Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Initial 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: Initial Report on System Requirements and Architecture

Type: Public

Editor(s): Pavle Skočir

E-mail: pavle.skocir@fer.hr

Author(s): Ivana Podnar Žarko, Konstantinos Katsaros, Mario Kušek, Elena Garrido Ostermann, Matteo Pardi, João Garcia, Aleksandar Antonić, Zvonimir Zelenika, Artur Jaworski, Michal Pilc, Christoph Ruggenthaler, Luca de Santis

Doc ID: D1.2-v2.0

Amendment History

Version	Date	Author	Description/Comments
v0.1	30/06/2016	Pavle Skocir (UNIZG-FER)	Initial Table Of Contents
v0.2	22/07/2016	Ivana Podnar Žarko (UNIZG-FER)	Updated ToC
v0.3	24/08/2016	Ivana Podnar Žarko (UNIZG-FER)	Updated ToC with assignments and links to confluence pages which will provide the initial input for the corresponding sections
v0.4	01/09/2016	Konstantinos Katsaros (ICOM), João Garcia (UW), Ivana Podnar Žarko (UNIZG-FER), Aleksandar Antonić (UNIZG-FER), Pavle Skočir (UNIZG-FER)	Updated ToC with assignments for Section 4, added content in Sections 3.2 & 3.3, Sections 6.1.1, 6.1.2, 6.1.4, 6.2.1 & 6.3.3
v0.5	08/09/2016	Pavle Skočir (UNIZG-FER), João Garcia (UW), Ivana Podnar Žarko (UNIZG-FER), Zvonimir Zelenika (VIP)	Added content in Section 5 and 5.1.1, updated Section 6.2.1 (FIWARE), added content to Sections 6.1.5, 6.1.6, 6.1.7, inserted new section in the Appendix Section 10.3 (IIRA overview)
v0.7	09/09/2016	Matteo Pardi (NXW), Michal Pilc (PSNC), Artur Jaworski (PSNC), Christoph Ruggenthaler (AIT), Konstantinos Katsaros (ICOM)	Added content in Sections 3.1, 4.1, 4.4, 5.3, 5.4, 6.1.3, 6.1.4, 6.1.5, 6.3.2
v0.8	13/09/2016	Elena Garrido Ostermann (ATOS), Luca de Santis (Navigo), Konstantinos Katsaros (ICOM), Pavle Skočir (UniZG-FER)	Added content in Sections 2, 4, 5.2, 6.3.1, 6.3.4, 6.3.6
v0.9	14/09/2016	Ivana Podnar Žarko (UniZG-FER), Pavle Skočir (UniZG-FER), Matteo Pardi (NXW)	Added content in Sections 5.1, 5.3, 6.2.2, 6.3.5, updated Sections 4, 5.2
v0.10	15/09/2016	Mario Kušek (UNIZG-FER), Pavle Skočir (UNIZG-FER)	Added content in Sections 5.5, 6.2.3, 6.2.4
v1.0	16/09/2016	Ivana Podnar Žarko (UNIZG-FER), Zvonimir Zelenika (VIP), Aleksandar Antonić (UNIZG-FER)	Updated content in Sections 2, 3, 5, 6.1.1, 6.1.6, 10.1, 10.3,
v1.1	16/09/2016	Pavle Skočir (UNIZG-FER)	Updated content in Section 5.1.2
v1.2	22/09/2016	Konstantinos Katsaros (ICOM), Elena Garrido Ostermann	Updated Section 4, 5.2.2, updates throughout the document according to comments from reviewers
v1.3	27/09/2016	Konstantinos Katsaros (ICOM), Ivana Podnar Žarko (UNIZG-FER), Pavle Skočir (UNIZG-FER)	Added subsection in Section 4, update of Sections 1, 2, 3, and 5.2
v2.0	29/09/2016	Ivana Podnar Žarko, Pavle Skočir, Konstantinos Katsaros	Update of all sections

Table of Contents

Table of Figures

Table of Tables

1 Executive Summary

The aim of Deliverable 1.2, entitled “Initial Report on System Requirements and Architecture”, is to document the initial collection of the symbIoTe system requirements and report a first 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 D1.1, 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 an initial collection of the system requirements with a focus on two domains: Application Domain and Cloud Domain. These two domains need to 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 securely interoperate, collaborate and share/trade devices and associated services.

The aforementioned requirements represent the main symbIoTe requirements that have served as drivers to create an initial version of the *symbIoTe functional architecture*. However, the list of identified requirements is extensive since it has been derived from five symbIoTe use cases. We report a comprehensive list of requirements and put them in relation to software components for the Application and Cloud Domain. 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 sequence 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* 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

[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”.

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 AIOTI reference architecture. It is motivated Figure 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.

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.

