the request to the Intermediate CA (Platform P in this example) that issues a certificate for Application.

Finally, each user should obtain one certificate for each associated resource.

5.4.2 Authorization Token

Token is a digital object used as a container for security-related information. It serves for authentication and/or authorization purposes and generally appears as a list of elements. Each element contains an assertion that further specifies properties assigned to the owner of the token.

Each token must contain an explicit expiration date, indicating the date until the token is considered valid. Moreover, the token also contains at the end an element that certifies its

authenticity and integrity. Depending on the chosen solution, validating a token could require different procedures. In symbIoTe, JSON Web Token (JWT) standard is used for tokens generation. JWT fits perfectly within the symbIoTe reference architecture. From a cryptographic perspective, the only preliminary requirement for its adoption is the deployment of a public-key infrastructure, which issues a private/public key pair to each entity in the system. In symbIoTe, we envision a trusted CA in the core layer that issues certificates to platforms and applications. AAMs use their private keys to generate and sign tokens. Any entity in the system that receives a token could easily verify its authenticity by gathering the public key of the issuer of the token (specified in the token itself). Important features such as the support for an expiration date are also integrated by JWT.

In addition, each token can be easily associated to a given entity in the system, e.g., the public key of the owner of the token can be embedded within the token. This can be used in the *challenge-response* procedure to prove the possession of the respective private key and verify that the application using the token is effectively the entity for which the token has been generated. This procedure avoids replay attacks and it is described in Section 5.4.4.

Token includes its type, i.e., if it was generated for a registered user (HOME) or by agreement exchange between AAMs (e.g. federation, foreign). Furthermore, we assume that each token embeds multiple attributes. Therefore, applications and components possess many tokens, each of them is associated with multiple attributes, received from an IoT Platform. In this way, applications and components collect a wallet of tokens, that they use carefully to access resources exposed by the symbIoTe Search Engine or RAP within IoT platforms.

The signature field, computed over the whole set of parameters previously mentioned, is included at the end of the token. It is generated through the ECDSA-256 algorithm, computed over the whole set of fields included in the token.

5.4.3 Token Validation

Validation of tokens, keys stored in them and the signature is one of the most important security tools in symbIoTe.

A unified API is exposed for the application developers to validate tokens, available in the Security Helper library. To check the validation, symbIoTe security layer firstly verifies the token string contents, to determine whether it is malformed during transmission, or whether it has a correct signature and has not yet expired. Afterwards, the AAM authority checks the token.

Core or Platform AAMs check if the token issuer exists in the symbIoTe ecosystem, if the issuer’s or subject’s public key was revoked, or if the issuer’s or subject’s certificate has expired.

5.4.4 Challenge-response procedure

The Challenge-Response procedure is designed to verify that a component/application using a token is really the entity for which this token has been issued by an AAM. The procedure leverages public key cryptography: the owner of the token is in possession of a private key associated with a public key stored in the token; therefore, it executes some

cryptographic operations by using such a private key; then to verify the authenticity of the application/component, an opposite operation is performed using the public key. In symbIoTe, a special challenge-response mechanism is designed in accordance with the REST paradigm.

Prerequisites

- Client wants to get access to a resource (Res 1), from Platform (Plat1);
- Each component in symbIoTe is in possession of its certificate and the corresponding private key;
- The user has n devices. For each device, it retrieves a private key, a certificate and valid tokens. From this moment on, the user-device pair is simply referred to as “client”.

Notation

- PV is the private key of the client.
- PK is the public key of the client.
- T_i represents the homeToken issued by the *i-th* Platform AAM to the considered user. The token stores in the SPK field the public key of the user. When a user is registering to the platform, he/she obtains a certificate in order to demonstrate the authenticity of its public key.
- H is a generic hash function.
- E() represents a cryptographic operation executed using a public key of the entity that will receive the message.
- S() is a signature vector that contains the signatures made on the hash of correspondent token and a timestamp.
- $S_{PV,i}$ is the signature made with the *i-th* private key associated with the correspondent subject public key.
- $\mathbf{T}=[T_1,T_2,...,T_n]$ is the token list owned by the client.

Scenario

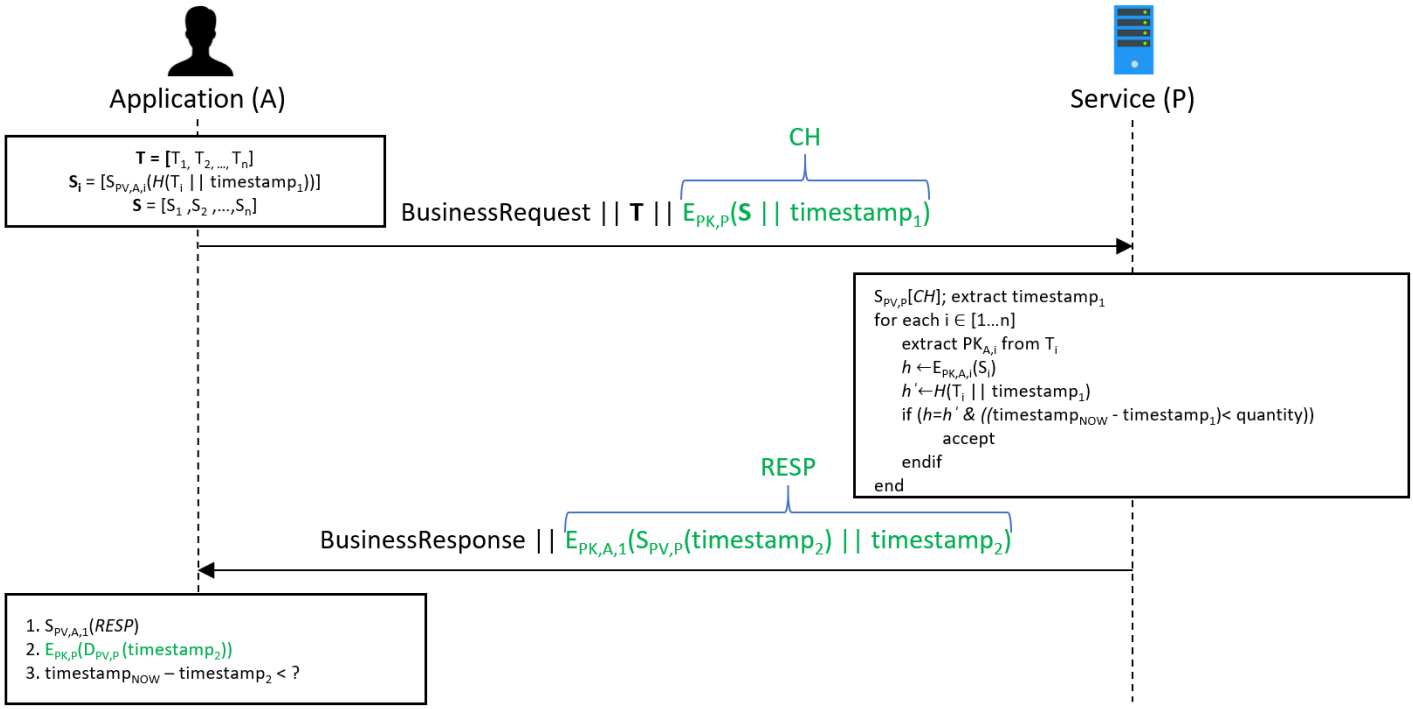

Figure 11 Challenge-response mechanism

The challenge and response mechanism involves the following steps:

1. It is assumed that the client already retrieved the set of tokens
   - For each *i-th* token, the client computes $S_i=[S_{PV,A,i}(H(T_i || timestamp_1))]$
2. The client sends a request to the remote entity, which contains:
   - The business request
   - The list of tokens: **T**
   - An implicit challenge calculated as: $E_{PK}(S_{PV,1}(H(Token_1 || timestamp_1)) || S_{PV,2}(H(Token_2 || timestamp_1)) ... || ... S_{PV,n}(H(Token_n || timestamp_1)) || timestamp_1)$
3. The remote entity performs the following operations:
   - $S_{PV,P}[CH]$; extract $timestamp_1$
   - for each *i* in [1…n]
   - extract $PK_{A,i}$ from T_i
   - $h \leftarrow E_{PK,A,i}(S_i)$
   - $h' \leftarrow H(T_i || timestamp_1)$
   - if $(h=h' \& ((timestamp_{NOW} - timestamp_1) < quantity))$ accept
   - end if
   - end
4. The remote entity sends an answer as: BusinessResponse || $E_{PK,A,1}(S_{PV}(timestamp_2) || timestamp_2)$
5. The client calculates:

- Decrypt the RESP: $S_{PV,A,1}(RESP)$
- It verifies the signature: $E_{PK,P}(D_{PV,P}(timestamp_2))$ and checks the freshness of the response.

When an application requests a foreign token, it will contain the subject public key as the first public key extracted from the token list during the challenge–response procedure. Note that timestamps are used to guarantee the freshness of requests and responses.

5.4.5 Attribute mapping

As previously mentioned, ABAC provides fine-granular access control, based on the list of attributes presented by the user to the particular authorization body, as well as the possibility to define complex access policies for any resource or service. However, since symbIoTe incorporates federated access control mechanisms, AAMs in the Application and Cloud domain must be able to provide mapping for attributes issued by symbIoTe core or any symbIoTe Compliant IoT platform. Nonetheless, access control mechanisms in a particular platform can differ from the ones used by other platforms or symbIoTe Core, meaning that those differences have to be resolved in order to grant access to the resources. In order to resolve those issues, required attributes set has been defined to enable issuing access tokens with attributes, validate access policies using those attributes, and exchange access tokens. To be able to exchange access tokens, attribute mapping rules need to be defined.

5.4.6 Access Policy Checking

The concept of Attribute Based Access Control (ABAC) represents a logical authorization model that provides a dynamic and context-aware access control mechanism. It serves to protect resources (i.e., data, devices, services and other) from unauthorized operations like reading, writing, editing, deleting, copying, modifying and executing. The owner of a resource establishes a policy that describes which attributes, who, and what operations can be performed on this resource. If a subject has the attributes that satisfy the access control policy established by the resource owner, then the subject is authorized to perform the desired operation on that object. The policies are defined by using eXtensible Access Control Markup Language, a standard maintained by OASIS.

Attributes used by an application trying to get access to resources on a certain IoT platform are checked when accessing AAM of that platform. In the case when the attributes used by the application and this IoT platform are not in the same format, attribute mapping needs to be performed. This functionality is executed on AAM, and attributes in the format of the IoT platform are forwarded to the application. When application then contacts IoT platform RAP, it uses the attributes in the same format used by the platform.

5.4.7 Revocation

The revocation process handles the synchronous and asynchronous revocation of authorization credentials. Synchronous means that token/certificate expire when the check revocation procedure compares the actual timestamp with the expiration value written in the token/certificate. Asynchronous means that the token/certificate is revoked before its expiration. Furthermore, the AAM manages a Token Revocation List (TRL) storing the list

of JWT ID ("*jti*" claim) tokens that have been revoked before their expiration. Therefore, if the JTI is present in this list, the access to resource or in general to the platform will not be allowed. For this reason, it may be contacted by any component in the architecture during the check revocation procedure. However, in addition to controlling the token ID, a check of the validity of the keys associated with the user is made. In fact, if keys are revoked (stored in the Public Key revocation list) this means that the pair (APP, device) hasn't the rights to access the resources of a particular platform.

symbIoTe will allow the following actors to revoke through dedicated APIs:

- symbIoTe administrator:
  - Platforms' certificates;
  - Core AAM users' client certificates;
  - Tokens issued by Core AAM;
- AAM administrator:
  - its users' certificates and tokens;
- symbIoTe user in its home AAM(s):
  - client certificates; and
  - its tokens.

All actors will need to present their credentials (username and password) to authorize their actions.

5.4.8 Other security requirements

At the basis of all aforementioned security mechanisms lies a requirement that all communication in symbIoTe is secured using TLS – namely all interface exposed to the internet and used to communicate symbIoTe clients with our services as well symbIoTe services among themselves (e.g. platform – platform or core – platform) are available only using the HTTPS protocol.

For maximum reachability and compatibility of symbIoTe with the firewall policies across the Internet, we suggest that those services should be running on standard HTTPS port 443.

Internal communication between components within the platforms is not symbIoTe responsibility however it is advised that symbIoTe integrators take utmost care to secure their components deployment.

Platform AAMs should have internet access to other AAMs in order to provide full security enforcement. If that is not possible, they should at least be able to retrieve security data from the Core AAM and if that is not possible they will fall back to offline validation.

5.5 Achieving Compliance Level-1 (CL1)

As already stated in Section 3.3, CL1 means platform syntactic and semantic interoperability. It is a prerequisite for any kind of IoT platform interoperability, and requires an open but controlled access to IoT platform services. There are two major requirements for platforms that want to become L1 Compliant:

1. the existing platform-specific information model needs to be mapped to the symbIoTe Core Information Model (semantic interoperability); and
2. the platform must integrate the symbIoTe Interworking Interface to open up its northbound interface and provide access to IoT services (syntactic interoperability).

Here we focus on the second requirement, since the first requirement is the topic of D2.4. We identify the components that need to be integrated with existing platform components in CLD, as well as the Core Service components in the APP required for an interoperable IoT ecosystem offering IoT services across platforms. Since authenticated and authorized access to offered services is vital for an IoT ecosystem, we include here also security-related components.

The component diagram of a symbIoTe ecosystem which includes L1 Platforms is presented in Section 5.5.1. It also identifies component interfaces. Communication diagrams specifying L1 functionalities are included in 5.5.2.

5.5.1 Component diagram

Component diagram with specified interfaces is shown in Figure 12. For CL1, symbIoTe system defines four interfaces:

- **Application Interface** used by symbIoTe core components to interact with applications or enablers;
- **Core Interface** used by applications or enablers to interact with symbIoTe core components;
- **Cloud-Core Interface** used by a symbIoTe Compliant Platform to interact with symbIoTe core components; and
- **Interworking Interface** used by symbIoTe core components to interact with symbIoTe Compliant Platform, and used by applications or enablers to interact with a symbIoTe Compliant Platform (a subgroup of interworking interface is named Application-Cloud Interface).

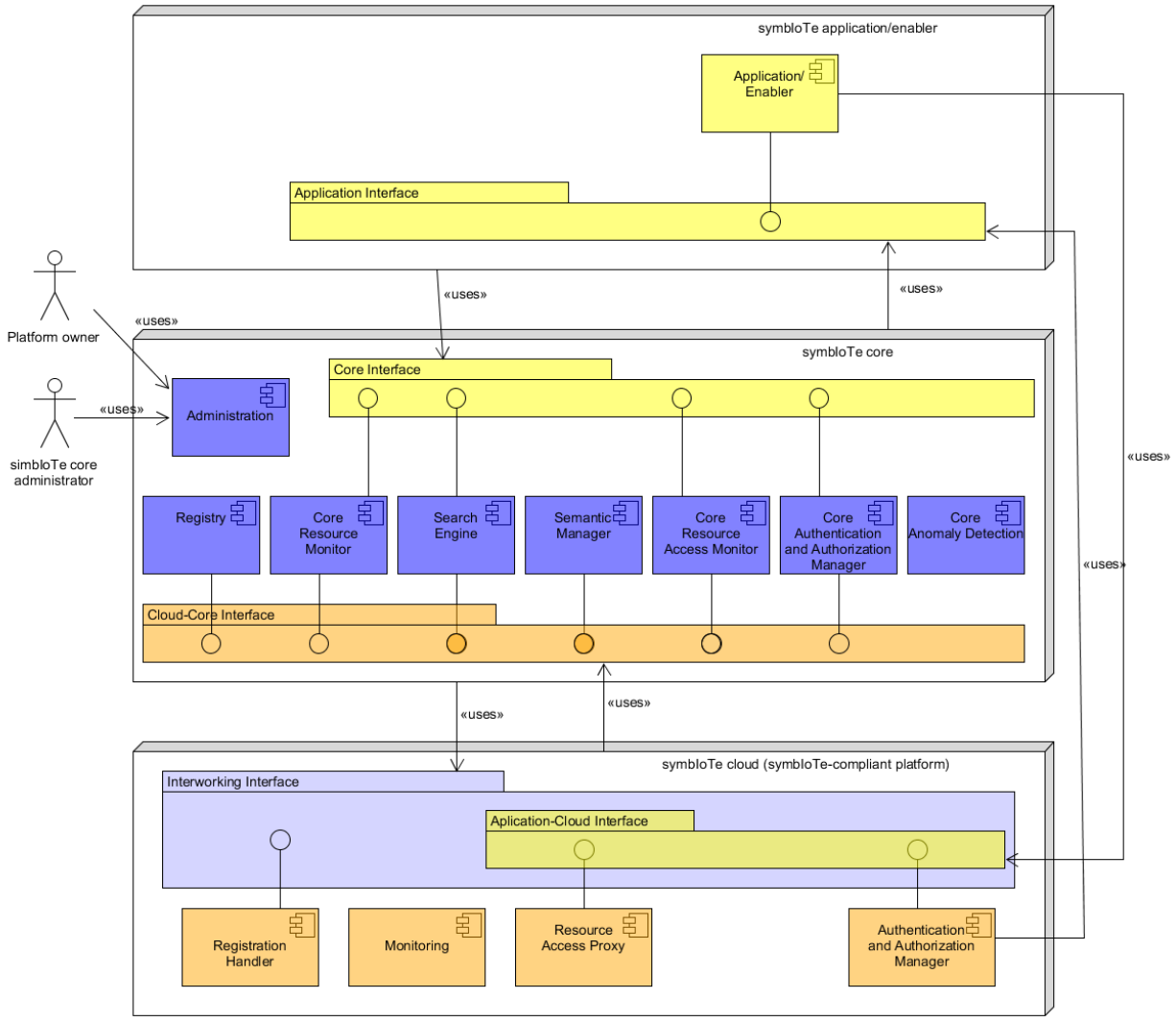

Figure 12 Component diagram for CL1

5.5.2 Communication diagrams

The functionalities defined for symbIoTe CL1 are the following:

- Platform registration/unregistration/update;
- Resource registration/unregistration/update;
- Search;
- Access to resources; and
- Monitoring.

Hereafter all functionalities are presented in the form of UML communication diagrams, with detailed description of the exchanged messages. Figure 13 shows generic legend for messages used in the diagrams.

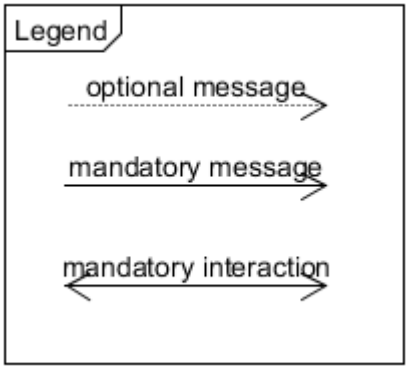

Figure 13 Legend – messages used in the following diagrams

5.5.2.1 Platform registration

IoT platform owner executes platform registration by using the administrative web application.
An enabler has two roles in the registration process:

- Platform role - registers enabler the same way as another IoT platform for exposing composite IoT services;
- Application role - registers enabler the same way as an Application for using IoT services which are searchable and exposed by symbIoTe Core Services.

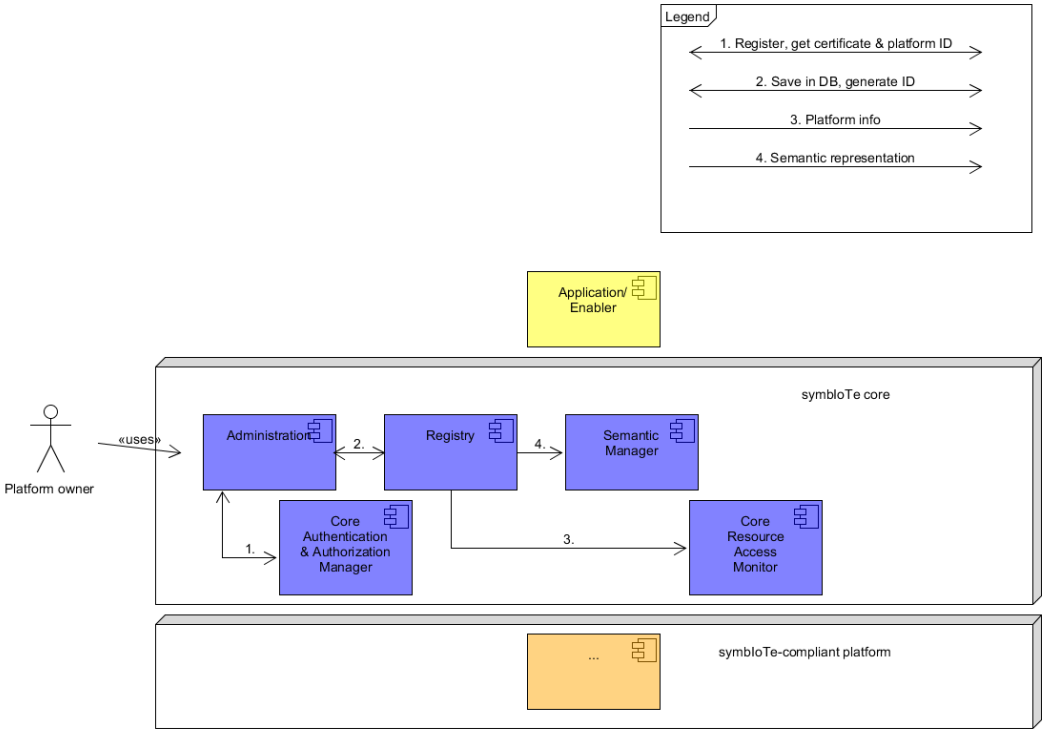

Figure 14 Platform registration

Detailed description:

- Message 1: IoT platform owner or Enabler owner sends a request for symbIoTe usage through the Administration web application. The user needs to create an account if it is not created previously. The request is either for a trial or for normal registration. In the response, the owner receives a certificate.
- Message 2: Request for registration is forwarded to the Registry which generates platform/Enabler ID.
- Message 3: Platform/Enabler information is sent to Core RAM.
- Message 4: Platform/Enabler information is stored in Semantic Manager. It can optionaly define its own PIM.

After the registration process, the IoT platform owner or the Enabler owner has acquired a certificate to access symbIoTe Core services, and a unique ID by which the platform or Enabler can be identified. The owner can subsequently configure the platform to become L1 Compliant.

5.5.2.2 Resource registration, unregistration and modification

IoT platform owner needs to register resources to symbIoTe Core Services in order to become discoverable to third party application developers or to the Enablers. Platforms also need to be able to unregister and modify the exposed resources. Enablers can also register the resources in the same way as IoT platforms.