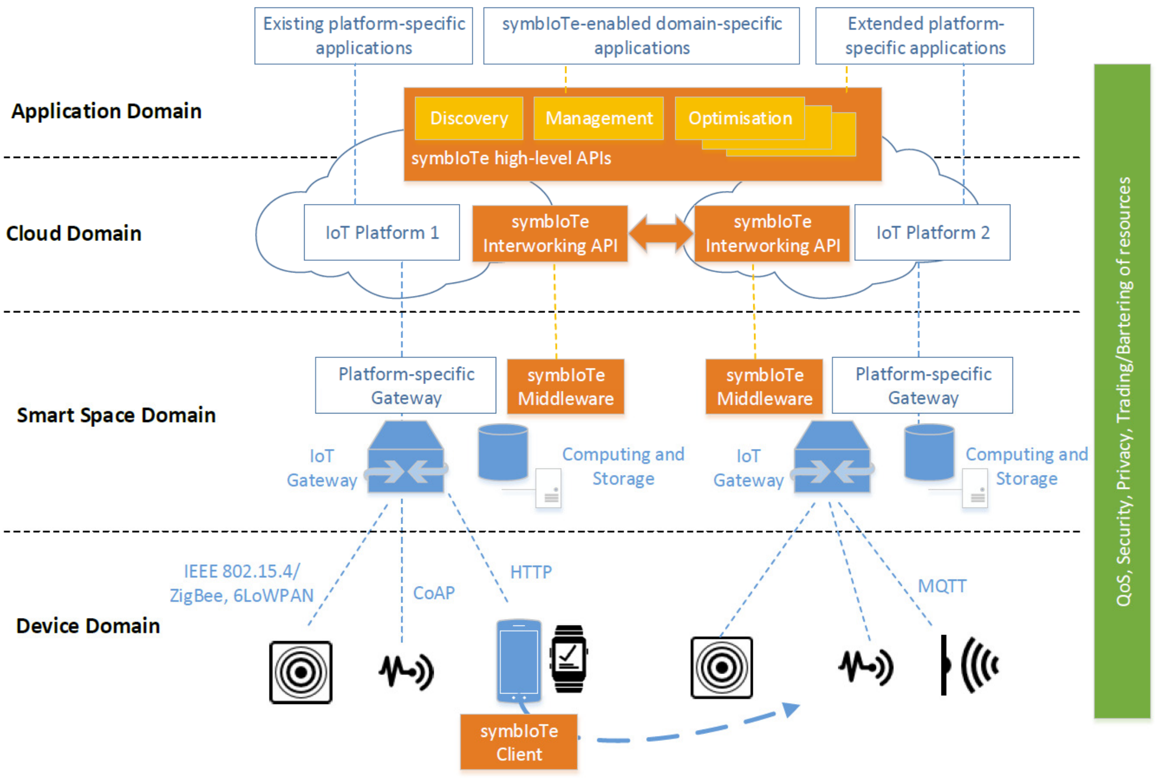
Existing platform-specific applications
symbIoTe-enabled domain-specific applications
Extended platform-specific applications
Application Domain
Discovery
Management
Optimisation
symbIoTe high-level APIs
Cloud Domain
IoT Platform 1
symbIoTe Interworking API
symbIoTe Interworking API
IoT Platform 2
Smart Space Domain
Platform-specific Gateway
symbIoTe Middleware
symbIoTe Middleware
Platform-specific Gateway
IoT Gateway
Computing and Storage
IoT Gateway
Computing and Storage
Device Domain
IEEE 802.15.4/ ZigBee, 6LoWPAN
CoAP
HTTP
MQTT
symbIoTe Client
QoS, Security, Privacy, Trading/Bartering of resources

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 D1.2 "Initial Report on System Requirements and Architecture" is to document the initial collection of the system requirements and a first version of the system's functional architecture, based on the current progress in tasks T1.3 and T1.4. The document reports the first version of system requirements and functional architecture with focus on two symbIoTe domains: Application Domain and Cloud Domain. System requirements have been carefully derived from the use case descriptions reported in deliverable D1.1 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 and Cloud Domain. Furthermore, this document reports on the symbIoTe security mechanisms which are incorporated into component descriptions and corresponding sequence diagrams, as well as an initial view on components and envisioned functionality needed for the Smart Space and Smart Device Domain.

This document will be used as input to 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, will use inputs from this deliverable as a starting point for defining trading and bartering mechanisms (T3.1), security and access scopes (T3.2), and will specify the design and implement components enabling IoT federations (T3.3). WP4 will further elaborate on the initial vision describing the Smart Space and Smart Sevice Domains presented in this document.

Tasks T1.3 (lasting until M12) and T1.4 (lasting until M18) will in the next period focus on requirements and architecture for the Smart Space and Smart Device Domain, as well as further refine (if needed) component descriptions for the Application and Cloud Domains. In the next period we also expect further elaboration of enablers, while their design and implementation is performed in task T2.3. The next version of this report (deliverable D1.4, due in M18) will list all identified requirements and provide the functional architecture of the entire symbIoTe system.

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].

- **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 an 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 through a set of attributes, potentially assigned by different platforms at different time instants. One or more attributes are carried within tokens and they 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**: four interoperability aspects coverd within the symbIoTe project.
- **Level-1 Compliant IoT Platform** (L1): offers an open symbIoTe-defined platform interface; platform devices are searchable within the symbIoTe Core Services.
- **Level-2 Compliant IoT Platform** (L2): implements functionality needed for platform federation.
- **Level-3 Compliant IoT Platform** (L3): supports dynamic smart spaces.
- **Level-4 Compliant IoT Platform** (L4): supports device roaming in visited domains; a smart device can use services in a visited smart space.
- **IoT Platform Federation:** an association of two 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 platforms.
- **Interworking Interface**: symbIoTe defined interface which opens up platform resources as IoT Services in the Cloud Domain.

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).

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 fulfill 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 reported in this document are derived from deliverable D1.1 “Initial Report on Use Cases” while the architecture definition is in line with the initial vision of symbIoTe architecture presented in DoA. The process of defining system requirements and architecture follows a two-step approach: Initial requirements and architecture are created in the first step with focus on Application Domain and Cloud Domain (reported in this deliverable D1.2). The second step will be reported in D1.4 “Final Report on System Requirements and Architecture” with focus on Smart Space and Smart Device requirements and architecture.

This deliverable **introduces the functional architecture of symbIoTe**. The symbIoTe Information model, which is being developed in T2.1 in parallel with tasks T1.3 and T1.4, will be presented in D2.1 (M10) “Semantics for IoT and Cloud Resources.” Domain-specific enablers mentioned in this deliverable will be elaborated on 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 introduced in this document will be documented in more detail within D3.1 (M11) “Resource Trading Mechanisms and Access Scopes”. An initial vision of Smart Spaces and Smart Devices presented within this deliverable will be extended through activities in WP4 (Dynamic Smart Spaces) and reported in D4.1 (M12) “symbIoTe Middleware Tools, Protocols and Core Mechanisms.”

2.4 Document structure

Section 2 presents the purpose of this document, as well as its relation to other deliverables within the project. Section 3 elaborates on symbIoTe goals and challenges, and provides details of symbIoTe domains and their relation to the four symbIoTe-specific interoperability aspects referred to as *Compliance Levels*. System requirements are presented in Section 4, while system architecture with the respective components, entities and interfaces is introduced in Section 5. Section 6 describes state of the art overview, with focus on reference architectures by standardization bodies, projects with similar goals as symbIoTe, and platforms by symbIoTe partners aiming to become 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. Considering the fact that there are many different IoT platforms already in the market, with various requirements and business goals, symbIoTe proposes a flexible interoperability concept (translated into four interoperability-related Compliance Levels) so that platform providers can choose an appropriate one to match their 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 in the market[2], there is a need for a viable interoperability framework enabling platform cooperation so that a new generation of IoT services and applications is 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 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 the 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 aims to design and develop 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 applications.
- Device discovery across various IoT platforms: symbIoTe needs to offer search mechanisms to efficiently discover devices across platforms which are accessible to third parties.
- Security: access scopes and identity management represent key requirements for authorized access to varios devices across platforms.
- 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.