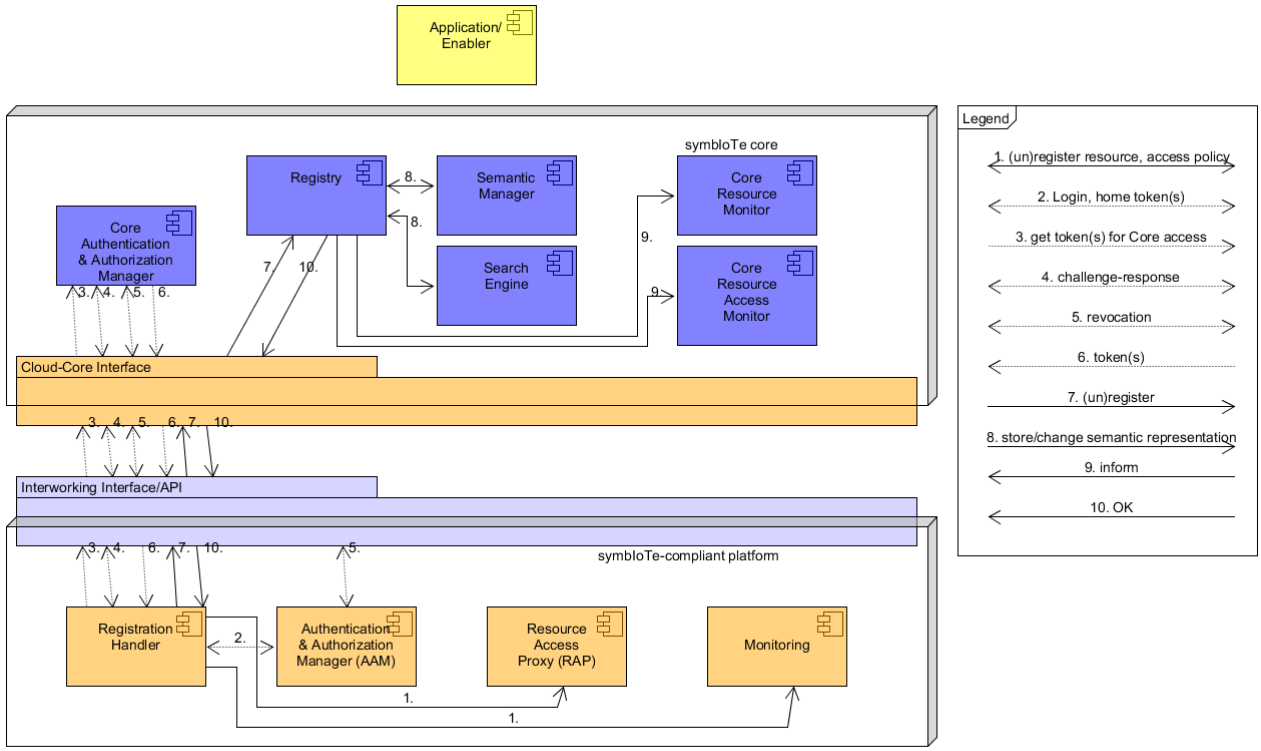

Figure 15 Resource registration, unregistration and modification

Detailed description:

- Message 1: Registration Handler notifies Resource Access Proxy and Monitoring in the same IoT platform of the resource(s) that will be registered/unregistered/modified through symbIoTe. It is used to register the resource(s) on the Resource Access Proxy, along with the access policy to access it, and to schedule availability checks through Monitoring.
- Message 2 (optional): Registration Handler performs a login to obtain home token(s) from its AAM. If the Registration Handler already has a valid token, this step is not needed.
- Message 3 (optional): Registration Handler performs a login to obtain token(s) from Core AAM. If the Registration Handler already has valid token(s), this step is not needed (as well as Messages 4 to 6).
- Message 4 (optional): challenge-response procedure between Core AAM and Registration Handler.
- Message 5 (optional): check revocation procedure between Core AAM and platform's AAM.
- Message 6 (optional): Core AAM returns token(s).
- Message 7: Resource Handler sends register/unregister/modify request to Registry.
- Message 8: Registry sends resource data to Semantic Manager and to Search Engine. Semantic Manager validates if the data conforms to CIM and optionally PIM, while Search Engine translates the data into RDF and stores/deletes/modifies resource description.
- Message 9: Registry informs CRM and CRAM of the new/deleted/modified resource.
- Message 10: Acknowledgement.

5.5.2.3 Resource search

Application/Enabler uses the Search Engine from symbIoTe Core Services to find the desired resources. Search Engine finds the resources, ranks and filters the results, and forwards the response to the application/Enabler.

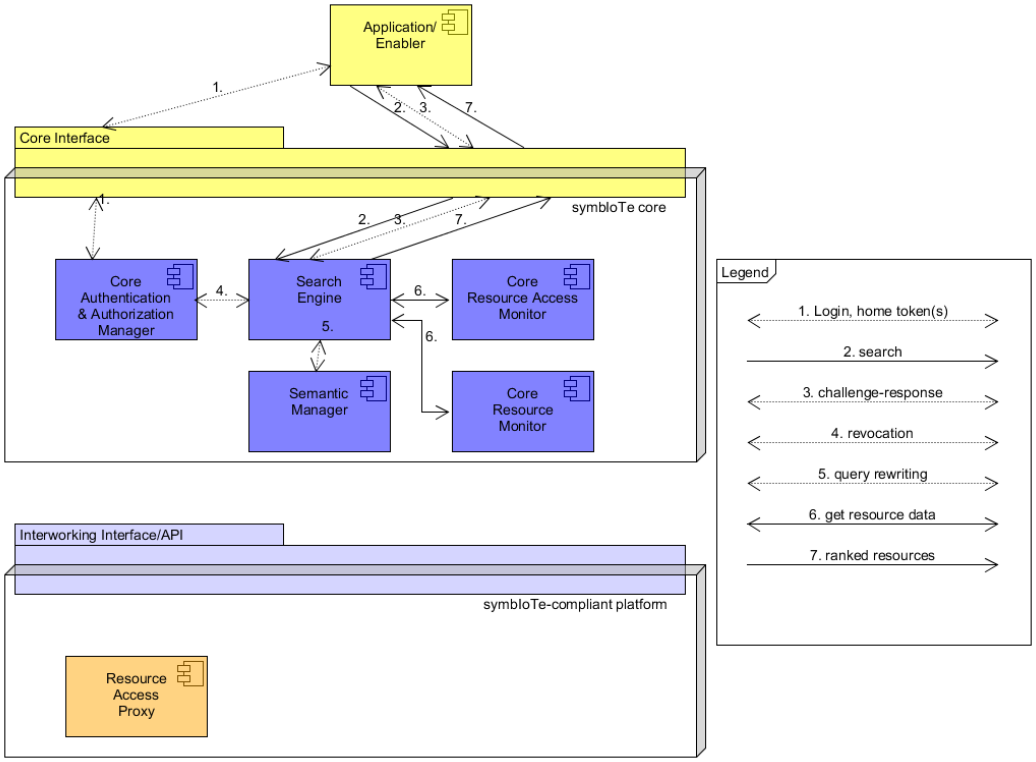

Figure 16 Resource search

Detailed description:

- Message 1 (optional): Application/Enabler logs in to Core AAM, and obtains home token(s). If the Application/Enabler has valid token(s), this step is not needed.
- Message 2: Application/Enabler sends the query with home token(s) to the Search Engine.
- Message 3 (optional): challenge-response procedure between Search Engine and Application/Enabler.
- Message 4 (optional): check revocation procedure between Search Engine and Core AAM.
- Message 5 (optional): Search Engine forwards request to Semantic Manager if query rewriting is needed.
- Message 6: Search Engine gets resource availability data from CRM and resource usage data from CRAM which is used for ranking the results.
- Message 7: Search Engine ranks and filters the results (according to access policies to private resources), and forwards the response to Application/Enabler.

5.5.2.4 Access to resources

After obtaining the list of desired resources, the application/Enabler chooses the ones it wants to access. It receives their URLs from the Core RAM, and in the next steps contacts the RAP at the resources' platform to acquire needed data.

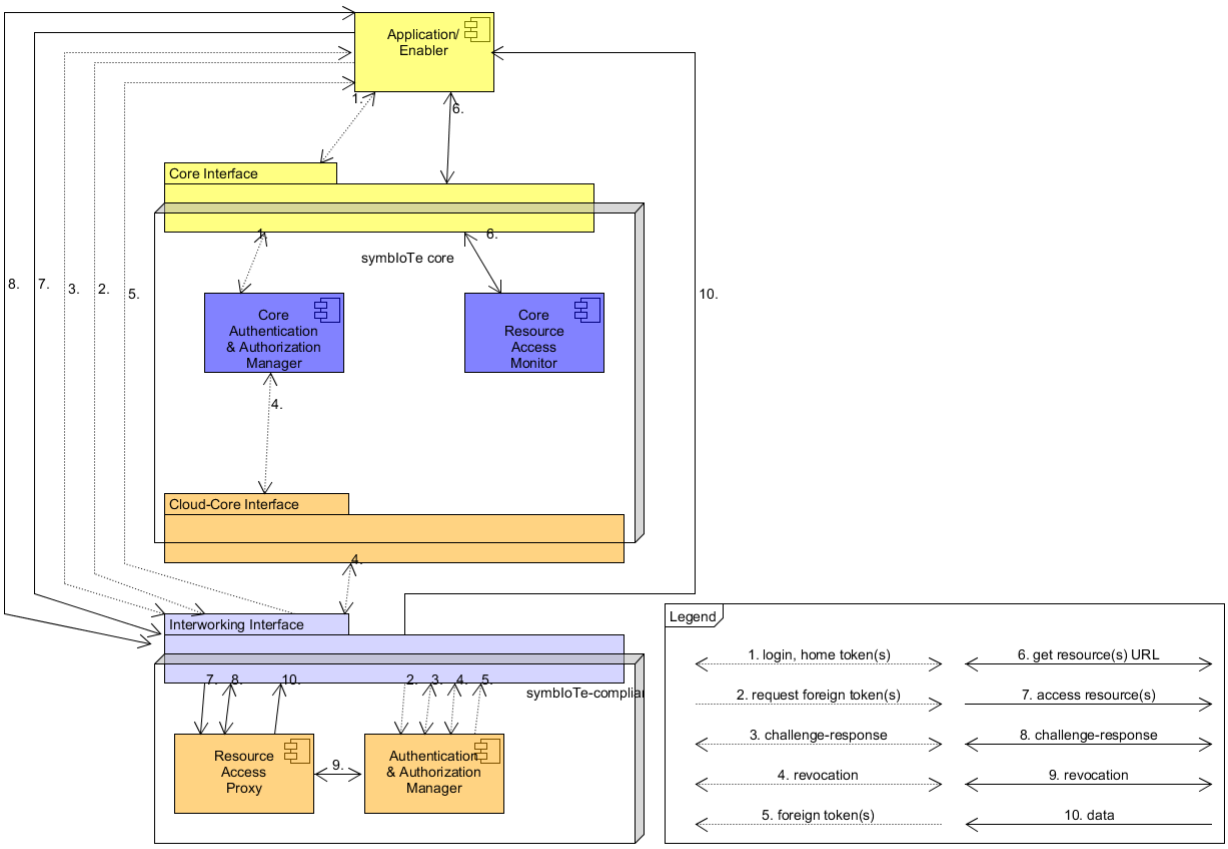

Figure 17 Access to resources

Detailed description:

- Message 1 (optional): Application/Enabler logs in to Core AAM, and obtains home token(s). If the Application/Enabler has a valid token, this step is not needed.
- Message 2 (optional): Application/Enabler requests foreign token(s) from the platform. If the Application/Enabler already has valid foreign token(s), this step is not needed (as well as Message 3 to Message 5).
- Message 3 (optional): challenge-response procedure between AAM and Application/Enabler.
- Message 4 (optional): check revocation procedure between AAM and Core AAM.
- Message 5 (optional): AAM provides foreign token(s).
- Message 6: Application/Enablers requests access to selected resources from Core RAM. Core RAM responds with a list of URLs from where resources can be obtained.
- Message 7: Application/Enablers accesses the selected resources through Resource Access Proxy with home/foreign/guest token(s).
- Message 8: challenge-response procedure between Resource Access Proxy and Application/Enabler.

- Message 9: check revocation procedure between Resource Access Proxy and AAM.
- Message 10: Resource Access Proxy returns the data generated by the resource. A user can request the latest measurement, historical measurements, or create subscriptions or receive data in streams. In case of an actuation request, one of the services offered by a resource is invoked.

5.5.2.5 Monitoring

Monitoring is a scheduled task for checking availability of the registered resources. Upon registration of resources, or certain changes, RH informs Monitor component of the resources needed to be monitored, as shown in . The result is forwarded to CRM so that the status of all results is stored centrally. This information is then used by the Search Engine to recommend resources to applications/Enablers.

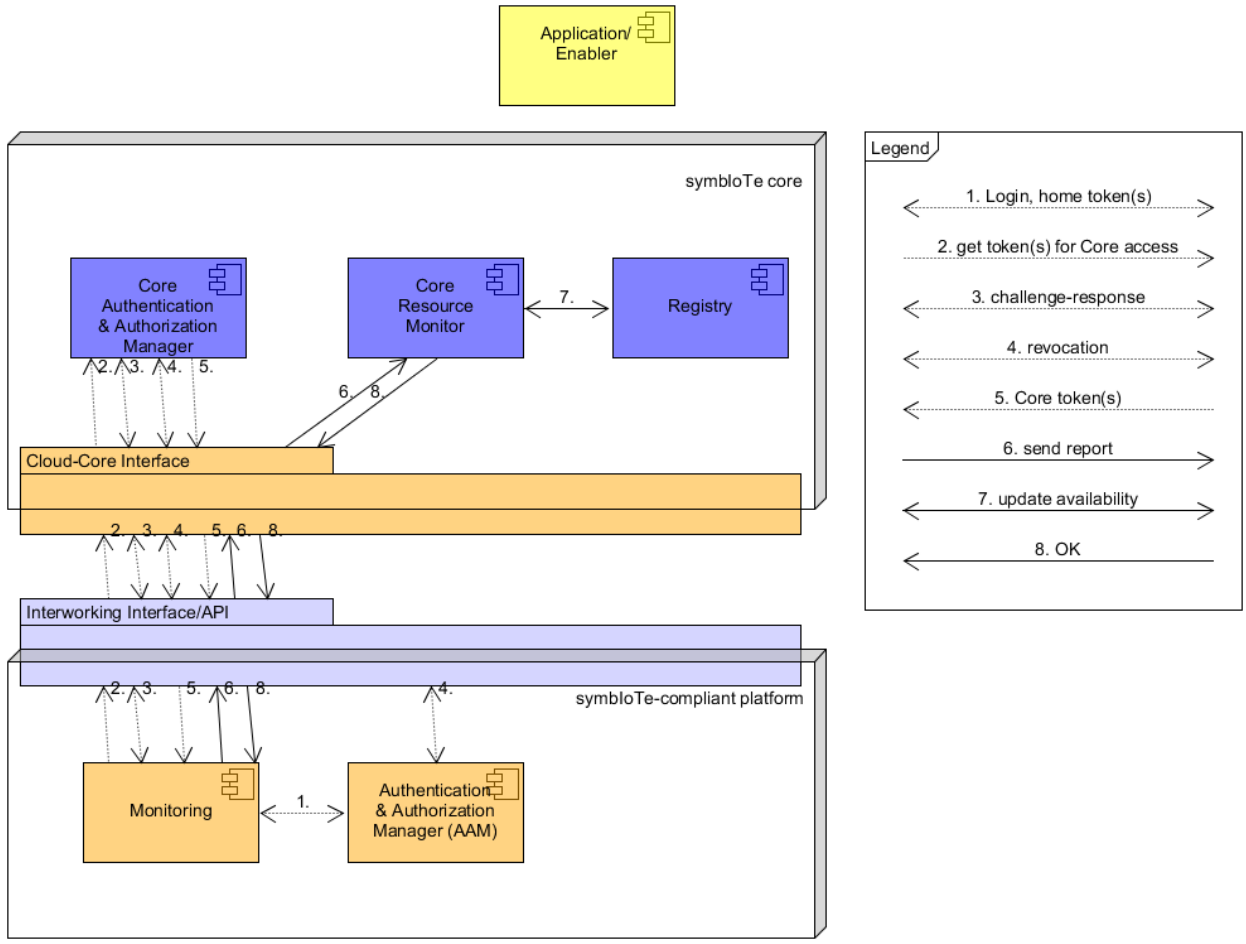

Figure 18 Monitoring resource availability

Detailed description:

- Message 1 (optional): Monitoring performs a login to obtain home token(s) from its AAM. If the component already has valid tokens, this step is not needed.

- Message 2 (optional): Monitoring performs a login to obtain token(s) from Core AAM. If the Monitoring already has valid token(s), this step is not needed (as well as Message 3 to Message 5).
- Message 3 (optional): challenge-response procedure between Core AAM and Monitoring.
- Message 4 (optional): check revocation procedure between Core AAM and platform's AAM.
- Message 5 (optional): Core AAM returns token(s).
- Message 6: Monitoring sends an aggregated report to CRM.
- Message 7: CRM stores changes in Registry.
- Message 8: Acknowledgement.

5.6 Achieving Compliance Level-2 (CL2)

CL2 assumes the creation of IoT platform federations, thus enabling organizational/enterprise interoperability. By forming federations, platforms can securely interoperate, collaborate and share resources according to accepted SLA. CL2 includes the additional functionality compared to CL1, sharing/bartering or trading of resources between platforms. The functionalities provided at this level enables the so-called organizational interoperability.

In this section we identify the components that an IoT platform needs to integrate with existing components in CLD in order to become L2 Compliant, as well as the Core Service components in the APP mainly required for creating federations and reaching an agreement for a platform to join a federation. Since authenticated and authorized access to offered services is vital for an IoT ecosystem, we also include security-related components.

The component diagram of a symbIoTe ecosystem which includes L2 Compliant Platforms is presented in Section 5.6.1. It also identifies component interfaces. Respective communication diagrams specifying CL2 functionalities are included in 5.6.2 and describe a simplified communication flow between these components.

5.6.1 Component diagram

The component diagram with the specified interfaces is shown in Figure 19. For CL2, the symbIoTe system defines the following interfaces:

- **Cloud-Core Interface** used by a symbIoTe Compliant Platform to interact with symbIoTe Core components; and
- **Interworking Interface** used by symbIoTe Core components to interact with symbIoTe Compliant Platform, and used by applications or components of a home platform to interact with another platforms within a federation.

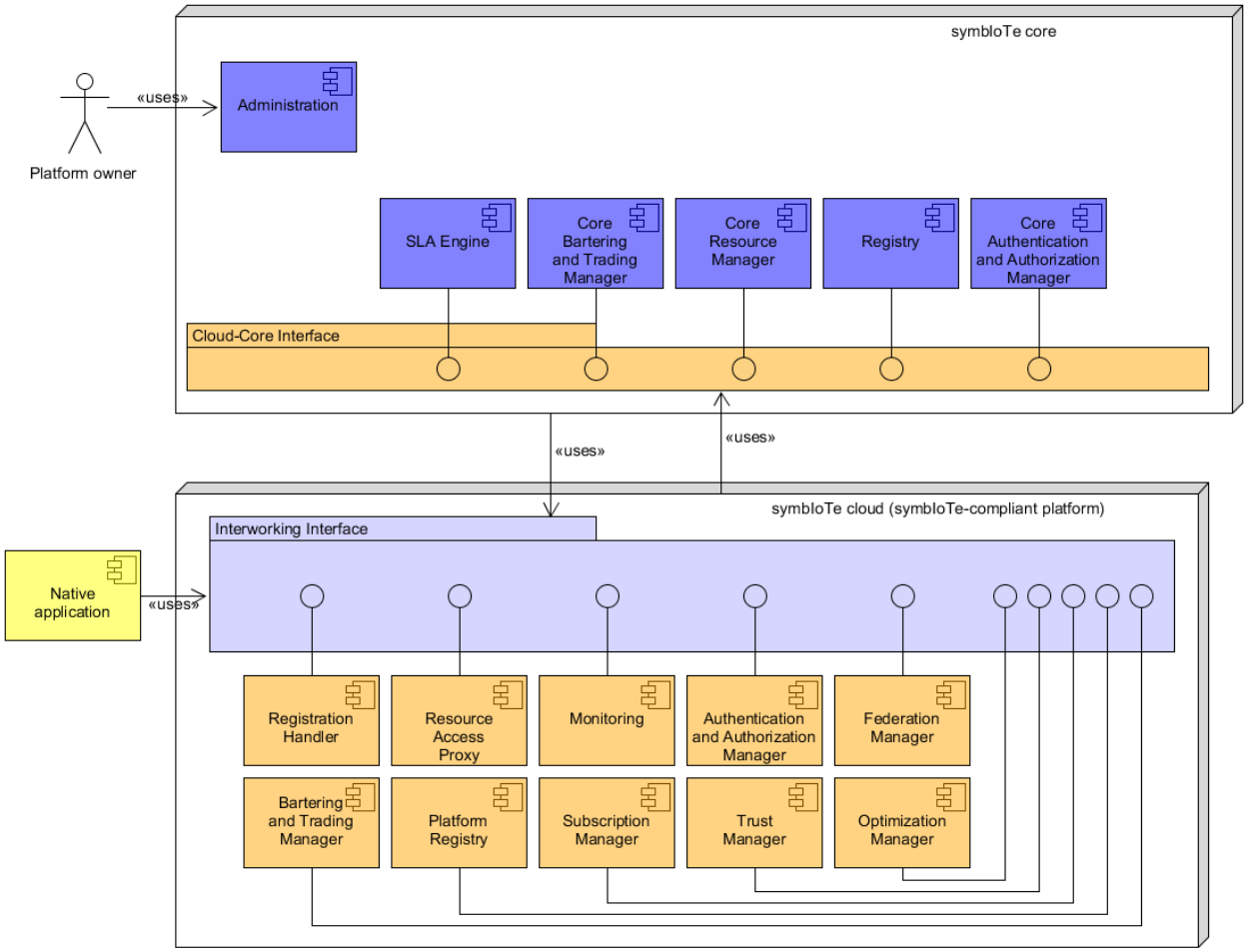

Figure 19 Component diagram for CL2

5.6.2 Communication diagrams

The functionalities defined for symbIoTe CL2 are the following:

- Federation management including SLA handling;
- Monitoring and SLA violation;
- Adding, updating, removing resources to be shared within federation;
- Access to federated resources; and
- Calculation of trust.

Hereafter all functionalities are presented in the form of UML communication diagrams, with detailed description of the exchanged messages.

5.6.2.1 Federation Management

Federation management includes all the interactions performed when an IoT platform wants to join an existing federation, leave the federation, or perform some updates. The updates can also relate to changes in bartering and trading agreements. The Platform

owner uses the GUI in Administration component to manage the federation. All the platforms need to be informed of the changes.

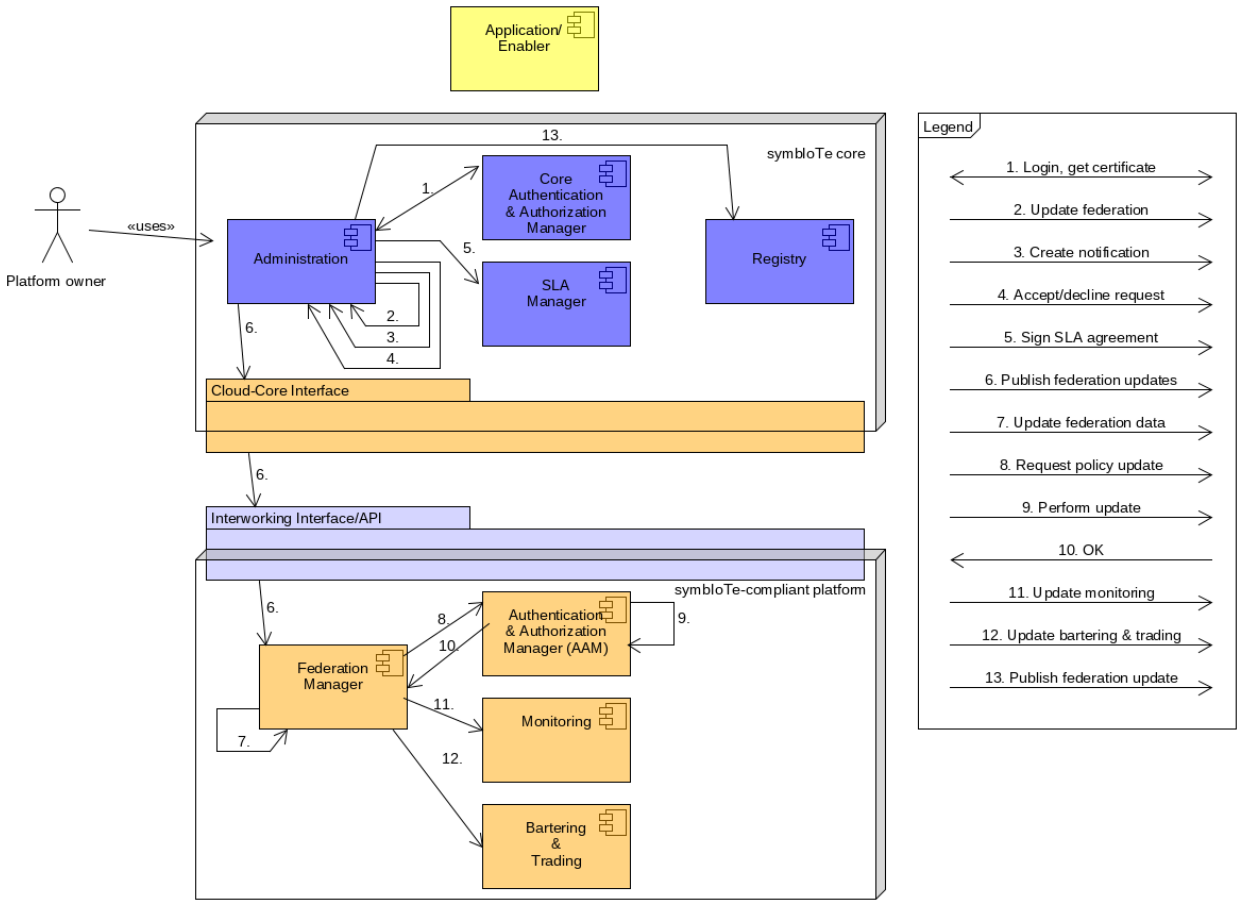

Figure 20 Federation management

Detailed description:

- Message 1: The IoT platform owner gets access to the Administration GUI by creating an account (if not already created before). In the response, the owner receives a certificate for communication with symbIoTe system.
- Message 2: The IoT platform owner updates the status of the federation within the Administration component.
- Message 3: The Administration sends a notification to the other IoT platform owners affected by the federation updates. The example of these federation updates is when Platform owner demands joining or leaving the federation.
- Message 4: The other IoT Platform owners are given the possibility to accept, decline or just take notice of the previously performed changes within the federation.
- Message 5: Depending on the nature of the operation, a SLA agreement is either signed, when joining a federation, or removed, when leaving it.

- Message 6: The Administration informs the Federation Manager, which is located within the IoT platforms belonging to the affected federations, about the update.
- Message 7: The Federation Manager updates the federation data as needed.
- Message 8: The Federation Manager informs the Authentication & Authorization Manager about the federation updates with the request to update the affected token issuing policies.
- Message 9: The Authentication & Authorization Manager updates the policies as needed.
- Message 10: The Authentication & Authorization Manager confirms the update of the pertained token issuing policies.
- Message 11: The Federation Manager informs the Monitoring component about resource rules to monitor. These rules will be based on the SLA agreement.
- Message 12: The Federation Manager informs the Bartering and Trading component about the federation updates so that it can take actions according to it.
- Message 13: The Administration informs the Registry about the relevant updates undergone by the federation.

5.6.2.2 Monitoring and SLA violation

Monitoring of the federation resources is necessary to be able to verify if SLAs are being respected. Monitoring component in each federated platform sends the data to Core RM, which forwards the data to SLA Manager to check for potential violations.

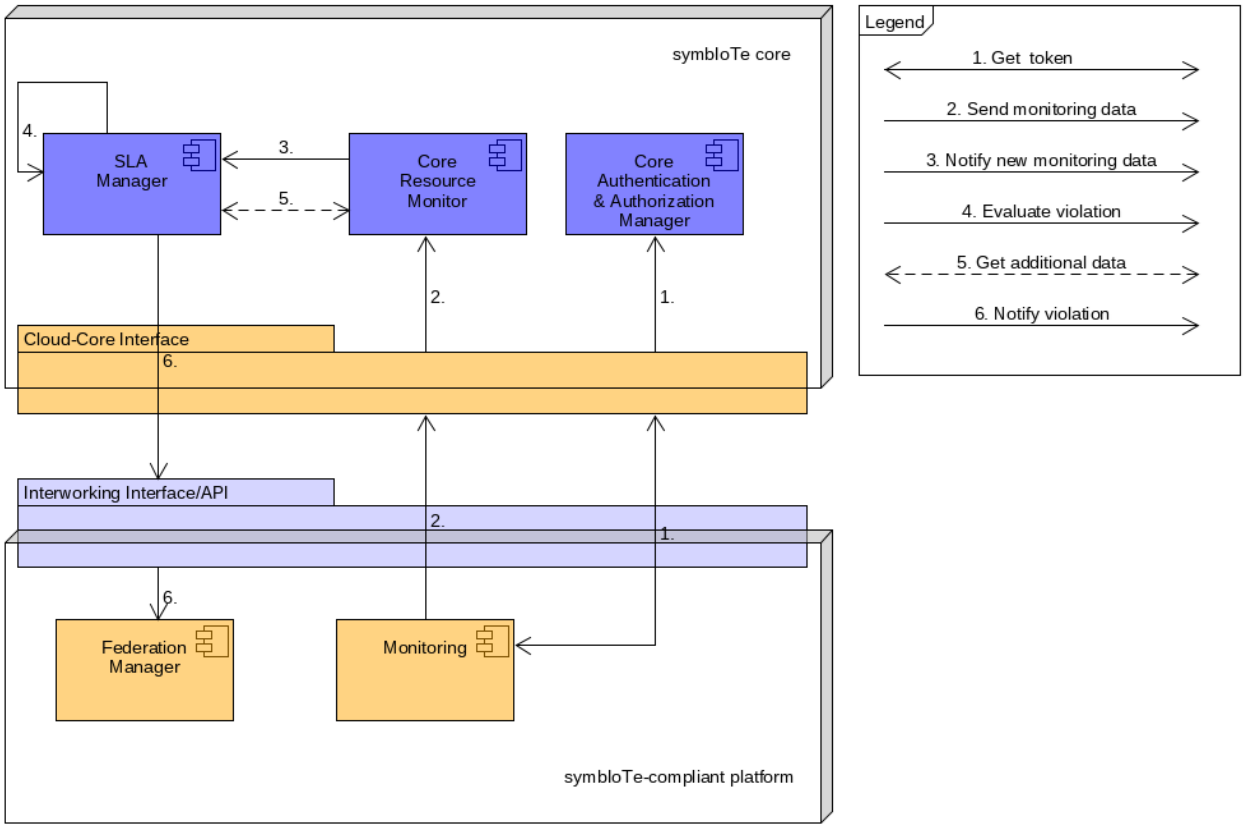

Figure 21 Monitoring and SLA violation

Detailed description:

- Message1: The monitoring component obtains token(s) from the Core AAM.
- Message 2: The monitoring component sends metrics to the Core Resource Monitor.
- Message 3: The Core Resource Monitor sends some of this metrics to the SLA Manager for evaluation.
- Message 4: With this metrics, the SLA manager checks if it means a violation of any signed SLA agreement of the source platform.
- Message 5: The SLA manager might need extra or aggregated data which will get from the Core Resource Monitor.
- Message 6: In case of a violation, it will inform the Federation Manager of each platform affected by the violation.

5.6.2.3 Add, update, and remove resources within federation

Each federated platform selects the resources it wants to expose in symbIoTe federation. It has the ability to add selected resources to the federation, to update them or remove from federation. Registration Handler performs all registration actions. The registered resources (both home and foreign) are stored in Platform Registry. Subscription Manager notifies all the other platforms within the federation of the changes.

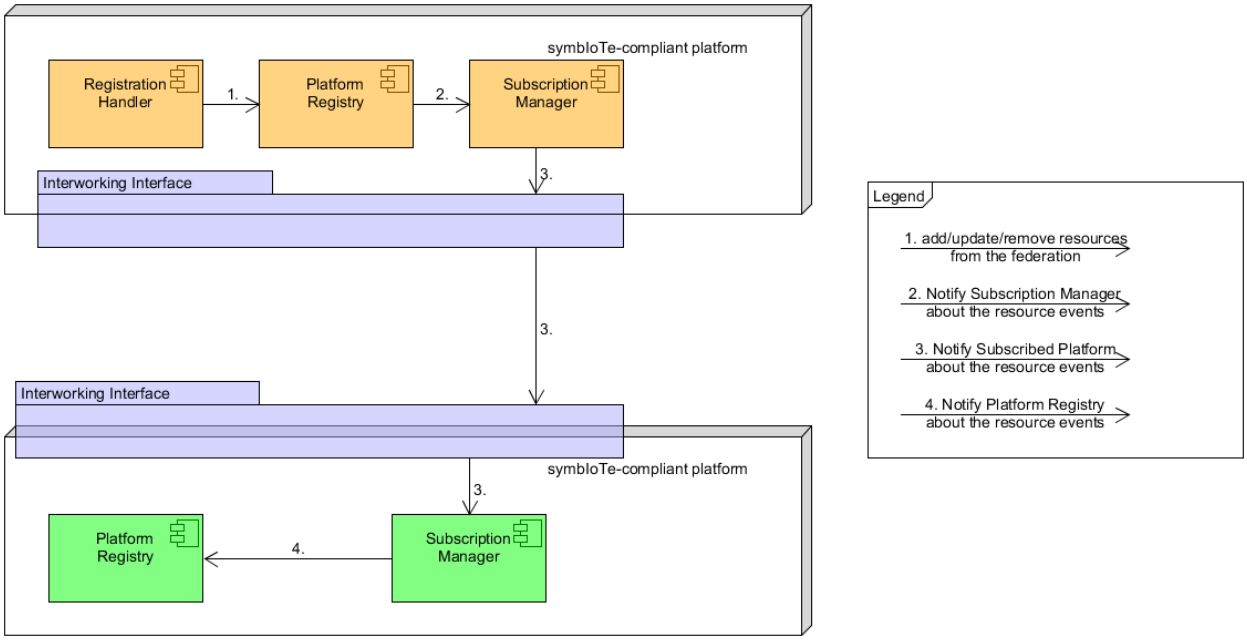

Figure 22 Add, update and remove resources in a federation

Detailed description:

- Message 1: Registration Handler wants to add/remove/update a resource to the federation, resource information is sent to Platform Registry.
- Message 2: Resource information is forwarded to Subscription Manager.
- Message 3: Subscription Manager notifies subscribed federated platforms of the new/removed/updated resource.
- Message 4: Subscription Managers of the platforms within federation forward the notification to their own Platform Registries where resource information is stored/deleted/updated.

5.6.2.4 Access to resources from a federated IoT platform

Native application finds the resource information shared within the federation in its own Platform Registry. When it wants to access resources from federated platforms, it does that through its Resource Access Proxy. Its home RAP then contacts the RAP of the platform that stores resources, and acquires wanted data. Bartering and Trading Manager is included in the process to be able to check the sharing or trading agreements between platforms.

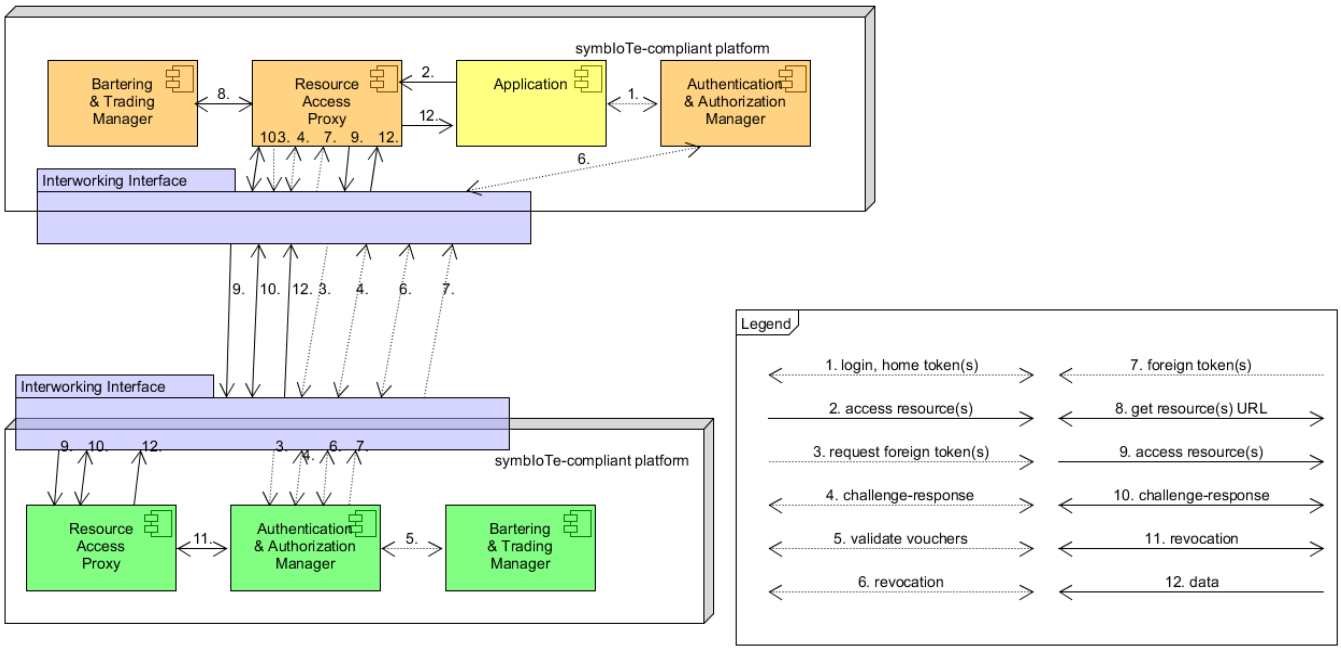

Figure 23 Access to federated resources

Detailed description:

- Message 1 (optional): Application logs into the AAM of its platform, and obtains home token(s). Optional if the application already has valid token(s).
- Message 2: Applications requests access to data from a federated IoT platform.
- Message 3 (optional): Application requests foreign token(s) from the federated IoT platform. Optional, if the Application already has foreign token(s), as are the messages 4-7.
- Message 4 (optional): challenge-response procedure between home RAP and federated platform AAM.
- Message 5 (optional): AAM checks if the home platform has valid vouchers to access its resources.
- Message 6 (optional): check revocation procedure between foreign AAM and home AAM.
- Message 7 (optional): foreign AAM provides foreign token(s).
- Message 8: home RAP requests access to selected resources with foreign token(s). Bartering & Trading Manager responds with a list of URLs from where resources can be obtained.
- Message 9: home RAP accesses the selected resources through foreign RAP with foreign token(s).
- Message 10: challenge-response procedure between foreign RAP and home RAP.
- Message 11: check revocation procedure between foreign RAP and foreign AAM.

- Message 12: foreign RAP returns the data. Home platform can request the latest measurement, historical measurements, create subscriptions or receive data in streams.

5.6.2.5 Calculation of Trust

Trust Manager calculates the trust level for resources offered by each platform to define platforms' trust level and if its resources are trustworthy. The data for calculating trust is obtained from Core Bartering and Trading component, Core Resource Monitor, and Monitor at the CLD level.

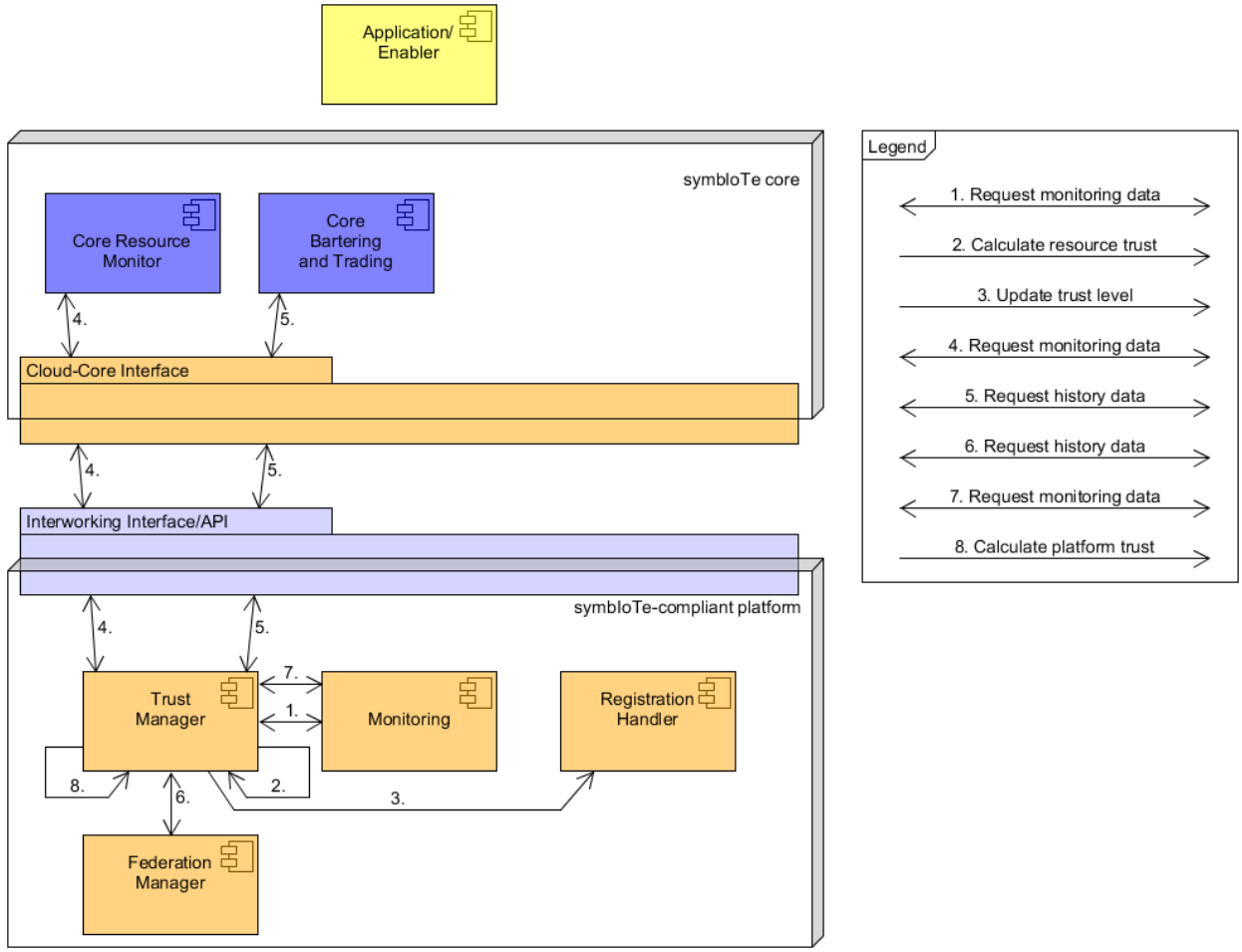

Figure 24 Calculation of Trust

Detailed description:

- Message 1: The Trust Manager requests data from the Monitoring with regard to the resources that the IoT platform offers within the federation.
- Message 2: The Trust Manager calculates the resource trust of the resources that are offered by the IoT platform within the federation.
- Message 3: The Trust Manager shares with the Registration Handler the updated resource trust level.

- Message 4: The Trust Manager requests from the Core Resource Monitor historical monitoring data concerning the resources offered by a specific IoT platform.
- Message 5: The Trust Manager requests from the Core Bartering and Trading historical data concerning the transactions in which a specific IoT platform has been involved.
- Message 6: The Trust Manager requests from the Federation Manager data related to a specific IoT platform.
- Message 7: The Trust Manager requests data from the Monitoring regarding the resources offered by a specific IoT platform within the federation.
- Message 8: The Trust Manager calculates the platform trust level for a specific IoT platform.

5.7 Achieving Compliance Level-3 (CL3) and Level-4 (CL4)

As already stated in Section 3.3, CL3 assumes that platforms integrate symbIoTe components within their SSPs to simplify the integration and dynamic reconfiguration of IoT devices within local spaces. CL4 provides support for roaming devices registered with a home platform that maintain their unique identifiers while moving through different SSPs. Devices are dynamically reconfigured within an SSP, so that every new device is reconfigured on the fly to become part of an SSP deployment within this local environment.

An L3 compliant SDEV is able to move from one SSP to another seamlessly, i.e. it is automatically reconfigured and re-annexed to a SSP. In particular, when visiting a "foreign" environment the SDEV will be able to use resources in the surrounding infrastructure, and offer its own resources to others, provided that the required SLAs are in place. In L3 mode, the SDEV is reconfigured as a new device each time it moves from one SSP to another.

On the other hand, L4 compliance mandates that a SDEV connecting to a new SSP maintains the association with its "home" platform (a record of L4 device SSP "location" and temporary URI is maintained in the Platform Registry in CLD), behaving as a roaming (as opposed to nomadic) device. This also implies that the L4 SDEV is always identifiable and traceable as it moves between SSPs.

In this section we identify the components and communication diagrams related to the SSP and CL3/CL4. Since parts of CL3 and CL4 are still in specification phase, some functionalities, especially relating to security, might be modified and will be reported in the deliverables within WP4.

5.7.1 Component Diagram

Figure 25 shows the component diagram related to CL3 and CL4, where an application interacts with devices inside or outside an SSP directly as well as with service/interface exposed by the SSP[9]:

[9] Only a subset of the services/interfaces are shown in this figure.

- **SSP Services/API** is used by the application/enabler and/or the upper modules of the symbIoTe architecture (CLD components) to interact with the S3 Middleware of the SSP.