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

- 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 which 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 will 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 devices which they want to advertise and make accessible via symbIoTe to third parties, while symbIoTe provides the means for discovery of IoT devices across platforms. It also hosts domain-specific back-end services (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 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 well- documented 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, which are managed by a visited platform, in accordance with predefined access policies.

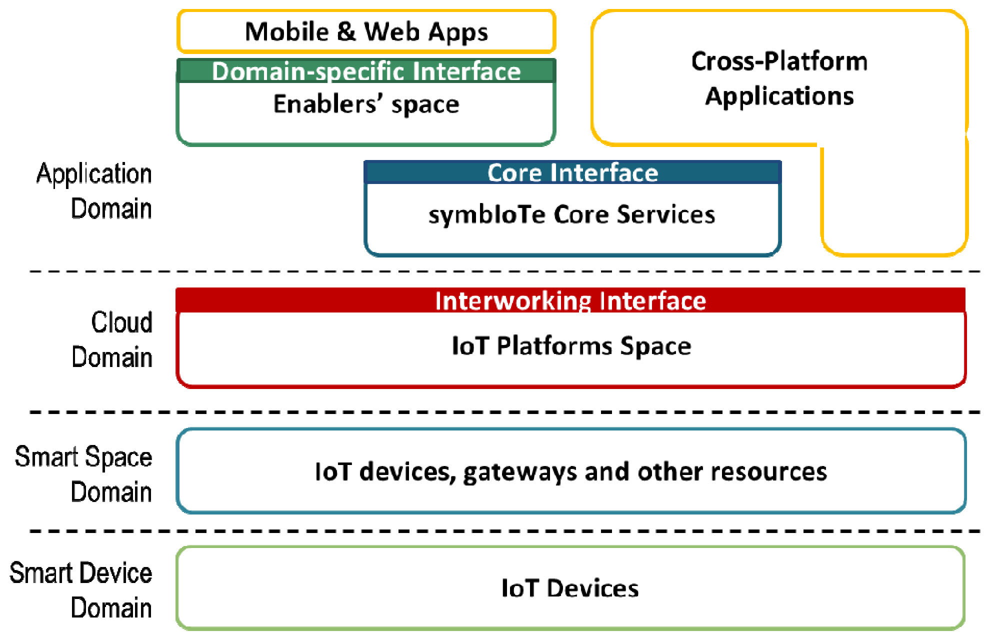

Figure 2 The symbIoTe high-level architecture

APP is designed to offer a unified view on different platforms to **a new generation of cross-platform IoT applications**. This is achieved by the symbIoTe Core Services which enable the discovery of 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] so that resources are searchable across platforms in a uniform way. 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. 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 the search functionality, the Core Services implement symbIoTe specific authentication and authorization methods providing the means for secure access to underlying platform-specific 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 model will be specified in deliverables D2.1 "Semantics for IoT and Cloud Resources" and D2.4 "Revised Semantics for IoT and Cloud Resources".

CLD hosts the Cloud-adjusted building blocks of specific platforms (e.g., a data store, data analytics and stream processing tools, tools for platform management, etc.). To enable a unified and secure access to platform resources, the symbIoTe Interworking Interface and corresponding API is defined and implemented to expose open platform resources which have previously been registered with the symbIoTe Core Services. We mark such IoT resources as *open symbIoTe-enabled resources*. CLD services also implement specific functionality for the exchange of information between two collaborating IoT platforms, e.g., for bartering and trading between platforms, and a specific protocol for the exchange of information between platforms.

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 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, possibly at the gateway level. The symbIoTe middleware exposes a standardized API for resource discovery and configuration of devices within a Smart Space, 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 smart space, 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 properties.

3.3 Compliance Levels

symbIoTe allows for flexible interoperability mechanisms which can be achieved by introducing an incremental deployment of symbIoTe functionality across the listed architectural domains (APP, CLD, SSP and SD). 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.

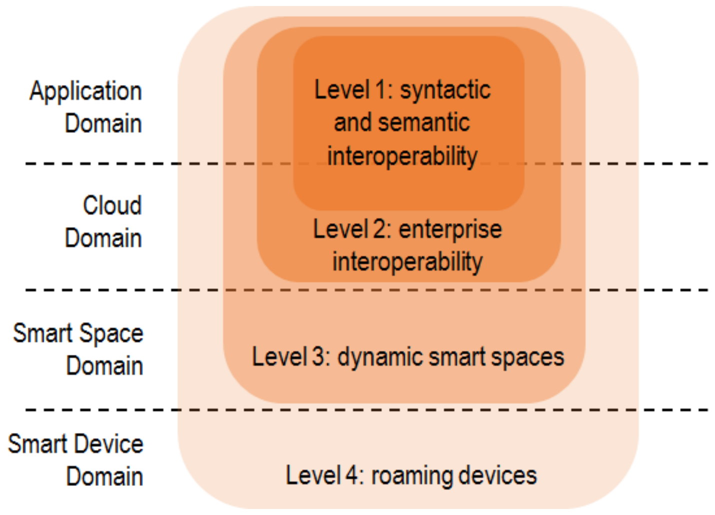
Application Domain
Cloud Domain
Smart Space Domain
Smart Device Domain
Level 1: syntactic and semantic interoperability
Level 2: enterprise interoperability
Level 3: dynamic smart spaces
Level 4: roaming devices

Figure 3 symbIoTe Compliance Levels

We define four different Compliance Levels for IoT platforms, as depicted in Figure 3. They reflect different interoperability modes, which an IoT platform can support. Different interoperability modes affect the functionality which needs to be supported by platforms, and require specific symbIoTe components to be integrated within different domains.

- **Level 1 symbIoTe Compliant Platform (L1 Platform)**: This is a "lightweight" symbIoTe Compliance Level 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 Compliance Level 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).

Note that L1 Compliance 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 L2, in the SSP for L3 as well as in both SSP and SDEV for L4 Compliance. 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 L2 Compliance, a platform should first adhere to L1 Compliance, while an L4 platform requires a full symbIoTe framework (i.e., an L4 platforms is also L1, L2 and L3 Compliant).

L1 Compliance relates to services placed in two domains, APP and CLD, as depicted in Figure 4. IoT platforms which want to become part of the symbIoTe ecosystem need to integrate the symbIoTe Interworking Interface with its existing components running in the Cloud Domain. This will enable semantic interoperability and open access to IoT services which a platform chooses to register and make discoverable via the symbIoTe Core Services. The access to those devices will stay under the control of a platform provider, while being enabled through the Interworking Interface. In addition to open access, symbIoTe also supports authenticated and authorized access to IoT services (more information is provided in Section 5.4).

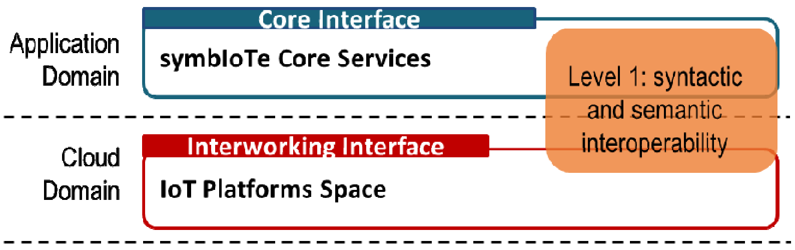

Figure 4 Level 1 Compliance

Figure 5 shows the benefits of symbIoTe and its interoperability concept by an example depicting two L1 platforms (platform A and platform B) using the symbIoTe Core Services. The figure depicts the process when an application searches for IoT devices, and after it identifies the adequate ones, the application accesses the identified devices offered by two platforms through the interworking interface. In other words, cross-platform applications i) use the symbIoTe Core Services to find adequate devices across platforms and ii) access, integrate and use those devices 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.

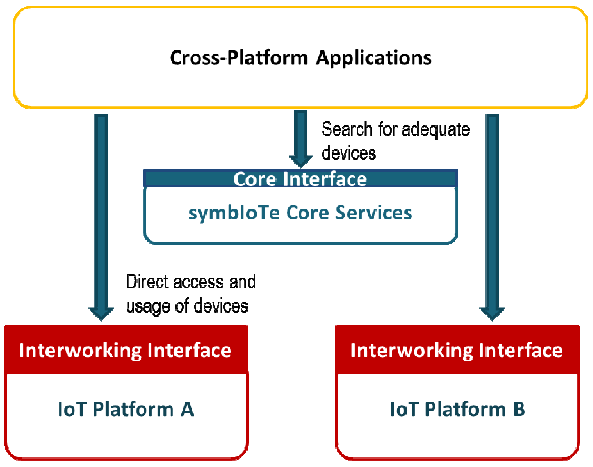

Figure 5 Platform cooperation: Level 1 Compliance

L2 Compliance involves components placed both in APP and CLD, but requires a significant extension of an existing platform deployment to enable a closer collaboration between two platforms, i.e., platform federation. This collaboration should adhere to a specified SLA and support symbIoTe-specific bartering and trading mechanisms. Figure 6 illustrates an example platform federation where it is possible to expose certain IoT services from platform A within the space of platform B. This creates an opportunity that an existing application ("native application") expands the set of available services within platform B since they appear as "native services" to an existing application built exclusively for platform B.

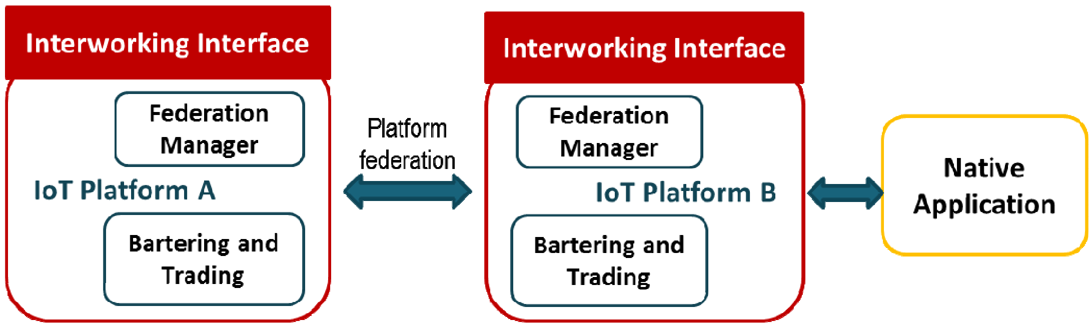

Figure 6 Platform collaboration: Level 2 Compliance

L3 and L4 Compliance mainly affect platform software which is deployed within a Smart Space, and may require specific software also at the level of IoT devices. Features similar to the ones appearing in APP and CLD are needed within SSP, but with quite a different and reduced scope since the platform needs to manage local resources (devices, storage, wireless medium) within the same environment. We assume that a number of platforms can occupy the same smart space. L3 Compliance refers to dynamic reconfiguration of devices within a Smart Space, so that a new device is reconfigured on the fly to become part of an SSP deployment within the smart space. L4 Compliance relates to interoperability at the smart device 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 3.1). 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 Compliance Levels thus resulting in the Application Domain, Cloud Domain and Smart Space and Smart Device Domain requirements. During the reported period, the project first focused on the specification of the Application and Cloud Domain requirements, so as to support the corresponding architectural design. However, a preliminary investigation of the requirements for the Smart Space and Smart Device Domains has also been conducted. 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.

In order to structure the specification of the system requirements, we have identified the following set of requirement attributes.

Domain: Following the structure of the symbIoTe stack, each requirement relates to one or more domains, namely: Application, Cloud, Smart Space, Smart Device. In several cases, requirements may apply to more than one domains. These are reported within the set of the highest Compliance Level / domain 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 architecure 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 withing 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 fulfillment 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.

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,
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 Compliance Levels. 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 Level 1 Compliance, the iterative process included the following steps:

Step 1: Revisit the previous level's requirements within the current Compliance Level (if applicable).

The purpose of this step is to identify requirements that pertain across Compliance Levels. Such requirements bear the potential of leading to more a efficient architectural design that identifies key functional components across the considered domains, further promising a modular design.