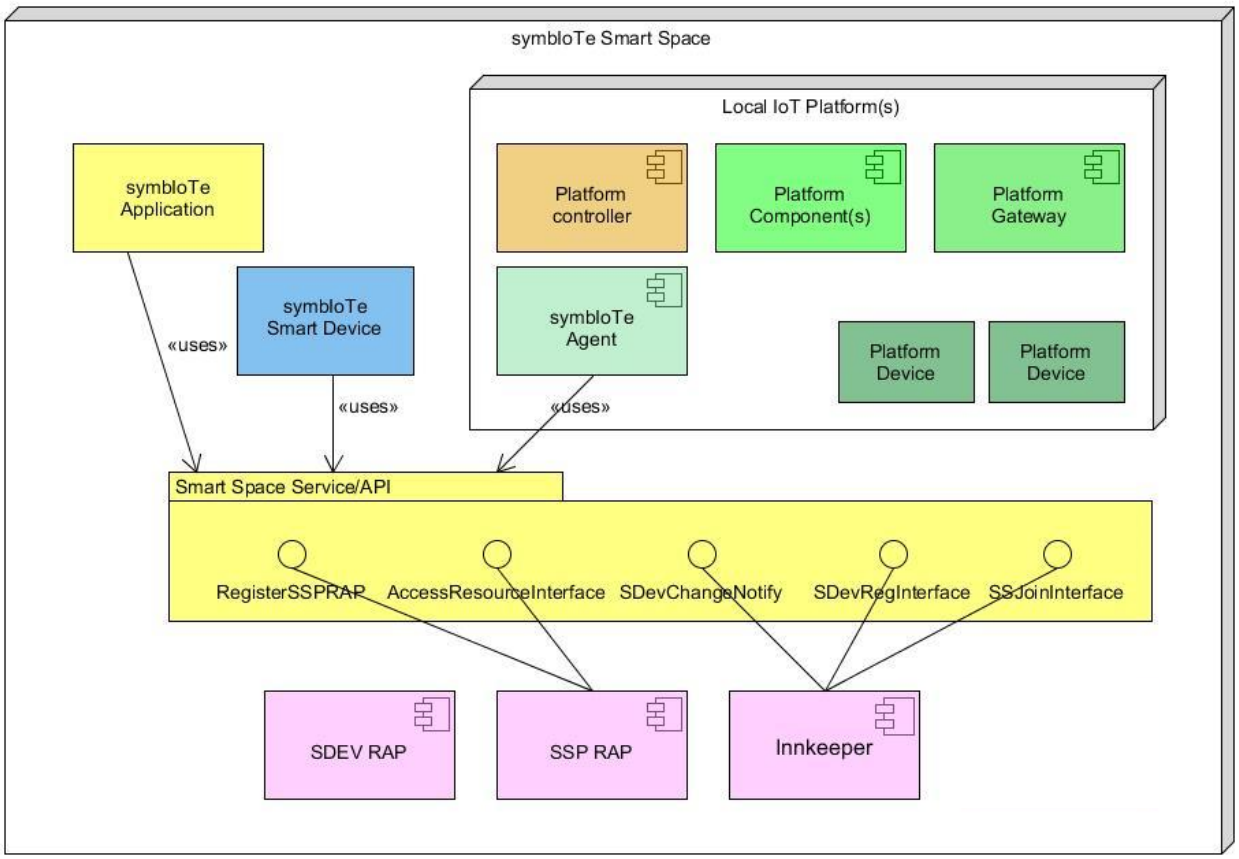

Figure 25 Component diagram for CL3 and CL4

5.7.2 Communications diagrams

The main identified functionalities for CL3 and CL4 are the following:

- App joins the SSP;
- SDEV (either L3 or L4) joins the SSP;
- Local platform joins the SSP;
- Access to SSP resources from within the SSP; and
- Access to SSP resources from outside the SSP.

5.7.2.1 Application joins SSP

The application (e.g., on a smart phone) joins an SSP to be able to access different IoT devices already associated with it. Firstly, it recognizes symbiotic Wi-Fi SSID, and then tries to access the Inkeeper to register. Afterwards, the application is able to access all local resources through SSP RAP.

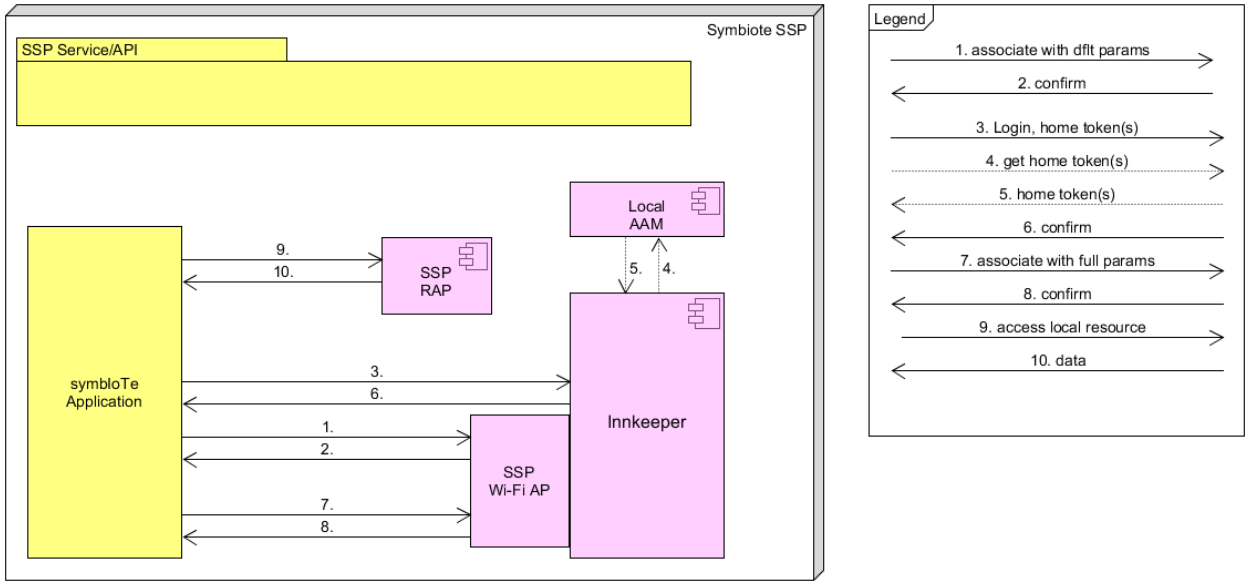

Figure 26 Application joins SSP

Detailed description:

- Message 1: application recognizes the symbiotic Wi-Fi SSID and requests access to the Innkeeper with limited connection.
- Message 2: acknowledgement for the first connection.
- Message 3: application contacts the Inkeeper to register with the SSP.
- Message 4 (optional): If the Application does not have token(s) to access SSP, it logs in to the local AAM to acquire them.
- Message 5 (optional): local AAM issues the token(s).
- Message 6: Inkeeper sends back the confirmation to the application that it can access the SSP.
- Message 7: Application associates to the SSP with possibility to access local resources.
- Message 8: Confirmation.
- Message 9: application requests access to local resources through SSP RAP.
- Message 10: SSP RAP responds with requested data.

5.7.2.2 SDEV (L3 or L4) joins SSP

SDEV tries to access the SSP by registering to the Inkeeper. Optionally, if there is connectivity to the Internet, it also registers to the Core. Note that in case of L4 roaming devices, a notification to the Platform Registry is also needed since it stores records about platform roaming resources. Afterwards, SDEV configures access to itself through SSP RAP, RAP Gateway and RAP in CLD.

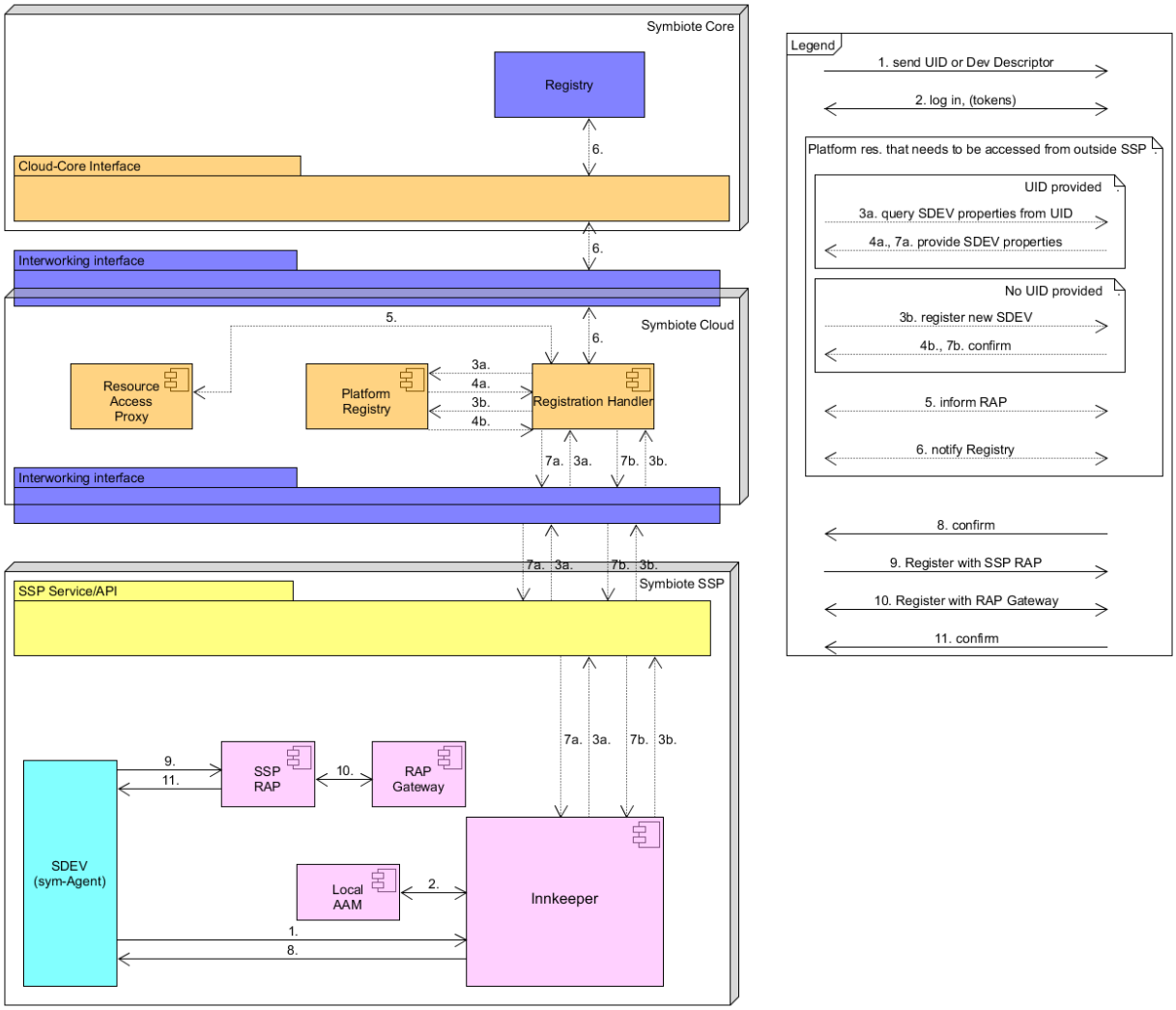

Figure 27 SDEV joins SSP

Detailed description:

- Message 1: SDEV sends a request to the Innkeeper to join the SSP.
- Message 2: Inkeeper checks if SDEV has the rights to access SSP.
- Message 3 (optional): if SDEV needs to be accessed from outside the SSP, it has to register to CLD and/or APP. If SDEV has sent its UID, then this is a SDEV with roaming capability, and its new location needs to be registered (Message 3a). If not, it is registered as a new device (Message 3b). Registration request goes through Registration Handler to the Platform Registry which stores properties of registered SDEVs.
- Message 4 (optional): Platform Registry responds with SDEV properties (message 4a), or confirms successful SDEV registration (message 4b)
- Message 5 (optional): Registration Handler informs RAP of the new location of SDEV so that L1 and L2 Platforms or applications are able access it.
- Message 6 (optional): Registration Handler registers SDEV to the Core so that L1 applications are able to find it.

- Message 7 (optional): Registration to APP and CLD is successful, SDEV properties of already registered device (message 7a) or just confirmation (message 7b) is sent to the Inkeeper.
- Message 8: Inkeeper confirms that SDEV has joined the SSP
- Message 9: SDEV registers its resources to SSP RAP
- Message 10: SSP RAP registers the same information to the Gateway RAP so that applications from outside SSP can access this SDEV.
- Message 11: confirmation.

5.7.2.3 Local platform joins SSP

Local platform can join SSP through sym-Agent in a similar way as a Smart Device. Firstly, it registers to the local Inkeeper. Optionally, it can register to the Registry in Core Services so that it can be discovered by L1 applications. Note that this diagram only shows the registration of the local platform. To register its resources, the platform's sym-Agent needs to follow the steps already defined for the case when SDEV joins the Smart Space, shown in Figure 27.

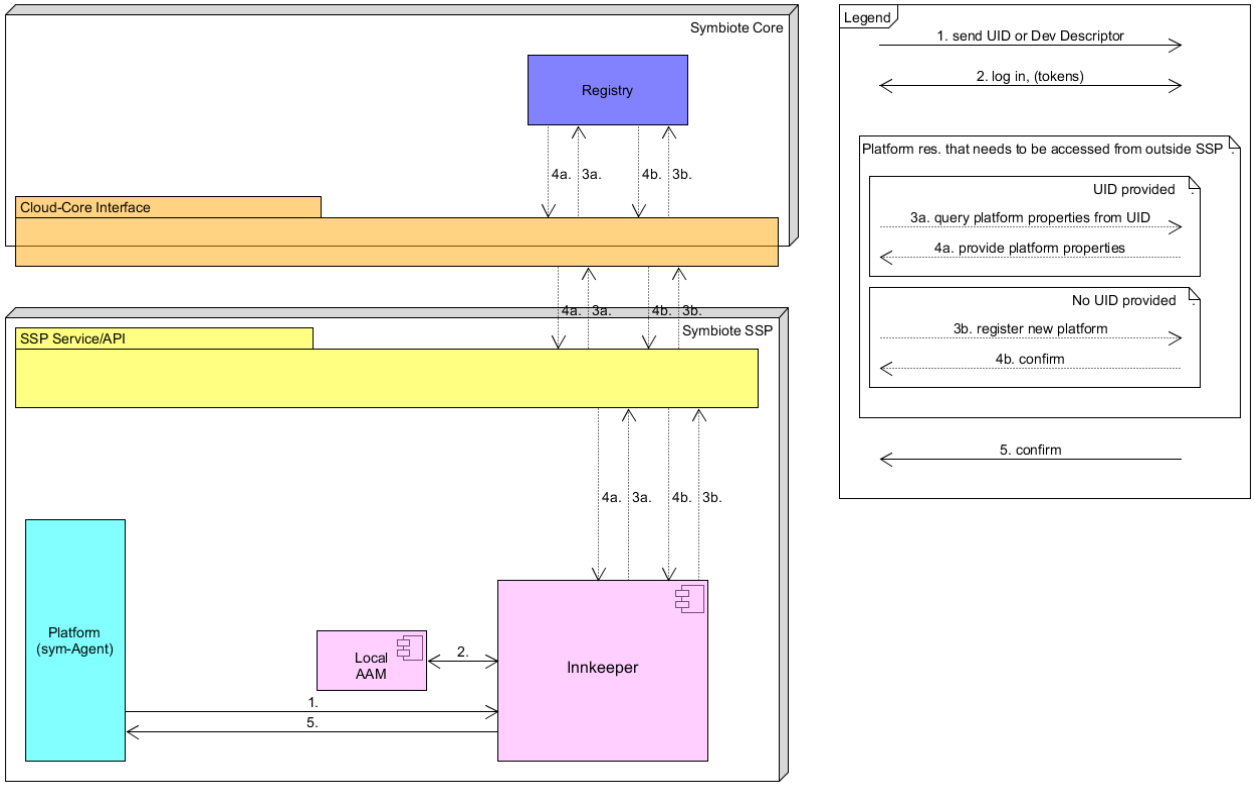

Figure 28 Local platform joins SSP

Detailed description:

- Message 1: sym-Agent sends a request to the Inkeeper to join the SSP.
- Message 2: Inkeeper checks if sym-Agent has valid tokens.

- Message 3 (optional): If the platform wants to be accessed from outside SSP, it registers to Registry in symbIoTe Core. If platform is already registered, the Inkeeper needs to provide platform properties (Message 3a). Otherwise, it registers a new platform (Message 3b).
- Message 4 (optional): Registry responds with platform properties for the platform which sent its UID (Message 4a), otherwise it sends a message about a successful registration (Message 4b).
- Message 5: Confirmation to the sym-Agent.

5.7.2.4 Access to SSP resources from within the SSP

SSP resources can be accessed from within the SSP by and application, Smart Device, or local platform. The access goes through SSP RAP component.

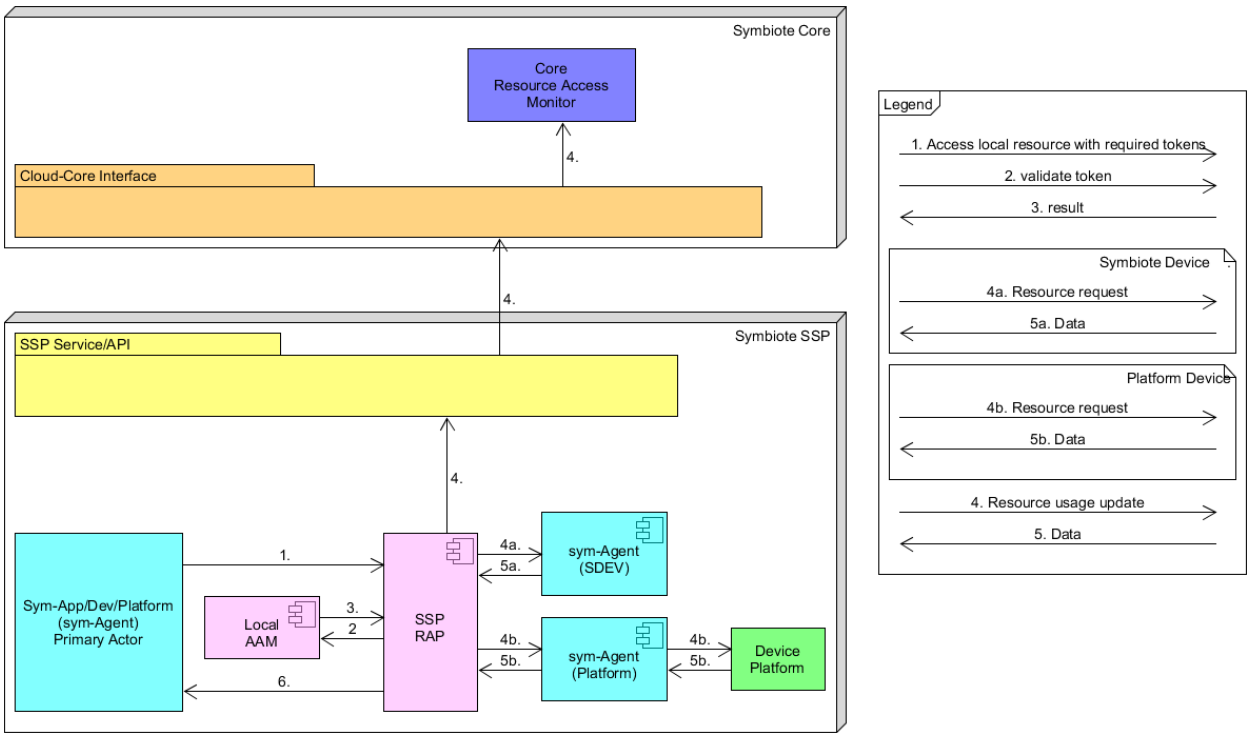

Figure 29 Access to SSP resources from within the SSP

Detailed description:

- Message 1: symbIoTe application or platform tries to access SSP resource through SSP RAP with corresponding token(s).
- Message 2: Local AAM checks if the token(s) are valid.
- Message 3: result of the verification.
- Message 4: If the tokens are valid, and if sym-Agent has access rights for the wanted resources, SSP RAP requests resource from a corresponding sym-Agent, either sym-Agent for SDEV or sym-Agent for local IoT platform. Additionally, SSP RAP sends information about the resource usage to Core RAM in APP.

- Message 5: SSP RAP receives response from SDEV (Message 5a) and local IoT platform (Message 5b).
- Message 6: SSP RAP responds with the wanted resources.

5.7.2.5 Access to SSP resources from outside the SSP

An application outside the SSP can access Smart Devices and local IoT platform resources through Interworking Interface and RAP component in CLD. The request is forwarded to RAP Gateway and SSP RAP in SSP.

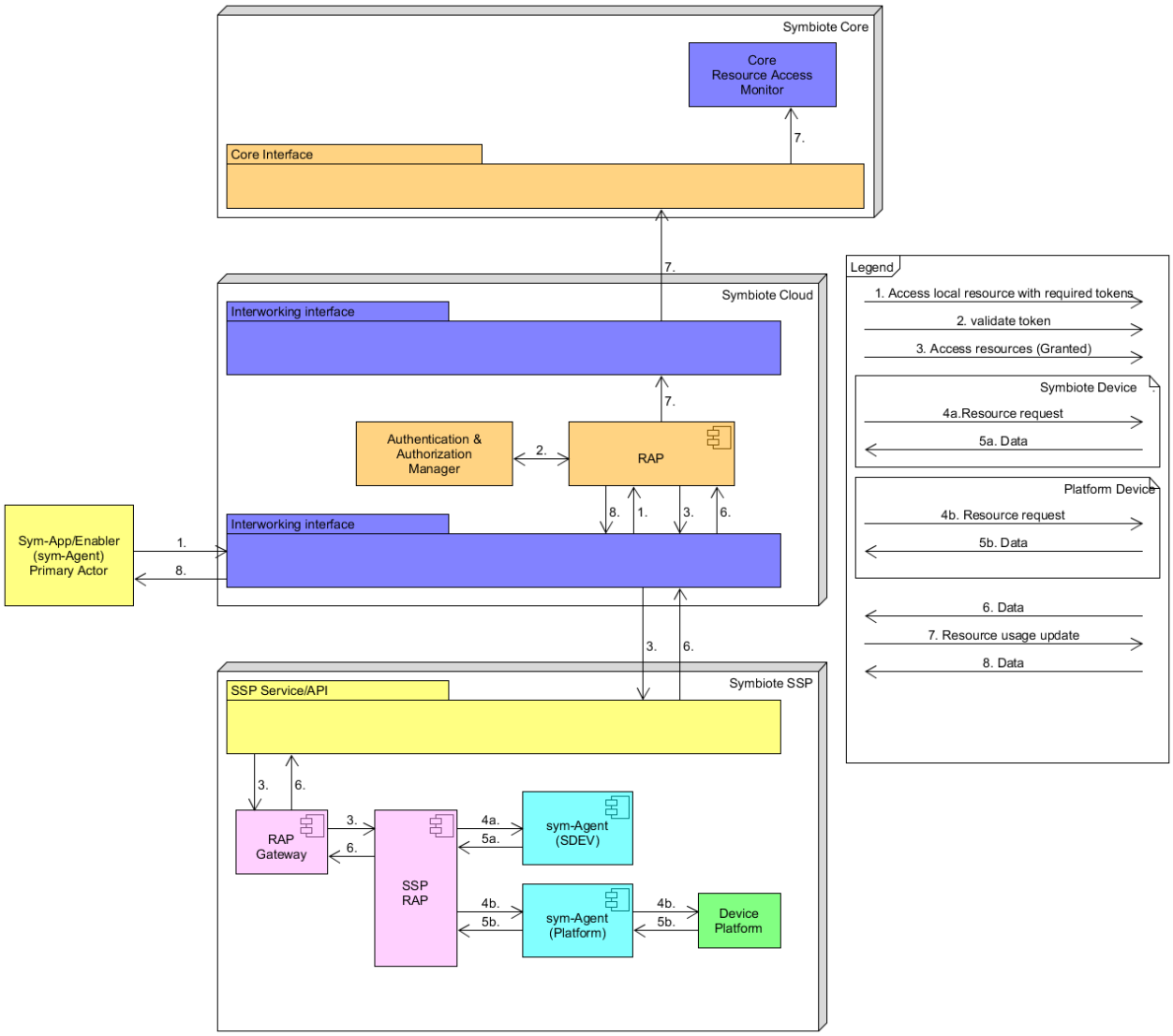

Figure 30 Access to SSP resources from outside the SSP

Detailed description:

- Message 1: application contacts RAP in CLD with token(s).
- Message 2: RAP requests token validation from AAM.
- Message 3: If the validation is successful, RAP forwards the request to RAP Gateway, which then forwards it to SSP RAP.

- Message 4: SSP RAP requests resource from a corresponding sym-Agent (sym-Agent for SDEV or sym-Agent for local IoT platform).
- Message 5: SSP RAP receives response from SDEV (Message 5a) and local IoT platform (Message 5b).
- Message 6: response is forwarded through RAP Gateway to RAP in CLD
- Message 7: RAP sends information about the resource usage to Core RAM in symbIoTe Core.
- Message 8: data is forwarded to the application.

6 State of the Art Overview and Reference to symbIoTe

This section presents the reference architectures defined by standardization authorities as well as projects and platforms with similar goals as symbIoTe. We put them into the context of the symbIoTe architecture and provide mappings where applicable. IoT platforms contributed by symbIoTe partners are also mentioned, with plans for their integration within the future symbIoTe-enabled IoT ecosystem. Finally, we conclude the section with a short summary of the symbIoTe positioning in the current IoT ecosystem.

6.1 Reference architectures

Defining the reference architecture for IoT has been the focus of various organizations and projects. Hereafter we present a selected list or relevant initiatives (AIOTI, oneM2M, Web of Things, OGC), specific reference models (Industrial Internet Reference Architecture, Reference Architecture Model Industrie 4.0, ISO/IEC IoT Reference Architecture), and projects (IoT-A) in order to put them in relation to the symbIoTe architecture.

6.1.1 AIOTI

The Alliance for Internet of Things Innovation (AIOTI) consortium, initiated by the European Commission, brings together stakeholders across the IoT universe. AIOTI has developed a High Level Architecture (HLA) for IoT [1] that serves as the basis for discussion within AIOTI. Due to its generic form, the architecture can be used as reference architecture for IoT platforms. An overview of AIOTI HLA is given in Appendix in the deliverable D1.2, while hereafter we concentrate on the mapping of the AIOTI layers to the symbIoTe architecture.

Figure 31 depicts a mapping between the symbIoTe architecture and AIOTI HLA. The Application Layer of AIOTI HLA corresponds to an Application or Enabler within the symbIoTe architecture. The Application Layer consists of one or more Application Entities that can be considered as a single Application/Enabler entity. Also, multiple instances of the Application Entity can be built-in into a single symbIoTe Application/Enabler (e.g. an application that uses an enabler is also an Application Entity). The IoT layer, as defined by AIOTI, stretches through the symbIoTe Application Domain and Cloud Domain components, containing all symbIoTe Core Services and symbIoTe-specific extensions of a platform. The two components of the Smart Space domain (the Inkeeper and Local Authentication and Authorization component), which assists in the Smart Space domain management, can be considered as part of the IoT Entity functionality. All those components mainly serve as support functions to provide IoT services (e.g., discovery of appropriate data sources, collaboration with other platforms). The AIOTI Network Layer, which spans through the Smart Space and Smart Device domain, is responsible for device management, ensures the connectivity of smart devices and provides support for device mobility and roaming.

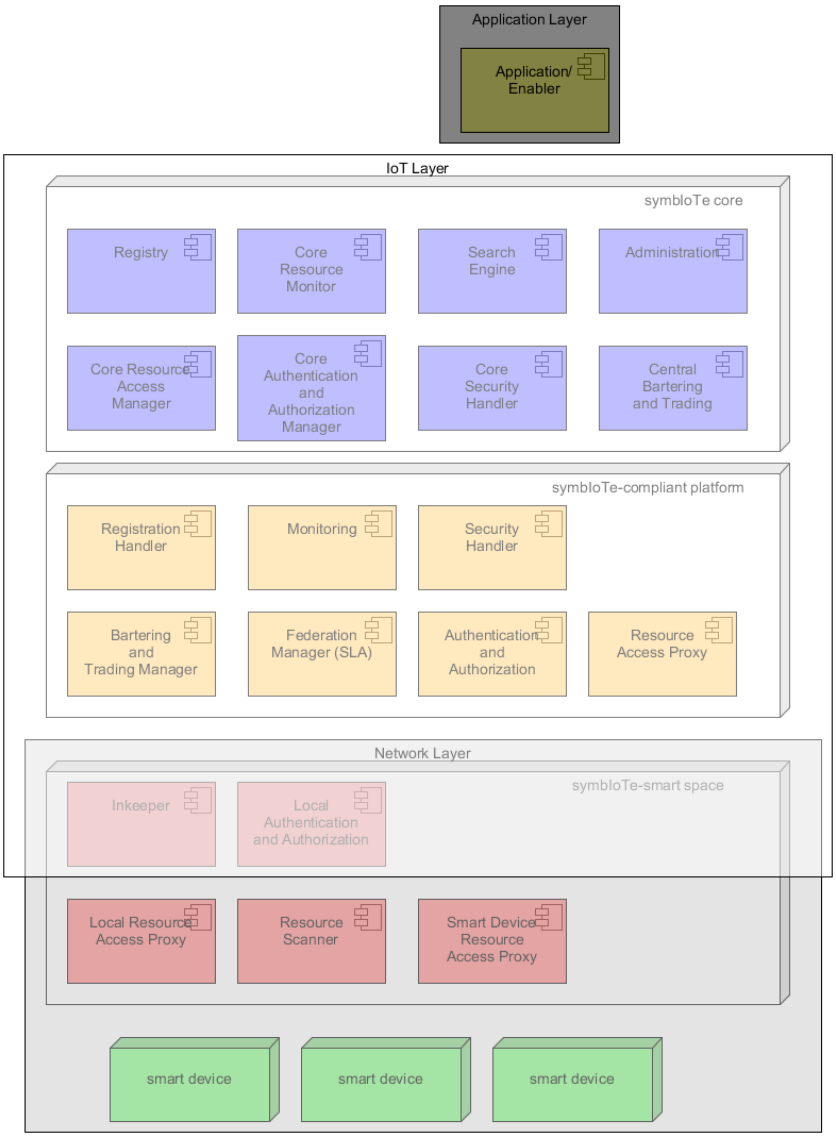

Figure 31 Mapping between the AIOTI HLA and symbIoTe architecture

Figure 32 depicts a mapping between the interfaces defined in AIOTI HLA and symbIoTe functional components. Instead of providing the exact origin and end-point of communication between components, we map only component end-points to AIOTI interface. The number in bracket corresponds to the interface label used in the AIOTI HLA description. An Application/Enabler listens for input from end-users or third parties, which is marked as Commands(1) in Figure 32. Components that support various IoT services (such as resource registration, resource search, resource monitoring, and security component) implement interfaces corresponding to AIOTI IoT interfaces(2). In addition to the IoT interfaces, a very important interface of the symbIoTe architecture is the Horizontal services interface. The interface is used for cross-platform communication, i.e. direct communication between platforms without third-party mediation. The horizontal services interfaces are used in symbIoTe for bartering and trading of resources, or they serve for telemetry communication in a device-roaming scenario. The Central Bartering and Trading component can have also the interface to communicate with other external third-party systems, such as banking in case of charging service. The Network control plain interfaces are mostly present in the Smart Space Domain to take care of device

management and monitoring within local environments. The data plain interfaces are used by an Application/Enabler to access resource-generated data: They can be located at the CLD and/or SSP, depending on the implementation. The Resource Access Proxy except the Data plain interface can have the Big Data interface, which can be used as data provider to a Big Data processing system.

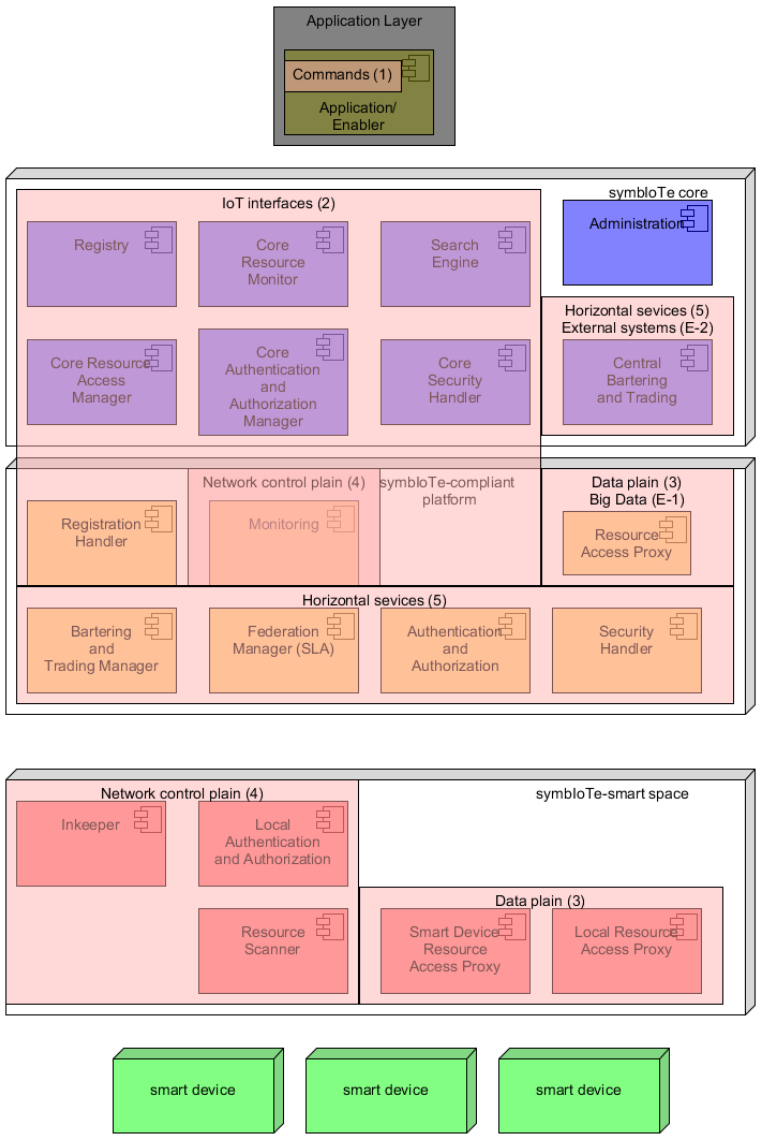

Figure 32 Mapping between the AIOTI HLA interfaces and symbIoTe architecture

6.1.2 oneM2M

oneM2M is a global standardization body for the machine-to-machine (M2M) communications and IoT which has been established in 2012 following an initiative from the European Telecommunications Standards Institute (ETSI). It is formed as an alliance of standardization organizations with 200 member companies from across the world working together “to develop a single horizontal platform for the exchange and sharing of data among IoT devices and applications” [6]. oneM2M focuses on standardization of platform interfaces and aims to provide an interworking framework across different

sectors. The telecommunications industry is clearly leading the oneM2M standardization efforts, but the membership includes a broad range of industries. However, mechanisms for interaction between different platforms are only vaguely addressed.

The first release of oneM2M specifications was published in January 2015, with updated editions in March and August 2016. The released standards cover requirements, architecture, application programming interface (API) specifications, security solutions and mapping to common industry protocols such as CoAP, MQTT and HTTP. One of the major oneM2M contributions in Release 1 is the definition of oneM2M functional architecture which identifies the main components (called nodes in the oneM2M language) within the field domain, which spans over various devices and gateways, and infrastructure domain, which refers to cloud resources. oneM2M pays special attention to the interworking of non-oneM2M devices with one-M2M compliant devices and identifies a special component, an interworking proxy, which is responsible for the full semantic interworking that includes the mappings of data models and protocols. Further details on oneM2M functional architecture are provided in the Appendix in the deliverable D1.2.

The symbIoTe architecture is motivated by the oneM2M functional architecture, and thus it is straightforward to map symbIoTe architectural domains to oneM2M nodes. Figure 33 depicts this mapping: Entities within the Infrastructure Node related to applications (IN-AE) can be mapped to APP, while Common Service Entities (CSE) within the Infrastructure Node map to the CLD. SSP relates to the oneM2M Middle Node, while Application Service/Dedicated node clearly maps to SDEV.

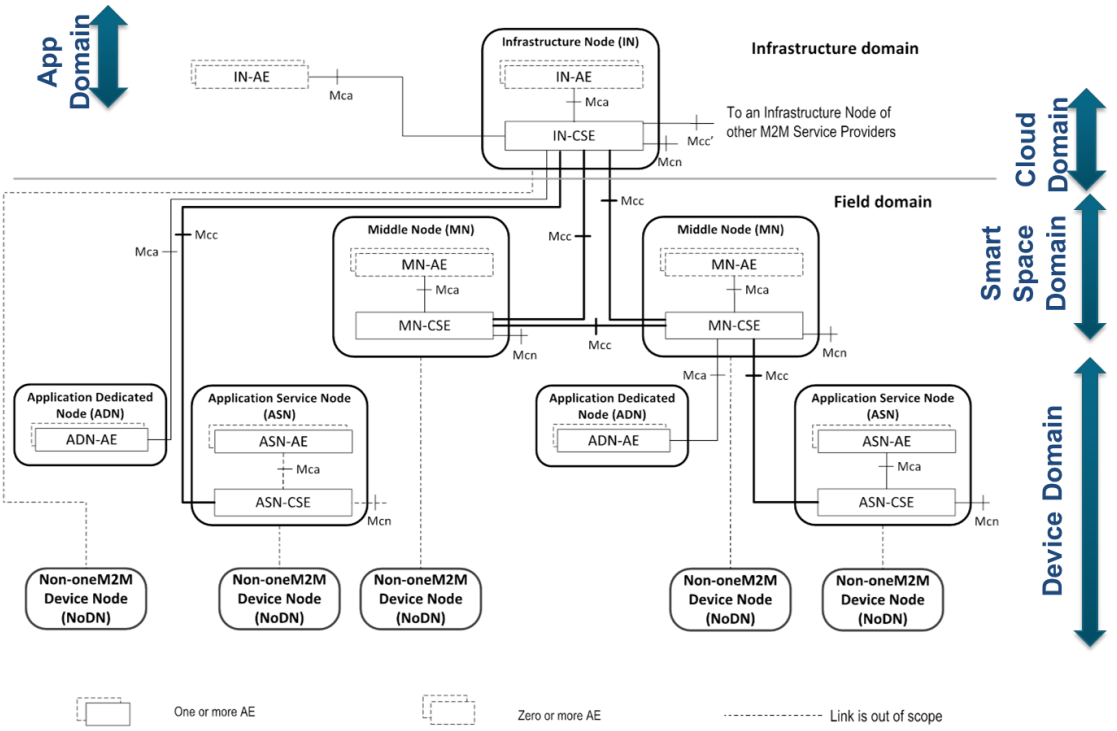

Figure 33 Mapping of symbIoTe domains to oneM2M functional architecture

Hereafter we present the mapping of oneM2M Common Service Entities (CSE), which define the features of oneM2M-compliant platforms, to symbIoTe components and their envisioned functionalities.

6.1.2.1 Compliance Level-1 (CL1)

In the case of CL1, a symbIoTe application is running outside the platform, while using CSEs provided by the platform, as shown in Figure 34. CSEs provided by a platform may include access to sensor data, activation of subscriptions, etc. within the oneM2M Infrastructure Node that maps to symbIoTe CLD. However, to access platform CSEs, external applications first use the symbIoTe Core Services which are not envisioned within the oneM2M infrastructure domain. Those services would thus need to be specified at the level of Infrastructure Node-Application Entity (IN-AE).

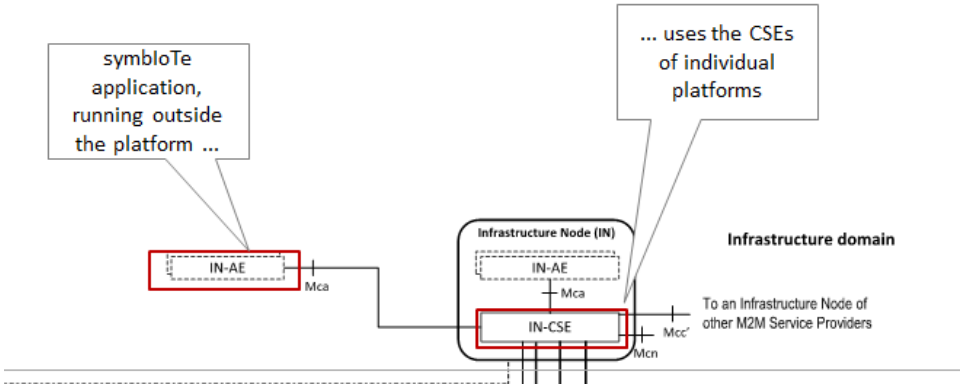

Figure 34 Mapping symbIoTe CL1 to oneM2M

However, oneM2M specifies some of the functionalities within a single platform which are comparable to the functions envisioned within symbIoTe Core Services to work across platforms. The major difference is in the scope, symbIoTe Core Services work across a number of platforms, while oneM2M CSEs are defined for a single platform. Figure 35 shows the symbIoTe Core Services and their relation to oneM2M CSEs (listed in Appendix in the deliverable D1.2). We have identified the following relationship between symbIoTe components and CSEs: Registry – Data Management and Repository; Core Resource Monitor (limited in functionality since it only monitors resources being offered by symbIoTe, but cannot manage the actual IoT devices and associated services) – Device Management; Search Engine – Discovery; Core Resource Access Monitor – Service Charging and Accounting; Core Authentication and Authorization Manager – Security.

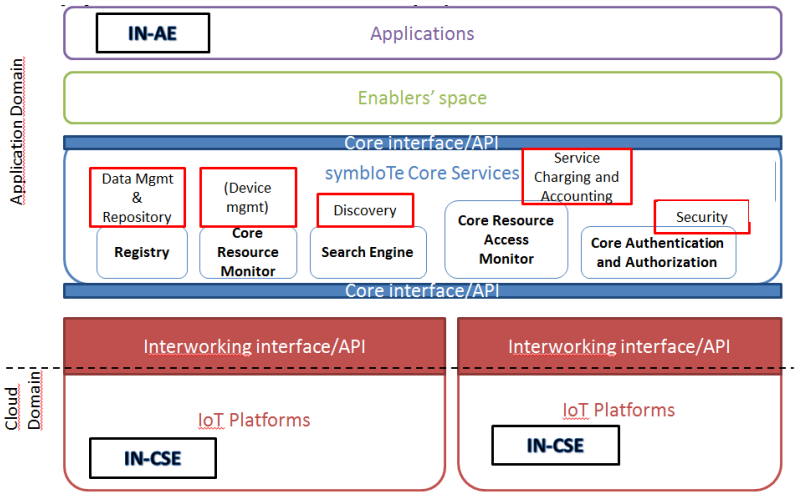

Figure 35 Mapping of symbIoTe Core Services to oneM2M CSEs

6.1.2.2 Compliance Level-2 (CL2)

In the case of CL2, existing (native) applications use CSEs within the platform space and reuse resources of other platforms within a platform federation, as shown in Figure 36. oneM2M already identifies an interaction between two platforms and names this interface Mcc', but no further details are currently provided regarding the functionality enabled though this interface. symbIoTe CLD components which are required for CL2 are not yet envisioned in oneM2M standards, and could thus be used to extend the existing CSEs of oneM2M-compliant platforms, especially the ones related to bartering and trading.

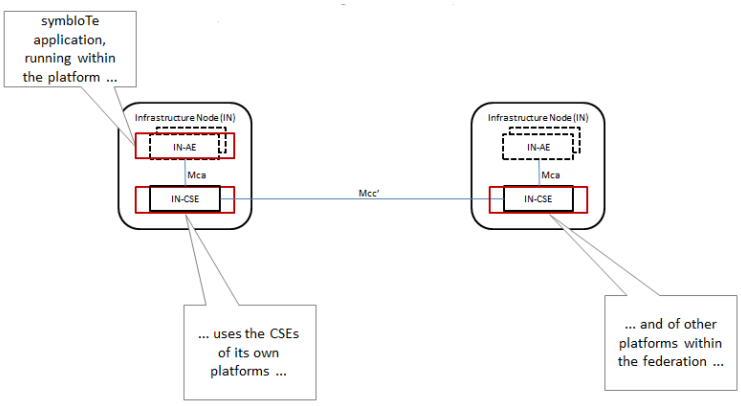

Figure 36 Mapping symbIoTe CL2 to oneM2M

Figure 37 shows the symbIoTe CLD components and their mappings to oneM2M CSEs. Those components would need to be integrated within existing CSEs of an oneM2M-compliant platform and to extend their functionalities. They can be mapped to oneM2M CSEs as shown in Figure 37: Registration Handler – Registration; Resource Access Proxy – Communication management & delivery handling, and Subscription and notification; Authentication and Authorization Manager – Security.

Some of the components in the CLD are also used to achieve symbIoTe CL1: For example, Registration Handler, Resource Access Proxy and Monitoring need to be implemented by L1 Platforms.