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 consolidates all comments and identifies 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
- Security requirements

4.3 Specified requirements

Table 1 below lists the 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 formed security team.

Table 1: System requirements

Index	Domain	Type	Category	Importance	Description	Use Cases
1	Application, Cloud	Functional	Interface	MUST	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	Application	Functional	Interface	MUST	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	Application, Cloud	(Non-)Functional	Interface	MUST	The system MUST support a common information model for the description of available IoT services across IoT platforms.	1,2,3,4,5
4	Application	Non-Functional	Interface	SHOULD	IoT services SHOULD appear to application developers in a homogeneous manner i.e., the interface for application developers should not differentiate across IoT platforms. Data source/identity shall be exposed to application developers.	1,2,3,4,5
5	Application, Cloud	Functional	Monitoring	MUST	The system MUST monitor the availability of the IoT services registered by IoT platform operators.	1,2,3,4,5
6	Application, Cloud	Functional	Monitoring	SHOULD	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	Application, Cloud	Functional	Monitoring	MAY	The system MAY perform functional performance tests on registered IoT services.	2,3,4
8	Application, Cloud	Functional	Management	MUST	The system MUST not allow the allocation of IoT services to applications or other IoT platforms if their availability and the potentially associated QoS cannot be guaranteed by the underlying IoT platforms. This requirement applies to IoT service types with exclusive access rights (e.g., actuators).	2,3
9	Application, Cloud	Functional	Management	MAY	The system MAY be able to reserve IoT services and associated resources for the future, for IoT platforms that support such feature.	2

10	Application, Cloud	Functional	Management	MUST	The system MUST support the exclusive use of IoT services , for underlying IoT platforms that support such feature.	2,3,4
11	Application, Cloud	Non-Functional	Interface	MUST	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
12	Application, Cloud	Functional	Interface	MUST	The common information model MUST support geo-reference information.	1,2,3,4,5
13	Application, Cloud	Functional	Management	SHOULD	The system SHOULD enable the prioritization access to IoT services if the underlying IoT platforms support prioritization, e.g., users with premium rights or who paid for access have priority over basic users when attempting concurrent access to some IoT services.	3
14	Application, Cloud	Functional	Management	SHOULD	The system SHOULD enable the control of access to the advertised IoT services for reasons related to local legislation, current overload, security issues, etc., if the underlying IoT platforms support it.	1,3,4,5
15	Application, Cloud	Non-Functional	Interface	SHOULD	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
16	Application, Cloud	Functional	Interface	MUST	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
17	Application, Cloud	Functional	Interface	SHOULD	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
18	Application, Cloud	Functional	Interface	MUST	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
19	Application, Cloud	Functional	Interface	MUST	symbIoTe MUST distinguish IoT devices which are fixed (geo-location does not change over time) and mobile (their location changes).	1,3,4,5
20	Application	Functional	Management	MUST	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. The system must manage all the underlying resources, include the required logic, ensure the required quality, performance, etc (see Requirements 2, 5-12)	1,2,3,4,5
21	Application	Functional	Management	SHOULD	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	1,2,3,4,5

					least to their creators.	
22	Application	Functional	Management	MAY	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
23	Application, Cloud	Functional	Interface	MUST	The system MUST recommend IoT services based on the search criteria defined by an application developer or IoT platform provider.	1,4,5
24	Cloud	Functional	Management	MUST	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
25	Application, Cloud	Non-Functional	Performance	SHOULD	Registration response time: SHOULD be in the order of minutes, depending on the volume of registered IoT services.	1,2,3,4,5
26	Application, Cloud	Non-Functional	Performance	SHOULD	Search response time: SHOULD scale with the volume of results data and the volume of available data (with an upper limit). It should be in the order of a few seconds.	1,2,3,4,5
27	Application, Cloud	Non-Functional	Performance	SHOULD	IoT service access latency: SHOULD have an upper limit of maximum 1-2 seconds.	1,2,3,4,5
28	Application, Cloud	Non-Functional	Performance	SHOULD	Volume of IoT services supported: the system SHOULD target large volumes of meta-data (big data) provided by IoT platforms.	1,3,4
29	Application, Cloud	Non-Functional	Performance	SHOULD	Number of IoT platform instances/enablers: the system SHOULD scale in the order of thousands of instances.	2,3,4
30	Application, Cloud	Non-Functional	Performance	SHOULD	Number of applications/enablers: it SHOULD scale in the order of thousands of instances.	2,3,4
31	Application, Cloud	Non-Functional	Performance	SHOULD	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
32	Application, Cloud	Non-Functional	Performance	SHOULD	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
33	Application, Cloud	Functional	Management	MUST	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
34	Application, Cloud	Functional	Management	SHOULD	The system SHOULD enable IoT platforms to control whether their IoT services 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.	1,2,3,4,5
35	Application, Cloud	Functional	Management, Interface	MAY	The system MAY enable IoT platforms to define access rules to their IoT services during the registration process. Such access rules refer to the intended availability of the IoT services to applications/enablers e.g., maxium 10 times per day, only from 7p.m. to 7a.m..	
36	Application,	Functional	Management	MAY	The system MAY periodically check the long term availability of	1,3,4

	Cloud				registered IoT services with the purpose of purging or invalidating the corresponding registrations.	
37	Application, Cloud	Functional	Management	MAY	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
38	Application, Cloud	Functional	Management	MUST	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 operation the application receives the data whenever this is pushed (published) by the corresponding IoT device.	1,2,3,4,5
39	Application, Cloud	Functional	Management	SHOULD	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
40	Application, Cloud	Functional	Interface	MUST	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
41	Application, Cloud	Functional	Management	MAY	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
42	Application, Smart space	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
43	Application, Cloud, Smart space	Functional	Management	MUST	The system MUST support an 'attribute mapping functionality' through which it is possible to map attributes generated/released in one IoT domain to the same/similar attributes valid in different IoT domains.	1,2,3,4
44	Application, Cloud	Functional	Interface	MUST	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
45	Application,	Functional	Interface	SHOULD	Enablers SHOULD be regarded as high-level IoT platforms that can	1,2,3,4,5