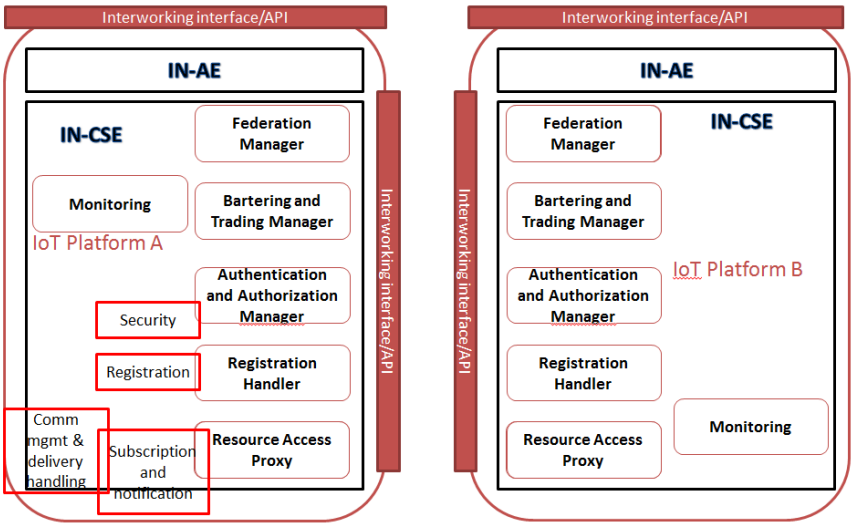

Figure 37 Mapping of symbIoTe CLD components to oneM2M CSEs

As it can be seen in Figure 36, CSEs of different platforms can communicate through the Mcc' reference point. For instance, CSE Registration can communicate with Registration of another platform using this reference point. However, the process of creating an IoT Federation in symbIoTe terms, with reaching Service Level Agreement (SLA) and with bartering and trading mechanism for shared resources, is currently not specified in the oneM2M functional architecture, and thus symbIoTe components for Bartering and Trading or Federation Manager cannot be mapped to oneM2M CSEs. This functionality proposed by symbIoTe has potential to be proposed for standardization within the oneM2M standardization process.

However, oneM2M-defined mechanisms can be used for data exchange between different IoT platforms. Figure 38 shows a mechanism proposed in [28] where an application (IN-AE smart city) from one platform (PF2) wants to receive data from a sensor (ADN-AE sensor) connected to another platform (PF1). In order to do so, procedures of mutual registration, resource announcement and subscription/notification are necessary.

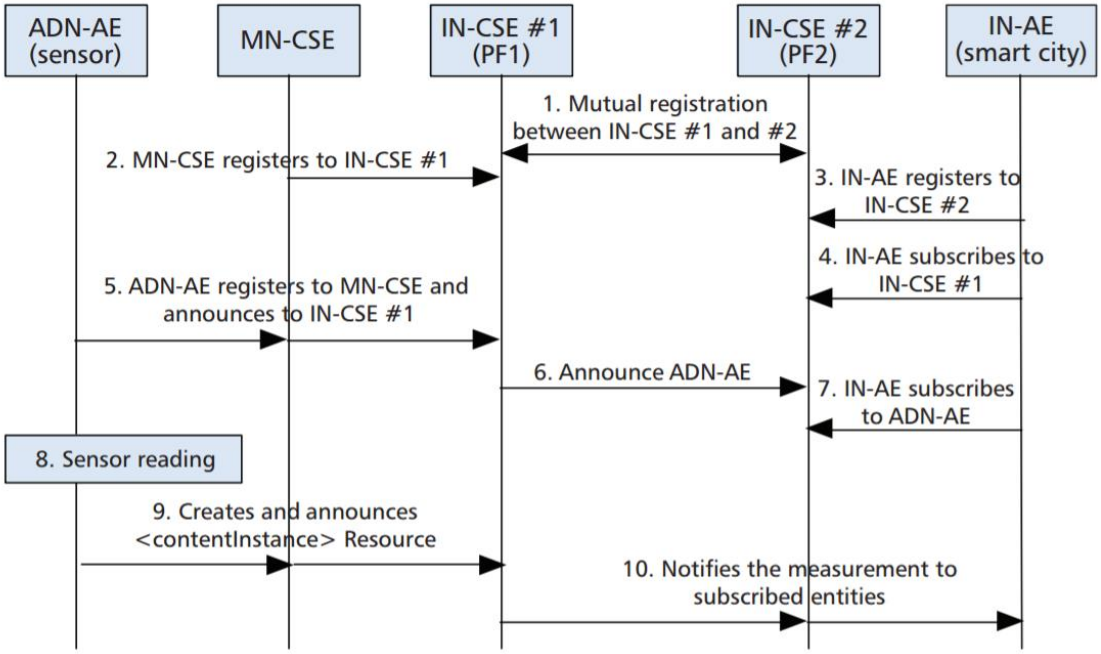

Figure 38 Mutual registration and resource announcements [28]

To enable data exchange between different platforms, their IN-CSEs need to mutually register, as shown in step 1. In this process, those CSEs express their interests in resources from the other CSE. In step 2, MN-CSE registers to IN-CSE #1. By doing this, all information from MN-CSE can be forwarded to applications registered to IN-CSE #1. IN-AE (smart city) registers to IN-CSE #2 in step 3, thus enabling receiving information available through this CSE. It also subscribes to IN-CSE #1 through IN-CSE #2 to receive all the notifications from this node.

When a new sensor (AND-AE) is attached to MN-CSE, it sends a registration message. This information is then announced to the nodes to which MN-CSE is registered, to IN-CSE #1, and then to IN-CSE #2 since those two INs are mutually registered. Announcement process creates a resource at a remote CSE linked to the original resource. Since IN-AE smart city application is interested in sensor readings in IN-CSE #2, it subscribes to AND-AE resource. When AND-AE performs a sensor reading, the

measurement is announced to MN-CSE. Afterwards, notifications are triggered to all subscribed entities (IN-CSE #1, IN-CSE #2, IN-AE smart city).

This mechanism served as the starting point for designing the process of sharing resource metadata between federated IoT platforms for symbIoTe CL2. Only resource metadata is shared through symbIoTe components, while access to resource data is done through other mechanisms.

6.1.2.3 Compliance Level-3 and Level-4 (CL3 and CL4)

In the case of CL3, symbIoTe Smart Spaces are considered, physical environments where one or more IoT platforms provide services. Through Smart Spaces, registered devices, referred to as symbIoTe Smart Devices, can communicate with other devices within the Smart Space, regardless of the local IoT platforms to which those devices belong to. Local IoT platforms can become L3 Compliant by implementing a symbIoTe Agent which connects them to SSP. Each device can become a Smart Device by running symbIoTe application enabling configuration with the Smart Space. L4 Platforms enable one additional functionality, IoT device roaming. In such a case, one Smart Device can change Smart Spaces, while those Smart Spaces are aware that this is one particular device. The components for enabling CL3 and CL4 are the same, the only difference is additional functionality of L4 Platforms. Figure 39 shows the nodes in oneM2M architecture involved when talking about Smart Spaces and CL3.

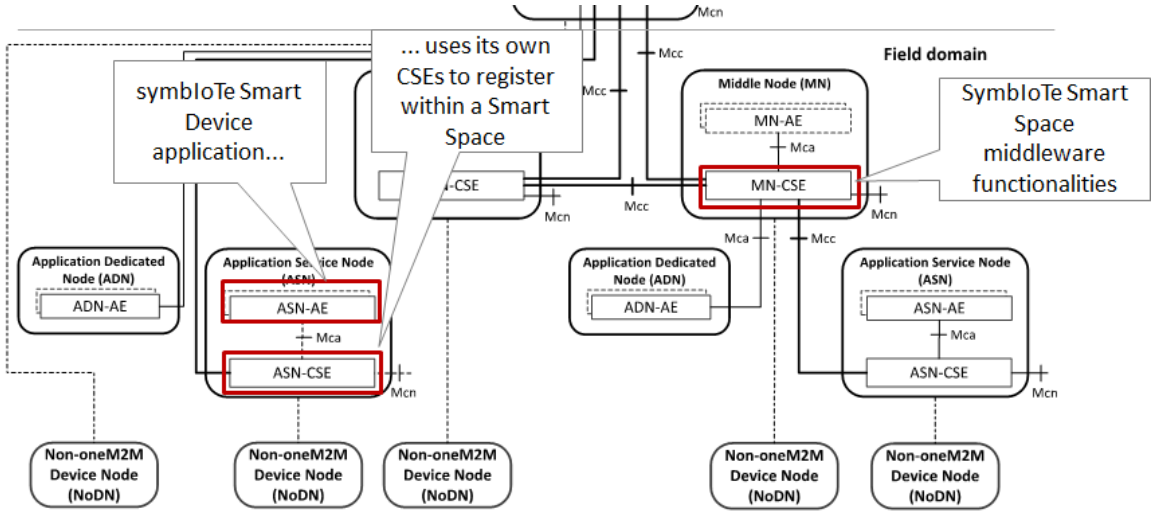

Figure 39 Mapping CL3 and CL4 to oneM2M

Smart Devices can be mapped to Application Service Nodes (ASNs) in the Field Domain. They contain services for registering to Smart Spaces, and functionalities enabling symbIoTe applications to access them, which should in oneM2M be situated in CSEs. Furthermore, ASNs within their Application Entities also need to be able to execute applications, the same way as Smart Devices (e.g. application running on a smart phone or an application performing sensor measurements on a sensor node).

Functionalities of the L3/L4 components are similar to L1 and L2 components, and can be mapped to CSFs (Registry, Resource Access Proxy, ...). However, CL3/CL4 enables interaction between devices on a lower level, when their platforms do not have a cloud. As a result, CL3 actually cannot be linked with oneM2M architecture approach. In oneM2M, connection between two IoT platforms should be through the Infrastructure node in CLD,

while symbIoTe also enables interaction on a lower level. This possibility takes advantage of the fog computing paradigm, which will be presented and compared when mentioning OpenFog Consortium approach in Section 6.1.9. While taking this notion into consideration, functionalities for devices within a physical place, referred to as Smart Space in symbIoTe approach, can be executed at a gateway (fog, or edge) level. Taking the data into the cloud is not necessary for some cases, e.g. for smart home applications where all the data that needs to be processed is within a certain physical place, and can be handled within it.

6.1.3 IoT-A

Authors of IoT-A noticed a trend in the IoT world that steered towards each IoT system manufacturer to produce its own, isolated IoT platform architecture. As a result, numerous IoT systems currently available on market cannot communicate with each other. IoT-A refers to this situation as an Intranet of Things, rather than Internet of Things, and has tried to come up with a solution that would facilitate creating applications which use multiple, heterogeneous platforms. As a result, they have created a set of guidelines, best practices and, most of all, the reference architecture, to help IoT system developers make their platforms interoperable [8]. As symbIoTe's main goal is to transform existing (and future) IoT platforms to become interoperable, the experience gained by the IoT-A team is valuable for the symbIoTe development process.

IoT-A, in its definition, compares the IoT world to a tree. Roots are various hardware devices, providing different functionalities and data using numerous transmission protocols, whereas leaves are concrete application use-cases, e.g. Smart City, Retail, Logistics. IoT-A places itself in a role of a trunk, thus providing architecture needed to connect roots (devices) with leaves (software applications).

Figure 40 The IoT-A tree [8]

To accomplish this task, IoT-A provides a Reference Architecture. As said before, the architecture provides a link between applications and devices using different Functional Components (FC), assembled together to form Functional Groups (FG).

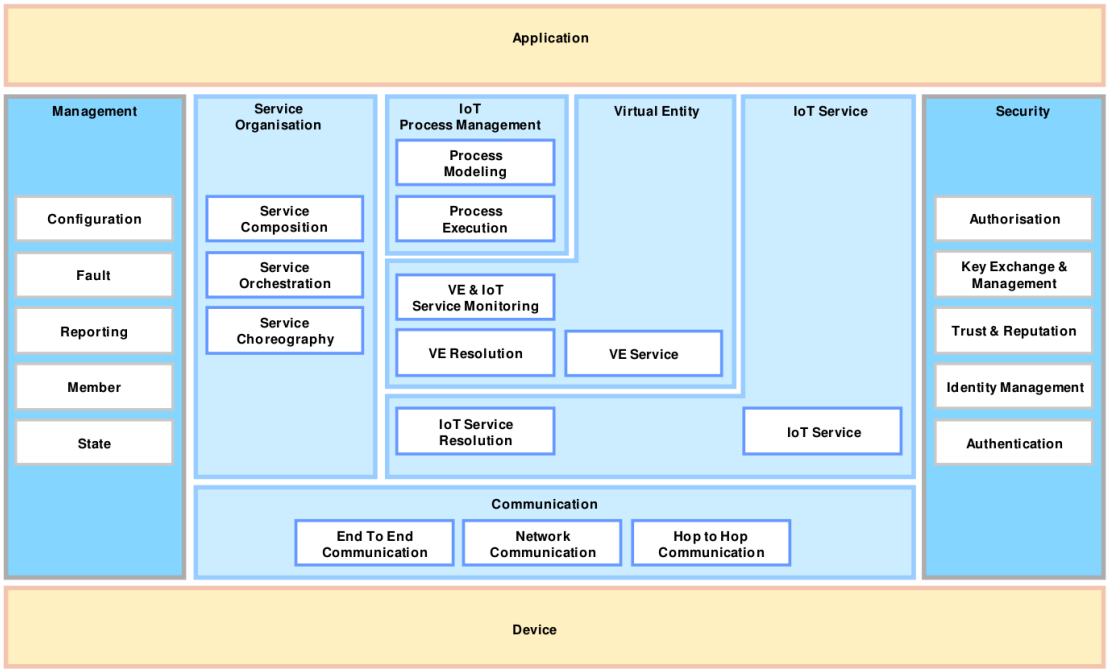

Figure 41 IOT-A reference architecture [8]

The main responsibilities of the Functional Groups, and their symbIoTe counterparts, are as follows [8]:

- The IoT Process Management FG is responsible for transforming traditional business processes into the IoT world. It provides the tools needed to model and execute complex business scenarios. Enablers in symbIoTe architecture provide this type of functionality, gathering data from multiple IoT platforms and offering them to applications.
- The Service Organization FG contains all the necessary functionalities for dealing with numerous services that construct IoT platforms. symbIoTe module with similar functionalities is the Search Engine, which glues together all the data concerning different platforms, their services, metadata etc.
- The Virtual Entity (VE) FG deals with discovering services that allow interacting and provide information about IoT platforms. It is also responsible for finding and managing VE associations. Cloud Domain modules deal with these responsibilities. However, besides working with a single platform, it also allows different platforms to communicate with each other.
- The IoT Service FG is responsible for handling low, device-level IoT services. Responsibilities of this FG are handled by Resource Access Proxy, which provide a uniform abstraction for interacting with different, heterogeneous IoT platforms.
- The Communication FG acts as a backbone, providing communication all the way from user applications, via IoT platform infrastructure, up to low-level devices. It provides uniform method of communication regardless of physical communication type. It is obvious that this FG links the majority of other components, but main modules handling its responsibilities in symbIoTe are Smart Space and Device Domains.

- The Security FG provides mechanisms, that allow to perform secure and trusted interaction between system modules. All these mechanisms are provided by symbIoTe Authentication and Authorisation Manager (AAM) and Security Handler (SH).
- The Management FG takes care of managing the whole system, i.e. configuration, member management and monitoring. Most of these functionalities are handled by symbIoTe Core components: Registry keeps track of all member (users, platforms, resources) information and Resource Monitor checks current system statuses.

Mapping of symbIoTe modules to IoT-A Functional Groups is depicted in Figure 42.

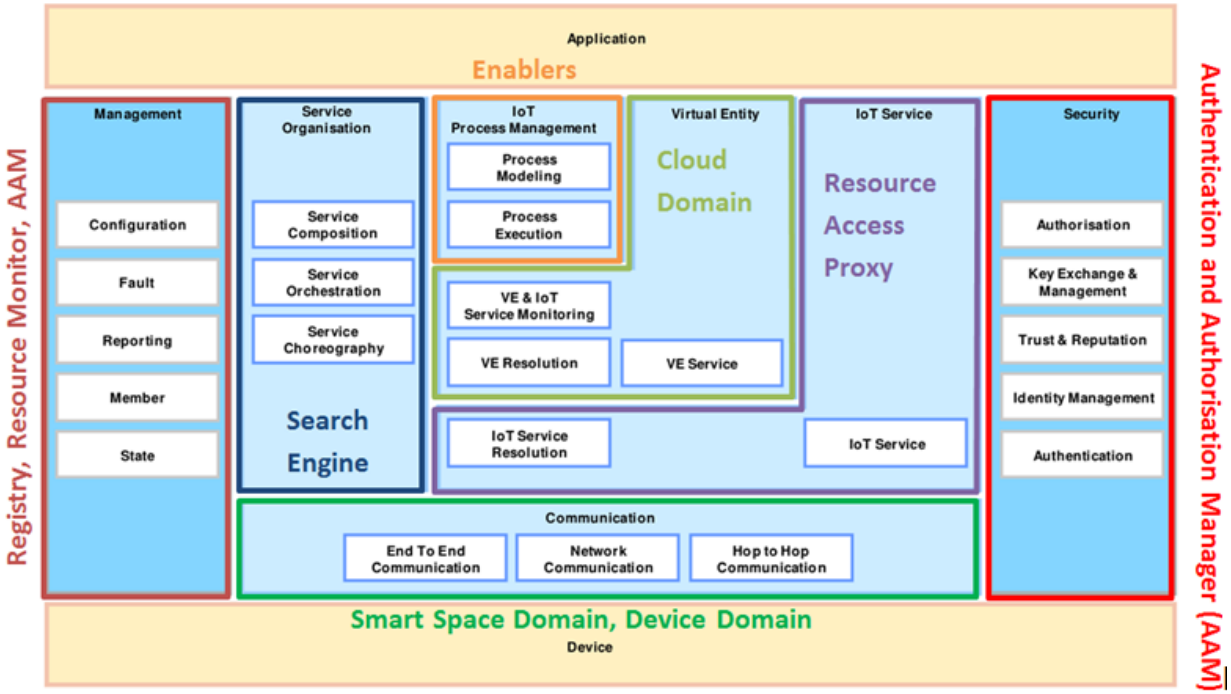

Figure 42 Mapping of symbIoTe to IoT-A reference architecture functional groups

6.1.4 Web of Things

Web of Things (WoT) intends to allow “things” to be part of the Web, enabling them to communicate with each other and become accessible through standard Web technologies. This is accomplished by reusing and extending the software architectures and well-known Web standards, while taking into account thing-to-thing interaction patterns, which are quite different from the ones we have today on the Web [9]. W3C has launched the Web of Things Interest group at the start of 2015 which has focused on gathering information on existing WoT-related solutions, and is now proposing to launch a Web of Things Working Group to standardize those aspects that the Interest Group believes are mature enough to progress to W3C Recommendations [10].

The WoT approach is based upon the fundamentals of Web architecture [10]:

- URIs for identifying Things and their descriptions;
- A variety of protocols for accessing Things, since no one protocol will be appropriate in all contexts;

- Metadata for describing Things as a basis for interoperability and discovery, and playing an analogous role to HTML for Web pages.

The listed principles also apply to the symbIoTe architecture, since the corresponding requirements have been identified in Section 4. However, in contrast to the WoT approach, which extends things so that they become interoperable, where thing-to-thing interaction is vital for dynamic and context-aware environments, symbIoTe assumes that things are managed by and shielded from third party applications and other platforms by platform-specific software components.

The WoT approach focuses on two abstraction layers within the communication stack [10]:

- Application layer: Programs that either implement a thing's behavior, or which interact with a thing, e.g. exposing or utilizing APIs for control of sensors and actuators, and access to associated metadata.
- Thing layer: Software objects that expose the compound state of devices or digital services; data and interaction model, metadata, semantic annotation, Thing Description.

Transfer-specific and networking functions are assumed to be in the focus of platform developers. symbIoTe takes here quite a different position, since our assumption is that platforms also focus on the “Thing layer”, and thus their existing components and APIs cannot be neglected. Thus symbIoTe integrates platforms as a whole, and not things as single entities, into an interoperable IoT ecosystem. Another important distinction is related to the interoperability focus of WoT. WoT clearly deals with syntactic and semantic interoperability, while issues related to organizational interoperability, which is of vital importance to symbIoTe, are currently out of scope.

Note that WoT has so far focused on integrating sensors.

6.1.5 OGC Sensor Web Enablement

The Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) activities focus on enabling WoT functionality by connecting all types of Web/Internet accessible sensors, instruments, and other real world objects. The vision is to foster interoperability within different sensor networks and platforms, as well as to define and approve the standards for plug-and-play web-based networks. The goal can be achieved by SWE’s offer of integration of several different standards. These SWE standards are already integrated and implemented in several projects in the domain of Sensor Web, such as SANY and OSIRIS, and in the global monitoring system (GEOSS), to name a few. These applications have contributed to the improvement of the existing standards’ specifications. For symbIoTe, the following OGC standards[10] are relevant, since they define services, models and interfaces that are also in focus of symbIoTe:

- **SWE Common Data Model Encoding** – Specifies a low-level data model and encoding in order to define and package sensor related data in a self-describing and semantically enabled way.

[10] The full list can be found on http://www.opengeospatial.org/standards/

- **Sensor Model Language (SensorML)** – Defines a data model and encoding to describe processes and processing components associated with the measurement and post-measurement transformation of sensor observations.
- **Observations and Measurements (O&M)** – Definition of a data model and schema for encoding measurements and observations.
- **Sensor Observation Service (SOS)** –Service model and interface encoding to provision sensor measurements and observations, from simple sensors to complex sensor systems, both physical and virtual.
- **Sensor Planning Service (SPS)** – Defines a service model and interface encoding for the execution of sensor tasking and parameterization requests. It is used to manage sensors and sensor networks and to influence the measurement process according to specific needs and requirements.
- **Sensor Alert Service (SAS)** – Defines an interface to connect to a sensor with a publish/subscribe model to be notified about alerts from the sensor.
- **Sensor Event Interface (SES)** – Defines an interface to be informed about sensor events (like the availability of new observations) in a publish/subscribe model.
- **Web Notification Service (WNS)** – This service offers to inform clients about notifications (alerts or events) from a sensor or from other elements within a processing chain.

These OGC SWE standards can be used to remove the entry barrier for anyone who wants to develop a WoT system, i.e., connect different devices and real world objects for interoperability and accessibility purposes. On the other hand, the OGC SWE services rely on fairly complex and “heavy” protocols. This implies an enormous challenge to implement these protocols on current IoT hardware with existing energy and memory constraints. Therefore, the OGC SWE standards may be more relevant in the context of stationary sensors or gateway servers which provide a link to underlying sensor networks, while symbIoTe requirements specify support for mobile IoT devices and actuators which are not in the focus of OGC SWE standards.

6.1.6 Industrial Internet Reference Architecture

The objective of the Industrial Internet Reference Architecture (IIRA) is to “create broad industry consensus to drive product interoperability and simplify development of Industrial Internet systems that are better built and integrated with shorter time to market, and at the end better fulfil their intended uses” [11]. The driving force behind IIRA is the Industrial Internet Consortium (IIC), which is an open membership organization and was founded as a program by the Object Management Group®. It is an international organization with members around the world including many major players in the industrial automation domain.