	Cloud				register their domain-specific services to the system, similar to native IoT platforms.	
46	Application, Cloud	Functional	Management	MUST	Bartering and trading MUST be available for B2B and B2C transactions.	2,3,4
47	Cloud	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	Cloud	Functional	Management	MUST	Forward auctions MUST be available.	3,4
49	Cloud	Functional	Management	SHOULD	Reverse auctions SHOULD be available.	3,4
50	Application, Cloud	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	Cloud	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	Cloud	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	Cloud	Functional	Management/Security	SHOULD	The B&T transactions (B2C) SHOULD assure the anonymity of end users.	3,4
54	Smart Space	Functional	Interface	MUST	The system MUST enable the discovery and registration of a new device that is willing to register with symbIoTe compatible platform middleware.	1,2,3,5
55	Smart Space, Smart Device	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	Smart Space	Functional	Management	SHOULD	The system SHOULD be able to prioritize the information send in the platform (IMPORTANT information 1st)	1,3,5
57	Smart Space	Non-Functional	Interface	SHOULD	The system SHOULD support the dynamic configuration of a subset of commercial sensors.	1,3
58	Smart Space	Functional	Interface	MAY	Inside Smart Space multiple gateways MAY be used as an alternative fallback router for a given device.	1,2,3
59	Smart Space	Functional	Management	SHOULD	SymbIoTe smart spaces SHOULD be able to operate without a permanent Internet connection. (see Security Requirement 22)	1,2,3,5
60	Smart Space	Functional	Management / Interface	MAY	Different local IoT Platforms MAY be able to interact locally (i.e. without mediation from cloud-based L2 symbIoTe components).	1,2,3,5
61	Smart Space	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	Smart Space	Functional	Management	SHOULD	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	Smart Space	Functional	Management	MUST	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	Smart Space	Functional	Management	MUST	An L4-Compliant Smart Device MUST have a globally unique identifier.	1,3
65	Smart Space	Functional	Management / Interface	SHOULD	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	Smart Space	Functional	Management / Interface	SHOULD	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	Smart Space, Smart Device	Functional	Management/Interface	MUST	SymbIoTe MUST accept visiting devices to be merged in the visited Smart Space.	1,2,3,5
68	Cloud	Non-Functional	Performance	SHOULD	The establishment of an SLA SHOULD complete within 1 sec, when an offering is already available.	3,4,5
69	Application, Cloud	(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
70	Application, Cloud	(Non-)Functional	Interface	SHOULD	The system SHOULD support mappings between two different extensions of the common information model.	1,2,3,4,5

Table 2: Security system requirements

Index	Domain	Type	Category	Importance	Description	Use Cases
1	All	Functional	Security	MUST	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	All	Functional	Security	MUST	The system MUST offer mechanisms for the **authorization** of symbIoTe entities/actors i.e., users/application developers, IoT platforms, developed applications and clients.	1,2,3,4,5
3	Application	Functional	Security	MUST	The SymbIoTe ecosystem must offer mechanisms to establish trust relationships - and thus implicitly trust levels - prior to applying security mechanisms for the first time. This information must be stored in a secure datastore. e.g. by PKI infrastructure. (E.g. online or offline means for verifying the true identity of the respective user/platform/software/...? shall be defined and put in place)	2
4	Smart Space	Functional	Security	SHOULD	The authentication to a smart space SHOULD work even if the smart space is disconnected from the Internet.	1,2,3,5
5	Application/Cloud	Functional	Security	MUST	The system MUST support the revocation of access rights to users/application developers, IoT platforms. *(Comment: Although in the*	1,2,3,4

					Yachting use case it might only be revoked when the system comes online again.)	
6	Application/Cloud	Functional	Security	MUST	The system MUST explicitly support access rights expiration.	1,2,3,4
7	Cloud	Functional	Security	MUST	The authentication mechanisms of the system MUST support identity federation (e.g. single sign-on).	1,2,3,4
8	Application / Cloud	Functional	Security	MUST	The system MUST preserve end-user privacy. (E.g. locations of end users / sent sensor data and their identity, e.g. via data anonymization)	1,2,3,4,5
9	Smart Space	Functional	Security	MUST	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	Smart Space	Non-Functional	Security	MUST	The system MUST ensure privacy protection on each layer, do not publicly expose e.g., devices information or application used by applications.	1,2,3
11	Application, Cloud	Functional	Security	MUST	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	Application / Cloud	Non-Functional	Security	MAY	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	Application / Cloud	Functional	Security	MUST	The system MUST provide the possibility to let users / entities choose where (enablers/IoT platforms) their data is being used and processed. The users/entities MUST be able to modify the privacy parameters regarding their data.	1,2,3,4
14	Application/Cloud	Non-Functional	Security	SHOULD	symbIoTe SHOULD detect and propagate any security error notifications through the system to application/enablers/end user.	1,2,3,4,5
15	Application	Non-Functional	Security	MAY	The terminology used to describe the system 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.	
16	All layers	Functional	Security	MUST	Access rules MUST be defined as an access policy.	
17	All layers	Functional	Security	MUST	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	Application/Cloud	Functional	Security	MUST	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	Application	Functional	Security and Usability	MAY	Symbiote MAY support Multi-Factor Authentication towards if the underlying platform supports it. (e.g. Authentication using password and PIN)	
20	Application	Functional	Security	MUST?	Mutual authentication must be supported by all security mechanisms. (I.e. NOT only the user/application/software/... must be authenticated against the platform but also vice versa in order to facilitate malicious platform detection)	4
21	Application/Smart space	Functional	Security	MUST	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 (MAY)
22	Smart Space	Functional	Security	MUST	Within smart spaces it must be possible to run a local symbIoTe core instance for privacy and security reasons (e.g. a symbIoTe core instance installed in a smart residence). This instance might not be connected to the Internet (but could). If connected to the Internet it might expose sensors to another remote symbIoTe core instance.	2,3(SHOULD)

5 symbIoTe Architecture

This section presents details of the symbIoTe architecture, with focus on two domains: Application Domain (APP) and Cloud Domain (CLD), as well as the functionality required for L1 Compliance. We introduce an extensive list of components and associated features which have been identified for APP (Section 5.1) and CLD (Section 5.2) 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 functions or Compliance Level they enable. Currently, the APP contains mostly components for L1 Compliance and some components required for L2 Compliance, while the CLD hosts mainly components required for L2 Compliance and some for L1 Compliance. As we further progress with identification of features required for L2, L3 and L4 Compliance, new components as well as component extensions will be defined with proper labeling of provided functionality with respect to symbIoTe Compliance Levels.

Section 5.3 presents an analysis of requirements for the Smart Space (SSP) and Smart Device Domains (SDEV) to identify a list of components for the two domains as well as to put them in context with APP and CLD components. Since security requirements play a vital role for symbIoTe, Section 5.4 describes and analyses the symbIoTe approach to security[4]. The final subsection goes a step further towards component design for the APP and CLD components and puts both the required APP and CLD components as well as security-related technical decisions into the context of L1 Compliance. We introduce interface definitions and sequence diagrams depicting component interactions to achieve semantic and syntactic interoperability within symbIoTe in Section 5.5.

Table 3 Template for component description

Component	**Name of the component**
Compliance Level	L1, L2, L3, or L4
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 will enable symbIoTe to become a mediator between applications and IoT platforms so that applications can use platform devices

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

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 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 L1 Compliance 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 list 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, but may represent a weakness for the symbIoTe ranking function if it does not understand the details about the resources. We envision that a mapping solution will be available to map an information model which symbIoTe does not understand to the one which symbIoTe understands. Further details regarding the symbIoTe information model and approach to semantics will be provided in deliverable D2.1.

The components deployed in APP shown in Figure 7 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;

[5] We use the term resource hereafter to reffer to various addressable services offered by both IoT platforms and enablers, as defined in Section 2.2
[6] https://thingful.net/
[7] https://www.shodan.io/

- **Core Resource Monitor (Core RM):** tracks availability of registered resources in order to ensure their availability;
- **Core Resource Access Monitor (Core RAM):** tracks information about resource popularity, i.e., which registered resources are being used by applications/enablers;
- **Core Authentication and Authorization Manager (Core AAM):** ensures that trusted platforms register resources with symbIoTe, while mapping resource access rights to proper credentials;
- **Core Security Handler (Core SH):** provides a set of generic security-related features required for the attribute based access control;
- **Core Bartering and Trading Component (Core BarT):** comprises all bartering and trading functionalities for L2 Compliance that need to be centralized, and are thus deployed within APP.

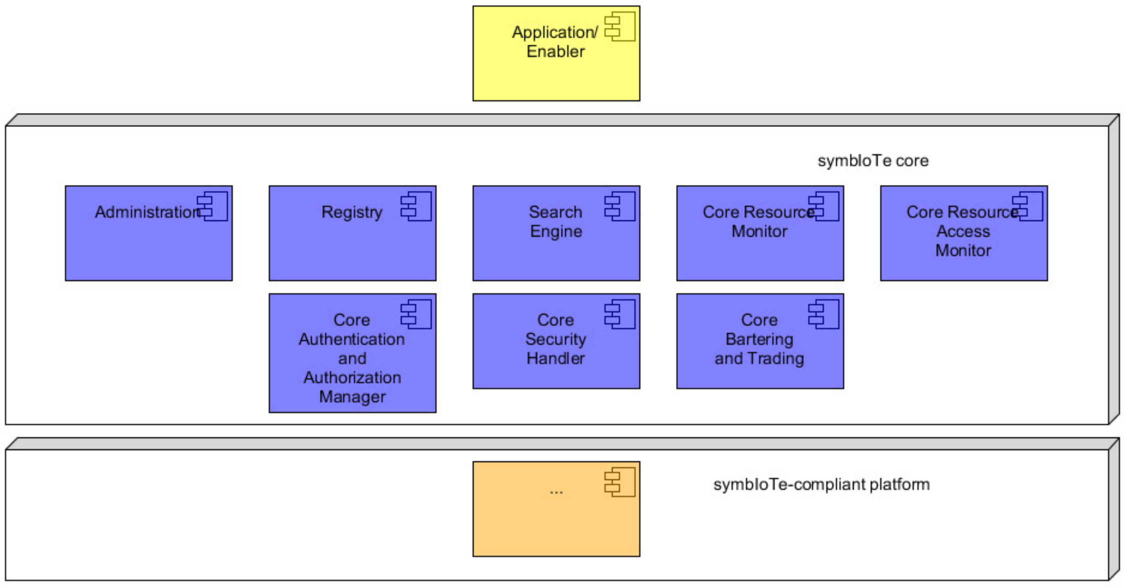
Application/ Enabler
symbIoTe core
Administration
Registry
Search Engine
Core Resource Monitor
Core Resource Access Monitor
Core Authentication and Authorization Manager
Core Security Handler
Core Bartering and Trading
...
symbIoTe-compliant platform

Figure 7 symbIoTe APP components

5.1.2 Component description

Hereafter we identify the core components for the Application Domain (symbIoTe Core Services). For some of the components, examples are provided showing how the component can be used.

Table 4 Administration

Component	Administration
Compliance Level	L1
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.
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.
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

Example 1: Platform registration

An instance of the OpenIoT platform running in Zagreb wants to register its resources to be searchable and accessible through symbIoTe. The required platform-specific registration data (platform-specific metadata) is as follows:

- Name: IoT-FER
- Type: OpenIoT
- Location: Croatia
- Resource Types: CO, NO_2, SO_2, T, H, P
- DataEndpoint: iot.fer.unizg.hr:12345/resources/
- Interworking API: iot.fer.unizg.hr

Note: All IoT platforms that will be symbIoTe-enabled should declare the metadata needed to register a platform instance with symbIoTe

Table 5 Registry

Component	**Registry**
Compliance Level	L1
symbIoTe Domain	APP
Description	This component must enable the registration of (composite) IoT services which are

	offered by IoT platforms to be discoverable through symbIoTe. In addition, the component should enable the registration 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. In addition, 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 IoT services, 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. A core information model with extensions must be supported by the Registry for the description of available resources across IoT platforms. The symbIoTe core information model should be compatible with existing standards, prominent ontologies in the IoT space and should be aligned (if possible) with the data model of other IoT-EPI projects. The model must support geo-referenced information. It must distinguish IoT devices which are fixed (their geo-location does not change over time) and mobile (their geo-location continuously changes). The information generated by IoT devices must be annotated by standard units. The encoding of units should adhere to a standard (e.g. UCUM). The information model of an IoT platforms and enablers registering their resources to symbIoTe should be aligned to the symbIoTe information model.
Provided functionalities	• Handles requests for resource registration and resource updates. • Assigns symbIoTe-specific unique identifiers to resources. • Stores resource metadata. All resources are described using the symbIoTe information model. • Receives resource availability information from the Core RM and updates resource status (e.g. online/offline).
Relation to other components	Search Engine: uses platform, enabler and resource-related information managed by the Registry. Core Resource Monitor: monitors availability of registered resources and pushes this information to the Registry.
Related use cases	All
Related requirements	1, 3, 11, 12, 15, 16, 18, 19, 25, 39, 45, S1, S2