The IIRA is based on the ISO/IEC/IEEE 42010:2011 standard for systems and software engineering architecture description. Following the conventions of this standard, the IIRA is using viewpoints to model the involved stakeholders and their concerns. When comparing the symbIoTe approach to IIRA, the most relevant viewpoint is the functional viewpoint. A driving idea behind the functional viewpoint is the observation of the unification of two different domains, the Information Technology (IT) and the Operations Technology (OT).

In general, Operations Technology (OT) has traditionally been controlled by closed systems, based on SCADA (Supervisory Control And Data Acquisition) systems and is

now experiencing a transformation towards systems based on the Internet Protocol (IP), with important changes happening in the visibility of such systems, their intelligence and interoperability. Thus, IIRA aims to define all viewpoints and considerations that have to be taken into account in this transformation of industrial systems towards IoT.

A mapping between the symbIoTe architecture and three-tier IIRA Implementation viewpoint general architecture and definition of functional domains is rather straightforward:

- symbIoTe Application Domain corresponds to IIRA Enterprise Tier (in both Business and Application Domains).
- symbIoTe Cloud Domain corresponds to IIRA Platform Tier (where platforms include Information Domain and Operations Domain).
- symbIoTe Smart Space and symbIoTe Smart Device Domain both correspond to IIRA Edge Tier as both gateways and endpoints belong to IIRA Edge Tier, with distinction that device-to-gateway communication belongs to Proximity Network while gateway-to-platform communication is function of Access Network. In IIRA the Edge Tier has a single functional domain – Control Domain.

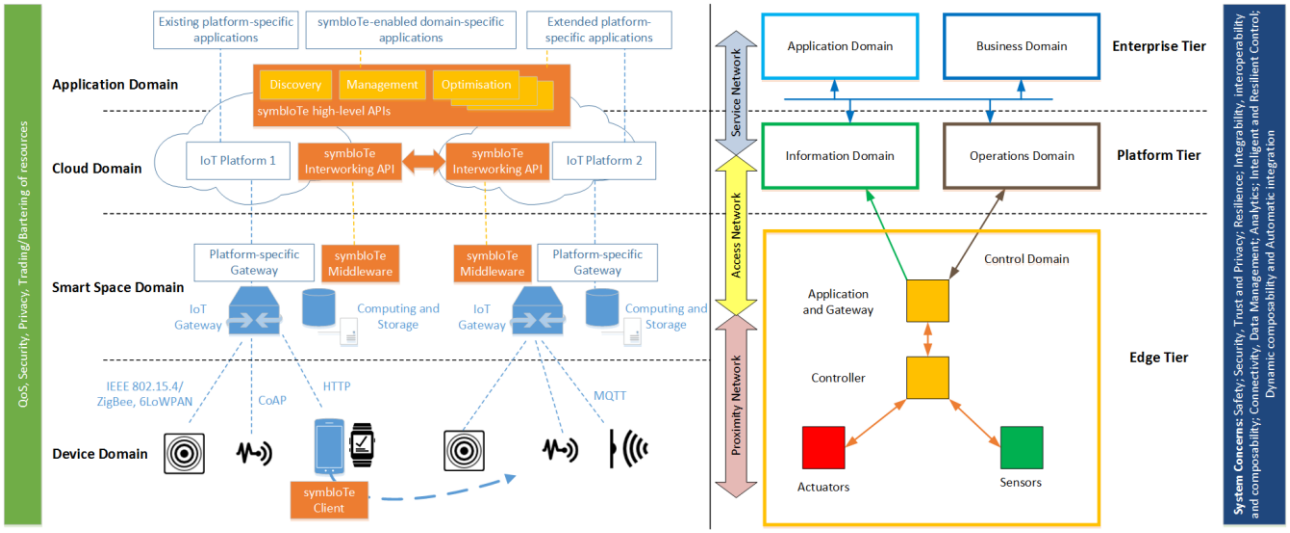

Figure 43 Comparison between symbIoTe and IIRA achitecture

Further details on IIRA are given in the Appendix of the deliverable D1.2.

6.1.7 Reference Architecture Model Industrie 4.0

The Reference Architecture Model Industrie 4.0 (RAMI4.0) [13] is an outcome of the cooperation between the VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA) and the Zentralverband Elektrotechnik- und Elektronikindustrie e.V (ZVEI). These two organizations are representing members from the German automation industry and the information and communication technology domain. Joint working groups from these organizations are developing the RAMI4.0.

According to the authors of RAMI4.0, the global IoT trend in Germany is driven by the so-called Industrie 4.0. The driving vision behind Industrie 4.0 is that real-time availability of all relevant information within the complete value chain, throughout all business layers and on all aggregation levels, will trigger a new industrial revolution. RAMI4.0 is supposed to

enable this vision by providing a unified approach to describe the complete automation landscape and to select appropriate standards or, when necessary, to provide requirements for additional standards. It is planned to be a German Norm, DIN SPEC 91345, which is currently being developed.

Like IIRA, RAMI4.0 also introduces a layered architecture concept to separate interoperability aspects. Starting from the lowest *Asset Level* and *Integration Level* which are dedicated to the integrability of system components, followed by the *Communication* and *Information Level* dedicated to the semantic interoperability, finally the *Functional* and *Business Level* is defined to support the composability of application units (see Figure 1 in [13]).

One important feature of RAMI4.0 is the integration of the office floor and the shop floor. In the past, these two layers have been mostly separated by different communication infrastructures as well as different types of information models. However, the interactions between both areas are becoming more and more important, and require more general information concepts. Because of that development, RAMI4.0 introduces a so-called *Industrie 4.0 Component* (I4.0) to encapsulate the individual building blocks of an automation application. These I4.0 components are based on a common semantic model between shop floor and office floor, and can be considered as a key feature of the RAMI4.0 approach. This common information model plays a primary role for the reference model. This concept is even introduced as a reference architecture model for semantic technologies.

Just as in the Internet of Things domain everything is considered to be a *Thing*, RAMI4.0 considers every automation component to be a *Thing*. In order to create a common information model, every automation component (Thing) becomes an I4.0 component by being surrounded with an *Administration Shell* (see Figure 8 in [13]). This shell contains the virtual representation and the technical functionality of the component. By wrapping all components into an Administration Shell, a common information and communication model is established, which is the backbone of the RAMI4.0 semantic interoperability concept.

When comparing RAMI4.0 to other reference and functional architectures, a Thing surrounded with an Administration Shell has conceptual similarities to a WoT thing, or oneM2M Application Node. In contrast to RAMI4.0, symbIoTe does not focus on the automation industry, nor it considers individual interoperable things as building blocks of the future interoperable IoT ecosystem, but rather integrates the services offered by IoT platforms and their device management tools through open APIs providing authenticated and authorized access to those IoT-based services.

6.1.8 ISO/IEC Internet of Things Reference Architecture

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) are worldwide standardization organizations responsible for many international standards used in a broad field of application areas. The concept of the Internet of Things has been studied for many years and has constantly gained momentum. In the past these developments have been done without any dedicated IoT standards. Now the ISO/IEC has decided that it is time for a guiding standard in order to achieve interoperability between different types of IoT systems. In 2014, the Working Group Sensor Networks (WG7) and Working Group Internet of Things (WG10) from ISO/IEC Joint Technical Committee 1 published a Working Draft on the topic

of IoT standardization called Internet of Things Reference Architecture (IoT RA) [14] which is currently still under development. The document is distributed only for review and comments and may not be referred to as an international standard. The IoT RA shall serve the following goals:

- describe the characteristics and aspect of IoT systems;
- define the IoT domains;
- describe the reference model of IoT systems; and
- describe interoperability of IoT entities.

In this early state, only a few design concepts are defined. One is the introduction of a common vocabulary service to be used by all layers. Context-awareness, discoverability and plug-and-play capability are considered major characteristics of an IoT system. This requires a common model for the IoT entities to provide a shared conceptualization for the architecture elements.

As the document is in an early stage, the focus is on the definition of terms and general concept, while interoperability and technical details are not yet well covered. Thus at this point is it still not possible to relate the symbIoTe to the emerging ISO/IEC RA.

6.1.9 OpenFog

OpenFog[11] is a consortium with the aim of standardizing and promote fog computing. The consortium was founded by Cisco Systems, Intel, Microsoft, Princeton University, Dell and ARM Holdings in 2015. Fog computing is an architecture that pushes the intelligence of a system (computing, storage, control and networking functions) down to the local area network, processing data in a fog node or IoT gateway.

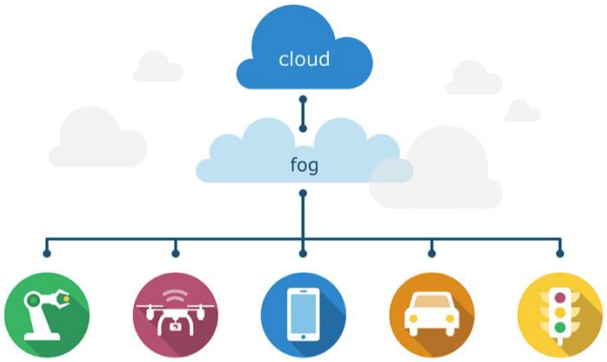

Figure 44 Fog Computing

In simple terms, it is an extension of the cloud to the edge, simplifying IoT applications by removing the need to be consistently connected to the cloud and allows for the deliverance of low latency computations. Using self-driving cars as an example, in a traditional cloud computing model, all readings would be sent to the cloud. There, they would be analysed by algorithms to detect any problem that would need some form of actuation (e.g. detecting something on the road and making the car go around it) and it would send the corresponding actions to be taken back to the car.

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

But one has to question the need to send everything to be processed to the cloud. The bandwidth needed to push sensor readings at scale can be too large, as can be the time to send and receive messages. This is even more important in the context of critical systems that need to take immediate actions upon detecting some problem.

Fog computing does not intend to replace the cloud, but rather to mutually benefit each other. Decisions and tasks such as data analysis can be taken to the fog nodes while other tasks such predictive analytics on historical data can be done at the cloud.

The OpenFog consortium released the OpenFog Reference Architecture[12], a universal technical framework designed to enable data-intensive requirements of the IoT, 5G and artificial intelligence applications. It is not a standards documents, with Must, Should and MAY requirements, but rather a high level guide for the industry to use architecturally. Basically, it is a series of recommendations for a successful fog computing implementation.

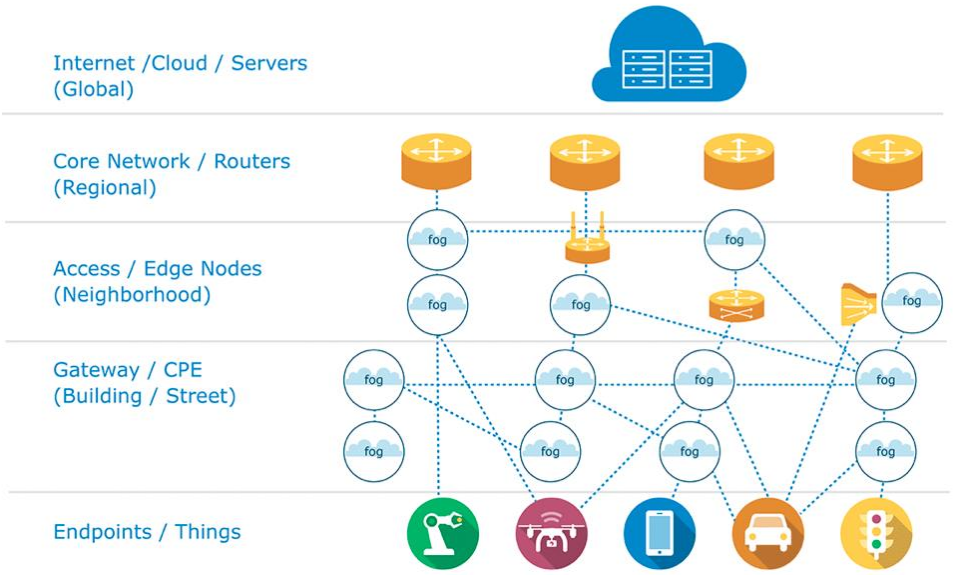

Figure 45 Fog Computing overview

Similar to symbIoTe, they have opted for an open approach, with the intention of avoiding vendor lock-in, lower system costs and accelerate market adoption.

The reference architecture is based on eight pillars defined by the consortium:

- **Security**: trust, attestation, privacy;
- **Scalability**: localized command, control and processing, orchestration and analytics, avoidance of network taxes;
- **Openness**: resource visibility & control, white box decision making, interoperability and data normalization;
- **Autonomy**: flexibility, cognition and agility, value of data;
- **Reliability, availability and serviceability**;

[12] https://www.openfogconsortium.org/ra/

- **Agility**: tactical and strategic decision making, data to wisdom;
- **Hierarchy**: fully cloud enabled computational and system autonomy at all levels; and
- **Programmability**: Programmable software/hardware, virtualization and multi-tenant, application fluidity.

While symbIoTe's APP and CLD architecture is cloud-centric, the concept of Smart Spaces is closer to what fog computing and OpenFog envision for the IoT world. Smart Spaces provide local functionality for devices and platforms such as dynamic reconfiguration of devices and support for roaming devices, as well as allowing platforms and devices to function without cloud connectivity.

6.2 Related projects and platforms

This section presents a short overview on platforms and projects which are relevant to symbIoTe goals and architecture. Following projects have been analysed: Fiware, Compose, Crystal, iCore and IoT-EPI projects.

6.2.1 FIWARE

FIWARE is a set of open-source components directed at Future Internet that can be used in any system that you might want to build. It can be applied in a variety of areas, such as smart cities or environment sustainability. Due to its modularity, it is very simple to use, only needing to make use of the components that you are interested in. The components that FIWARE provides can be accessed via REST APIs, facilitating the process of developing IoT applications. It provides replicability capabilities, allowing, for example, for a solution developed for one city to be deployed in another city easily.

Hereafter we list a set of FIWARE components that are the most relevant to the symbIoTe architecture:

- **Orion Context Broker**[13] provides NGSI9[14] and NGSI10[15] interfaces that allow:
  - The registration of resources (e.g. temperature sensor);
  - For updates regarding the resources to be sent (e.g. temperature changes);
  - For notifications to be sent with a given frequency or when a change happens (e.g. temperature changes);
  - Querying the context broker to obtain up-to-date information provided by the sensors.

  Orion can be relevant to symbIoTe by enabling the registration of resources to symbIoTe and for symbIoTe to provide a way for application developers to obtain data from sensors.

[13] http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[14] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification
[15] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification

- **KeyRock**[16] provides identity management functionality. It provides common functionalities needed to handle users' access to networks, services and applications such as secure and private authentication, authorization and trust managemet, user profile management and Single Sign-On. It is relevant for symbIoTe as a tool used to handle users' life-cycle functions. In symbIoTe, this functionality is to be implemented in Authentication & Authorization Manager (AAM) and Security Handler (SH).
- **IDAS**[17] is a backend device management. It translates protocols specific to the IoT into NGSI context information protocol, ready to be consumed by Orion. This allows devices to be represented in a FIWARE platform. The component can be relevant to symbIoTe as it has functionalities similar to Registration Handler (RH), and Resource Access Proxy (RAP). Additionally, since it enables registration of devices, it could also be relevant in Smart Space and Smart Device Domain (SSP, SSDEV).

Several of these enablers can be deployed using already built images. FIWARE also provides an enhanced OpenStack-based cloud environment. The usage of these popular projects can be relevant to the deployment and maintenance of the symbIoTe ecosystem.

6.2.2 COMPOSE

COMPOSE (Collaborative Open Market to Place Objects at your Service) is an open-source ecosystem aiming at transforming the Internet of Things into an Internet of Services [20]. The main vision of the project is to integrate the IoT with the IoS (Internet of Services) through an open marketplace, in which data from Internet-connected objects can be easily published, shared, and integrated into services and applications. The marketplace provides all the necessary technological enablers, organized into a coherent and robust framework covering both delivery and management aspects of objects, services, and their integration. The platform offers connectivity to IoT devices accompanied by advanced data management capabilities, including real-time data processing capabilities. The project develops novel approaches for virtualizing smart objects into services and for managing their interactions. This includes solutions for managing knowledge derivation, secure and privacy preserving data aggregation and distribution, dynamic service composition, advertising, discovering, provisioning, and monitoring. To validate different aspects of the platform COMPOSE addresses the following application areas: smart shopping spaces, smart city and smart territory.

[16] http://catalogue.fiware.org/enablers/identity-management-keyrock
[17] http://catalogue.fiware.org/enablers/backend-device-management-idas

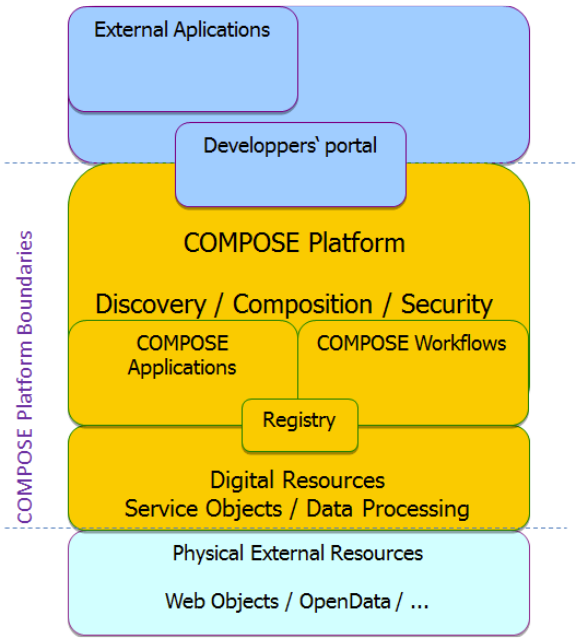

Figure 46 COMPOSE high level architecture [15]

The architecture of COMPOSE system is shown in Figure 46. When mapping it to symbIoTe architecture, it is obvious that COMPOSE Platform is actually similarly designed as symbIoTe Core Services. The upper level comprises external applications, while the layer below comprises things/devices available from different sources. Web objects, physical external resources, open data from Figure 46 can be mapped to IoT services offered by L1 IoT platforms.

Service objects are an internal representation of physical external resources or web objects. Registry component has the similar functionality as the component with the same name from symbIoTe – it holds semantic metadata for service objects hosted by the platform, and enables discovery of those objects by external applications. Apart from that, COMPOSE hosts the composition services engine to help external developers combine the base service objects and applications into workflows and external applications.

6.2.3 CRYSTAL

CRYSTAL (CRitical sYSTem engineering AcceLeration) aims at establishing and pushing forward an Interoperability Specification (IOS) and a Reference Technology Platform (RTP) as a European standard for safety-critical systems [21]. The goal of the project is to reduce complexity of the integration process, i.e. to enable interlinking and sharing data between different systems based on standardized and open Web technologies. Such solution should enable interoperability among various life cycle domains.

The main idea of the so-called Interoperability Specifications (IOS) is to rely on common interoperability services, providing a common ground for integrating lifecycle and engineering tools across different engineering disciplines and from multiple stakeholders involved in the development of large scale safety-critical systems (i.e. the projects focuses on four domains: the automotive, aerospace, rail and health sector). The common denominator of the IOS is based on a lightweight and domain-agnostic approach, providing basic capabilities for handling the whole lifecycle of engineering artefacts manipulated throughout the development of safety-critical embedded systems.

Even though Crystal is not focusing on IoT platforms but rather on engineering processes and allows data sharing between them, it is related to symbIoTe goals as it ensures a data repository that allows different system to access the data, providing also the necessary semantics for each system to use the data. The results from this project are related to symbIoTe component Registry which stores data from different sources and enables a unified access to this stored data from external applications. However, symbIoTe covers an enlarged scope in terms of interoperability aspects than Crystal.

6.2.4 iCore

The iCore initiative addresses two key issues in the context of the Internet of Things (IoT), namely how to abstract the technological heterogeneity that derives from the vast amounts of heterogeneous objects, while enhancing reliability and how to consider the views of different users/stakeholders (owners of objects and communication means) for ensuring proper application provision, business integrity and, therefore, maximize exploitation opportunities [22]. To validate the proposed solutions, iCore addresses the following use cases: ambient assisted living, smart office, smart transportation, and supply chain management.

The iCore architecture comprises three levels of functionality: virtual objects (VOs), composite virtual objects (CVOs) and functional blocks for representing the user/stakeholder perspectives, as shown in Figure 47. VOs are representations of real world objects that can be aggregated and merged in order to create new Virtual Composite Objects (VCOs) that extend and generalize real world objects' functionalities and features. They are semantically described by using RDF triplets. On top of VOs and VCOs, service enabling functions offer services to applications via API such as data analysis ets.

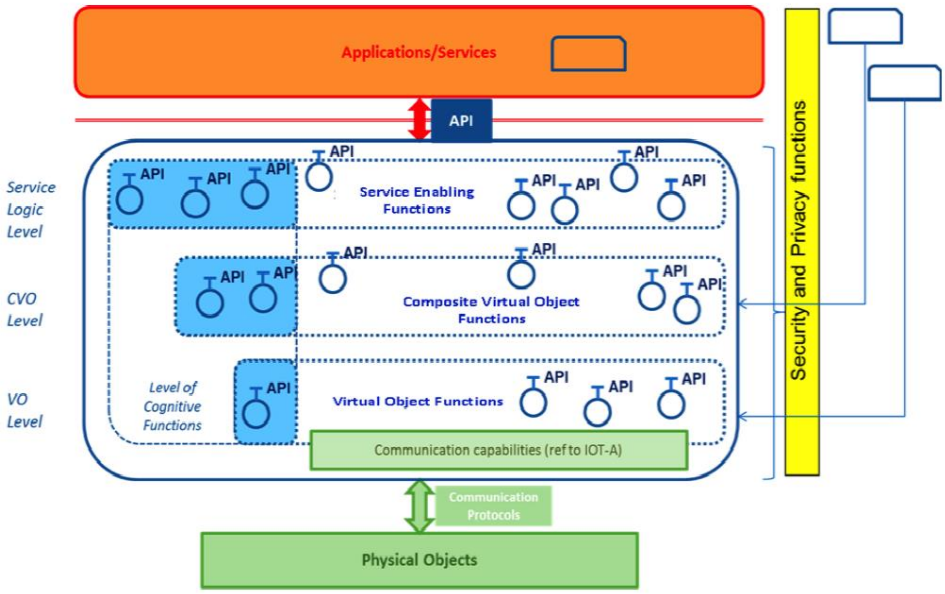

Figure 47 iCore architecture [16]

Architecture of iCore is similar to symbIoTe core architecture, except that symbIoTe core does not store the data provided by real-world objects, it is only a mediator between applications and data provided by physical objects that is stored on IoT platforms.

6.2.5 Positioning of symbIoTe with regard to other IoT-EPI Projects

SymbIoTe project is a part of the IoT-European Platforms Initiative (IoT-EPI) formed to build a vibrant and sustainable IoT-ecosystem in Europe and aiming to maximize opportunities for platform development, interoperability and information sharing. Other projects which are a part of IoT-EPI are AGILE, BIG-IoT, bIoTope, VICINITY, Inter-IoT and TagItSmart[18].

In terms of the holistic and cross-domain approach which covers both devices, gateways, cloud services and applications, symbIoTe has most similarities with Inter-IoT which also considers interoperability at various levels of the IoT stack. A major difference between symbIoTe and Inter-IoT is in the approach towards the information model. While Inter-IoT enables platform interoperability by mapping different platform information models, symbIoTe approach enables each platform to register extensions of the Core Information Model so that platforms can register the aspects of their resources they regard important. In such a way, platforms that do not use a standardized information models, that can be mapped to other well-known models, can join the symbIoTe interoperability framework with less effort, without the need to define mappings. We see this as a more flexible approach that facilitates different SMEs, who are symbIoTe's primary target group, to interoperate with other IoT platforms. Additionaly, since symbIoTe offers flexible interoperability mechanisms enabled by an incremental deployment of symbIoTe functionality, IoT platforms can choose the layer on which they want to enable interoperability. For instance, if platforms do not have a Cloud layer, they can integrate symbIoTe SSP components without the need to use the Cloud if this is not in their interest.

When looking at the symbIoTe Core Services developed for the Application Domain, they are relevant and comparable to the BIG-IoT architecture which develops an IoT Marketplace with a significantly broader scope than the symbIoTe Core Services. The envisioned symbIoTe Smart Space middleware can be put into relation to the AGILE gateway which supports various devices and communication protocols. An analysis of potential IoT-EPI project synergies shows that potential points for collaboration and common agreement are open APIs being defined at the platform level (symbIoTe Interworking API) and gateway level (symbIoTe Smart Space API).

6.3 IoT Platforms contributed by symbIoTe partners

Several platforms belonging to symbIoTe partners will be included in the symbIoTe ecosystem. In this section, their most important features are mentioned, as well as plans for their integration.

6.3.1 OpenIoT

The OpenIoT platform is an open source IoT platform enabling the semantic interoperability of IoT services in the cloud. At the heart of OpenIoT lies the W3C Semantic Sensor Networks (SSN) ontology, which provides a common, standards–based model for representing physical and virtual sensors. OpenIoT also includes sensor middleware that eases the collection of data from virtually any sensor, while at the same time ensuring their proper semantic annotation. It offers visual tools that enable the development and deployment of IoT applications with almost zero programming. Another

[18] For short project descriptions visit http://iot-epi.eu/.

key feature of OpenIoT is its ability to handle mobile sensors, thereby enabling the emerging wave of mobile crowd sensing applications. The platform is currently available as an open source project (https://github.com/OpenIotOrg/openiot/) and supported by an active community of IoT researchers, while being extensively used for the development of IoT applications in areas where semantic interoperability is a major concern. As of June 2014, it consists of nearly 400.000 lines of code. In February 2015 there we 25 active registered contributors to the OpenIoT source code and 66 users registered in developers mailing list. OpenIoT received an award from Black Duck[19], as being one of the top ten open source project that emerged in 2013.

Within symbIoTe project, specific wrappers for the OpenIoT platform will be implemented to enable symbIoTe CL1. Additionally, OpenIoT will be integrated within the symbIoTe IoT federation, enabling CL2.

6.3.2 Symphony

Symphony is the Nextworks platform for the integration of home/building control functionalities, devices and heterogeneous subsystems. Symphony can monitor, supervise and control many different building systems, devices, controllers and networks available from third-party suppliers. By intelligently correlating cross-system information, a flexible and highly efficient platform is delivered to the stakeholders. The system is a service-oriented middleware integrating several functional subsystems into a unified IP-based platform. As hardware/software compound, Symphony encompasses media archival and distribution, voice/video communications, home/building automation and management, and energy management.

6.3.2.1 Architecture

The concept schematic of the Symphony suite is depicted in the following two pictures. Being a commercial product whose IPR belongs to Nextworks, internal details of the platform cannot be disclosed.

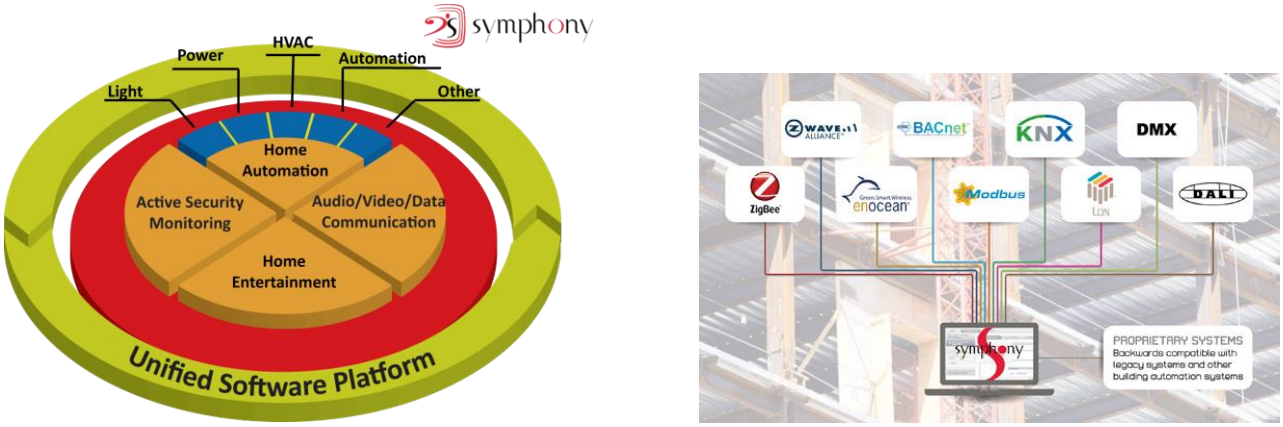

Figure 48 Symphony platform concept

[19] https://www.blackducksoftware.com/about/news-events/releases/black-duck-announces-open-source-rookies-of-year-winners

Nextworks is currently evolving the Symphony platform to further pursue functional decomposition and allow service modules to be distributed out of the stand-alone system over a wide area (i.e. local cloud, public cloud), in a truly flexible IoT paradigm. The next planned step for Symphony is to include energy management options and become highly distributed on a variety of hosting systems (e.g. domestic NAS or micro servers hosted at home or at providers' curbs or in the cloud) and highly flexible to incorporate more and more technology drivers for sensor/actuators.

Within symbIoTe, Symphony will aim to achieve CL3, thus enabling creation of dynamic smart spaces. The symbIoTe architecture and its smart residential collaboration application scenario are key enablers of this strategy, allowing to open the system to the interoperation with other IoT systems and inspiring the development of key features like:

- Generalized abstract model for all the ICOs (smartphones, printers, sensors, actuators, etc.) with APIs to implement context-driven decisions/actions
- Automatic resource discovery and dynamic configuration of services
- Seamless multi-protocol adaptation, control of various systems, publishing of large amounts of unstructured data (BigData) across the various and distributed decision points
- Seamless use and integration with diverse local area / personal area connectivity media like Zigbee, Z-Wave, Bluetooth LE
- Distributed execution of the platform middleware across any locally available devices (e.g. mobile devices, residential devices, local routers)

6.3.3 Mobility BaaS

Nowadays cities are looking to implement systems that will allow them to actively get a feel of their surroundings so that they can act in real time. The problem is that most of the time, this type of systems do not integrate very well with each other, either because they are from different vendors or because they are open-source projects built by independent developers which need some work to integrate with the rest of the backend. With this in mind, the Mobility Backend as a Service (MoBaaS) offers a set of services, in the form of APIs, which intend to eliminate the friction created by having services from multiple vendors.

MoBaaS will be a symbIoTe-enabled platform aiming to achieve CL1, used in the Smart Mobility and Ecological Routing Use Case. It enables integration of data from many sources, focusing on the mobility aspect of the city. Figure 49 shows which types of services the MoBaaS offers and which kind of devices can be connected to it.

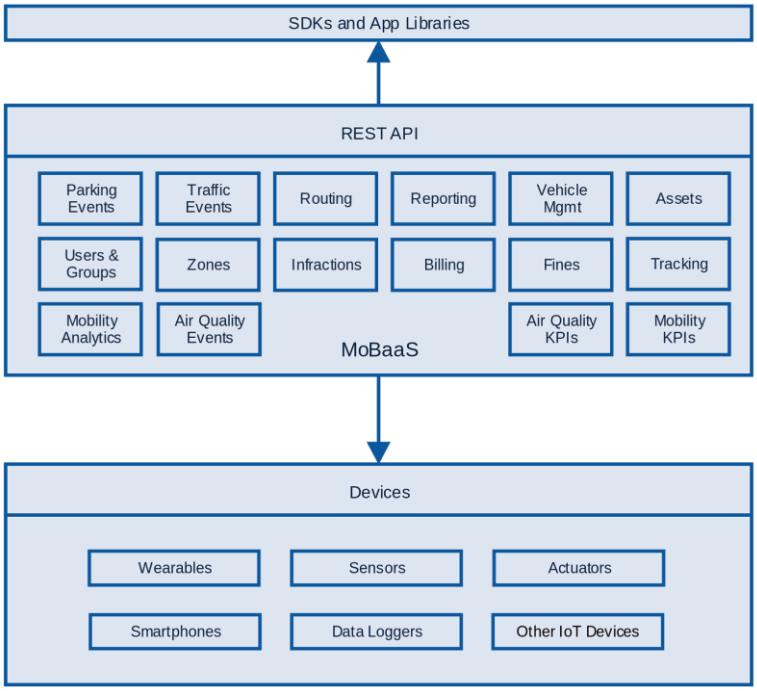

Figure 49 MoBaaS overall architecture

Figure 49 represents the overall architecture of the MoBaaS and is divided into three layers. The first layer, on the bottom, represents the devices that can be connected to the MoBaaS and send data to the backend. These devices range between a sensor placed somewhere in the city to a wearable that anybody can use. The middle layer is the MoBaaS itself, which provides services via REST APIs. The MoBaaS provides many services, ranging from APIs which grant access to devices to APIs that provide events for those devices. For instance, if there are parking devices registered, the API will have access to the check-ins and check-outs of vehicles (i.e. the timestamp a vehicle entered and left a parking spot) during the time that the device is online. Additionally, it also offers a routing engine which intends to broaden the spectrum of possible applications to be developed. For example, it enables the possibility of having the optimal route to a specific parking spot. The MoBaaS also offers an API that is not directly related to the mobility scene, but that can be of great value for routing purposes. This API provides air quality events which allow for applications with optimal ecological routing to be designed.

Lastly, the top layer provides an easy way for everyone to integrate their applications via well-defined REST APIs and start using the MoBaaS.

6.3.4 nAssist

nASSIST is a software platform designed and conceived to allow agile, continuous management of data in the energy efficiency, security and automation fields. nAssist is a complete tool with which powerful solutions can be built whilst ensuring scalability, flexibility, integrity and system security. The platform integrates various drivers, embedded systems, SDKs, databases (NoSQL, MS Sql, Cloud Storage), Web applications, mMobile applications and other software components (Scheduler, Complex event processing unit and Event Manager). It is built following a Service Oriented Architecture paradigm and has

been designed to be easily adapted to different areas of application that use, or implicitly need, data collection and data processing from logical or physical devices (sensors and actuators).

nASSIST has modular design allowing for rapid expansion of the system. It is fully prepared for large networks situated in different locations with high reliability and redundant services as well as to support multiple customer. The platform interface has been developed from day one for individual client customizations and for access from all standard web browsers and mobile devices. nASSIST can be connected to different platforms that generate alarms and incidences, positioning or sensor measurements by means of specific drivers. The communication with hardware platforms is bidirectional, allowing remote control of devices and systems.

Within symbIoTe project, the nAssist platform will be used in Smart Residence use case as symbIoTe L1 Platform.

6.3.5 Navigo Digitale

The Navigo Digitale IoT Platform (ND) is a platform created to manage digital assets pertaining to harbours used for boating and yachting. Its scope embraces both physical entities (objects) and immaterial entities (documents and workflows).

It consists of a distributed platform, with instances associated to different ports across Europe and running part in the cloud, and part on premise.

The ultimate purpose of ND is to provide services to the harbour’s activities (B2B) and to its end-users (B2C).

The services of ND are created from the combination of functions and information made available by physical and immaterial entities.

For example, a service can be based on one or more workflows which are triggered and driven by data and actions provided by one or more physical objects. Such objects are either monolithic (e.g. a weather report station or a water/petrol station instrumented with remote control interfaces) or composite, that is "container" objects which in turn are made of different objects (e.g. a building or a yacht equipped with a control platform). In order to build its services, ND will need to be able to access all of these types of objects and to extract the relevant information they can provide in a seamless way.

If we abstract physical and immaterial entities and consider them all as "network controllable objects", we can see that ND is in a broad sense an IoT platform. Furthermore, since its objects can be in turn governed by "inner" IoT platforms, we may consider ND as an IoT meta-platform, or IoT hierarchical platform. For such objects, the data model seen by ND is linked to the foreign platform's data model for that specific object. For example, if a yacht (Vessel object in ND) is equipped with a symbIoTe Compliant system (e.g. Nextworks’ Symphony), the latter system’s data will be exposed in the ND Vessel object, including information about lights, sensors, engine control room’s monitoring, etc.

Access to inner IoT platforms will be possible for any symbIoTe Compliant Platform: for example, if ND needs to access a vessel’s fuel tank level (say, to propose a convenient fuel provider or suggest a route) and the vessel is equipped with a symbIoTe Compliant Platform, ND will be able to search for the resource, get authorization to access it, and communicate with the vessel’s RAP to retrieve the data. In the symbIoTe ecosystem, ND

has a functionality of an enabler, it uses symbIoTe core to access the resources offered by IoT platforms connecting devices in vessels.

6.4 Summary of symbIoTe position in the IoT ecosystem context

In this subsection, we summarize the main findings of the analysis comparing the symbIoTe approach and its proposed architecture with other projects and initiatives in the IoT space.

When comparing the symbIoTe architecture to the prominent reference architectures, we can conclude that the symbIoTe architecture is in line with both the AIOTA and IoT-A reference architectures. In fact, we can map the interfaces defined in AIOTI HLA to the symbIoTe functional components, while IoT-A functional groups have their counterparts in symbIoTe components. However, it should be noted that symbIoTe aims as implementing functionalities for IoT device discovery, look-up, and name resolution across different platforms which do not necessarily follow the IoT-A reference architecture, but rather decide to expose their devices as Virtual Entities accessible through REST-based interfaces.

The symbIoTe architecture is motivated by the oneM2M functional architecture, but symbIoTe extends the scope by identifying additional features, in particular those related to platform federations, bartering and trading as well as device roaming. In oneM2M, platforms are supposed to interact only through the Cloud Domain and a corresponding interface, but oneM2M does not provide many details on the particularities of platform-to-platform interaction. symbIoTe specifies this process in more detail by defining mechanisms for platform interaction (Bartering & Trading, SLA agreement). Additionally, symbIoTe provides platforms with the possibility to share their resources to third-party applications by using symbIoTe Core Services, which is a feature not envisioned in oneM2M.

When comparing symbIoTe to other related projects listed in Section 6.2, the major difference of symbIoTe is in the enlarged scope of interoperability concepts. Projects such as COMPOSE, CRYSTAL, iCore and a recent project FIESTA-IOT[20] focus on syntactic and semantic interoperability, but not on organizational interoperability. symbIoTe considers original features related to organizational interoperability by supporting the creation of IoT Federations for secure interoperation, collaboration and sharing of resources between two platforms, as well as IoT Device Roaming. There are also certain differences in the proposed approaches when comparing them to L1 Compliance as specified by symbIoTe. The major difference when comparing symbIoTe L1 solution to iCore is in the fact that iCore stores the data provided by platform devices, while symbIoTe stores only their metadata required for effective search. The COMPOSE project is not actually focusing on IoT platforms but rather on the engineering processes and allows data sharing between them, while CRYSTAL is targeting safety-critical systems.

[20] http://fiesta-iot.eu/

7 Conclusion

This document presents the final collection of the symbIoTe system requirements and reports the system's functional architecture, with the respective components, entities and interfaces. System requirements have been derived during an iterative process and based on symbIoTe use cases. They are related to a wide range of features across the IoT stack, from smart devices and gateways to cloud-based platform components and applications. Some important symbIoTe specific requirements are as follows: the system must support both sensors and actuators, and allow them to be mobile and change location; mobile devices should be able to interact with their surrounding environment in visited domains; access to both sensors and actuators is provided directly through the platforms managing those devices while symbIoTe serves as an intermediary between applications and platforms; access to platform devices needs to be authenticated and authorized.

The document sets the foundations of the symbIoTe functional architecture in the context of various interoperability aspects which are being supported by the symbIoTe interoperability framework (syntactic, semantic and organizational interoperability). symbIoTe defines an interoperability framework for IoT platforms and thus does not strive to become another "superplatform": it does not store any sensor-generated data outside of IoT platform boundaries, but rather acts as a mediator between applications and platforms ensuring secure and uniform access to platform resources through well-defined interfaces (CL1). It supports Platform Federations for secure interworking of collaborating platforms that want to barter/trade their resources (CL2). Moreover, it facilitates dynamic configuration of IoT devices in Smart Spaces (CL3) and roaming of IoT devices (CL4). The functional architecture is built around a layered stack in accordance with the AIOTA reference architecture, and defines four domains: Application, Cloud, Smart Space and Smart Device domain. It is motivated by the oneM2M architecture, but symbIoTe extends the scope by identifying features which go beyond the oneM2M functional architecture: These are related to platform federations, bartering and trading as well as device roaming.

In this document we have focused on defining the components for the Application and Cloud Domain, Smart Spaces and Smart Devices Domain based on the identified requirements. We have defined system behavior supporting syntactic and semantic interoperability and identified communication diagrams describing component interactions between platforms forming federations, thus enabling organizational interoperability. Furthermore, we specify communication diagrams depicting the interactions enabling dynamicity in Smart Spaces and roaming of IoT devices.

8 References

[1] H. van der Veer, A. Wiles. Achieving Technical Interoperability – the ETSI Approach. ETSI White Paper No.3, 3rd edition, April 2008

[2] IERC. IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps. Position Paper, March 2015

[3] Murdock, P., Elloumi, O. (eds). AIOTI High Level Architecture. Release 2.0, 2015

[4] Murdock, P., Elloumi, O. (eds). AIOTI High Level Architecture. Release 2.1, 2016

[5] Carrez, F. (ed). Final architectural reference model for the IoT v3.0. Release 3.0, July 2013

[6] oneM2M. The interoperability enabler for the entire M2M and IoT ecosystem. oneM2M whitepaper, 2015

[7] oneM2M. M2M Functional Architecture. Technical specification, 2016, URL: http://www.onem2m.org/images/files/deliverables/Release2/TS-0001-%20Functional_Architecture-V2_10_0.pdf

[8] IoT-A. Deliverable D1.5 – Final architectural reference model for the IoT v3.0. Technical specification, 2013

[9] J. Heuer, J. Hund, O. Pfaff. Toward the Web of Things: Applying Web Technologies to the Physical World. IEEE Computer, 48(5): 34-42, 2015

[10] W3C White Paper for the Web of Things, 2016, URL: https://www.w3.org/2016/09/IoTW/white-paper.pdf

[11] Industrial Internet Consortium. Industrial Internet Reference Architecture Technical Report, 2015, URL: http://www.iiconsortium.org/IIRA-1-7-ajs.pdf

[12] R. M. Soley. First European testbed for the Industrial Internet Consortium. Bosch Blog, 2015, URL: http://blog.bosch-si.com/categories/manufacturing/2015/02/first-european-testbed-for-the-industrial-internet-consortium/

[13] VDI/VDE, ZVEI. Reference Architecture Model Industrie 4.0 (RAMI4.0). Status Report, 2015, URL: http://www.zvei.org/Downloads/Automation/5305%20Publikation%20GMA%20Status%20Report%20ZVEI%20Reference%20Architecture%20Model.pdf

[14] International Organization for Standardization. ISO/IEC AWI/WD 30141/20.00 Internet of Things Reference Architecture (IoT RA). 2016, URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=65695

[15] COMPOSE project. Deliverable D1.2.2 – Final COMPOSE architecture document v1.0. Technical specification, 2014

[16] iCore project. Deliverable D2.3 Architecture Reference Model. Technical specification, 2013

[17] L. Macvittie. ABAC not RBAC, Welcomne to the (IoT) world of contextual security. 2015

[18] V. Hu, D.Ferraiolo, R. Kuhn, A. Schnitzer, K.Sandlin, R.Miller, K.Scarfone. Guide to Attribute Based Access Control (ABAC) - Definition and Considerations. NIST Special

Publication, 2014, URL: http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf

[19] Zhou Y, De S, Wang W, Moessner K. Search Techniques for the Web of Things: A Taxonomy and Survey. Sensors, 16(5), p. 600, 2016

[20] COMPOSE – Collaborative Open Market to Place Objects at your SErvice, URL: http://www.compose-project.eu/, access: September 2016

[21] CRYSTAL – Critical sYSTem engineering AcceLeration, URL: http://www.crystal-artemis.eu/, access: September 2016

[22] iCore – Empowering IoT through Cognitive Technologies, URL: http://www.iot-icore.eu/, access: September 2016

[23] Crystal project. Deliberable D601.021: Interoperability Specification v1, 2014

[24] Fi-ware project. Deliverable D2.2: High-level Description. Technical specification, 2011

[25] G. Malim. Looking for a Benchmarking Framework for IoT platforms. IoT global network, 2016, URL: http://www.iotglobalnetwork.com/iotdir/2016/02/16/looking-for-a-benchmarking-framework-for-iot-platforms-1031/

[26] Network Working Group. Key words for use in RFCs to Indicate Requirement Levels. Request for Comments 2119, 1997, URL: https://www.ietf.org/rfc/rfc2119.txt

[27] IEEE Standards Association. IEEE Standard for Information Technology – Systems Design – Software Design Descriptions. Active standard, 2009, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5167255

[28] J. Kim *et al*., "Standard-based IoT platforms interworking: implementation, experiences, and lessons learned," in *IEEE Communications Magazine*, vol. 54, no. 7, pp. 48-54, July 2016. doi: 10.1109/MCOM.2016.7514163

9 Abbreviations

AA	Authentication and Authorization
AAM	Authentication and Authorization Manager
ABAC	Attribute Based Access Control
APP	Application Domain
B&T	Bartering & Trading component
CL	symbIoTe Compliance Level
CLx (1-4)	Level-x (1 to 4) symbIoTe Compliance
CLD	Cloud Domain
DoA	Description of the Action
GA	Grant Agreement
IIRA	Industrial Internet Reference Architecture
IoE	Internet of Everything
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
RA	Reference Architecture
RAP	Resource Access Proxy
RAM	Resource Access Monitor
RAMI4.0	Reference Architecture Model Industrie 4.0
RM	Resource Monitor
S3	symbIoTe Smart Space Middleware
SDEV	Smart Device Domain
SLA	Service Level Agreement
SLO	Service Level Objective
SOTA	State of the art
SSP	Smart Space Domain
QoS	Quality of Service