Example 2: Resource registration

An air quality sensor managed by an OpenIoT instance registers with symbIoTe. The required data to register/update a resource is as follows:

- Location: [45.8 N; 16 E]
- Features: CO, NO_2, T, H, P
- Id: 2321-2312-abcd
- symbIoTe-platform-UId: OpenIoT-xxxx-yyyy-zzzz
- Description: CO is measured in mg/m^3, Temperature is measured in Celsius. Raw data obtained from sensor nodes is open access.

Note: All resources will be described using the symbIoTe information model.

Table 6 Search Engine

Component	**Search Engine**
Compliance Level	L1

symbIoTe Domain	APP
Description	This component enables application and enabler developers to search for resources available through symbIoTe Core Services using a web-based GUI. Search requests should specify metadata and relevance criteria for a search request, as well as application/enabler credentials. The component uses metadata associated to registered resources maintained by the Registry to calculate resource scores concerning 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 also be exposed and should 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. • Finds the available resources and provides "snippets" describing resources which include "symbIoTelized" URLs 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.
Relation to other components	Registry: provides platform, enabler and resource-related information needed to perform the search operation. Core Resource Access Monitor: serves as a proxy for URLs appearing in search results.
Related use case sections	All
Related requirements	2, 4, 19, 23, 26, 31, 33, S1, S2

Example 3: Search example
A Smart Mobility and Ecological Routing enabler needs to find available sensors that measure air quality in Zagreb. An example query could be defined as follows.

- Location: [45.8 N; 16 E]
- Features: CO, T

Note: We need to identify metadata relevant to search queries across different use cases.

Table 7 Core Resource Monitor

Component	**Core Resource Monitor** (Core RM)
Compliance Level	L1
symbIoTe Domain	APP
Description	This component must monitor the availability of registered resources to regularly update resource status (online/offline/unavailable) maintained by the Registry. 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. It uses 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). The component needs to provide appropriate credentials when checking resource status. In addition, it 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	• 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	Registry: manages resource-related monitor information. Search Engine: uses performance-related data obtained by the Core RM for creating and annotating search results.
Related use cases	All
Related requirements	5, 6, 7, 8, 32, S1, S2

Example 4: Core Resource Monitor
When an air quality sensor is registered with symbIoTe as shown in Example 1, the Core Resource Monitor tries to access the registered resource end point to check its status.

Table 8 Core Resource Access Monitor

Component	**Core Resource Access Monitor** (Core RAM)
Compliance level	L1
symbIoTe Domain	APP
Description	This component must monitor which resources are selected in search results to maintain resource popularity information within the Core Services. It acts as a proxy which redirects applications and enablers to the actual resources offered by symbIoTe-enabled platforms. It should keep track of the resources assigned to applications/enablers. Such information is vital to mark resources with exclusive rights as unavailable, enable prioritized access to resources, and may be useful to estimate resource load. The component must grant exclusive use of IoT resources, for underlying IoT platforms that support such feature. Thus, it will not redirect applications and enablers to resources if their availability the potentially associated QoS cannot be guaranteed by the underlying IoT platforms. This applies in particular to resource types with exclusive access rights (e.g., actuators) or computing/storage/network resources supporting the delivery of information generated by resources. It should 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. It should enable the control of access to the advertised IoT services for reasons related to local legislation, current overload, security issues, etc., if the underlying IoT platforms support it. It may be able to reserve registered resources for the future, for IoT platforms and enablers which support such feature.
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 • Grants exclusive access to a particular resource (if the platform/enabler supports such feature) • Enables resource reservation (if the platform/enabler supports such feature) • Enables prioritized access to a resource (if the platform/enabler supports such feature) • Enables control of access to the advertised resources according to local legislation, current overload, security issues etc.
Relation to other components	Resource Search: 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, 8, 9, 10, 13, 14, 27

Example 5: Core Resource Access Monitor
When the Smart Mobility application chooses the air quality sensor introduced in Example 2 from search results, this action will be noted by the Core RAM as (SmartMobilityUID, uses resource: iot.fer.unizg.hr:12345/2321-2312-abcd/). The application request is forwarded to resource endpoint iot.fer.unizg.hr:12345/2321-2312-abcd/.

Table 9 Core Authentication and Authorization Manager

Component	**Core Authentication and Authorization Manager** (Core AAM)
Compliance Level	L1
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 and symbIoTe-based applications, through the use of X.509 certificates. It must authenticate components belonging to a given IoT platform federated 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 federated with symbIoTe, that would like to search and access to resources. It must issue “core tokens” for components and applications that successfully complete the authentication procedure in the core layer. Core tokens contain attributes (i.e., roles, properties, permissions) that are used to access to resources and/or services, according to the Attribute Based Access Control (ABAC) paradigm. It must also revoke “core 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 • convert home tokens in core tokens 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) It may manage a Token Revocation List (TRL), that contains all the tokens that have been revoked by AAMs module in each IoT platform before the expiration date indicated in the token itself.
Provided functionalities	• Authenticates a user registered in symbIoTe core within symbIoTe. • Releases a set of tokens storing trusted attributes, if the authentication process was successful. • 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 the token in a new set of attributes which are valid within the core layer, if an entity that is not registered within the core layer is in possession of a valid token. This procedure is called "Attributes Mapping Function".
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 10 Core Security Handler

Component	**Core Security Handler** (Core SH)
Compliance Level	L1
symbIoTe Domain	APP
Description	This component provides a set of libraries implementing security functionalities. It must manage security material (i.e., usernames, passwords, certificates, received tokens etc.) of the component or the application that uses its functionalities. It must perform authentication procedures on behalf of the component or application that uses its functionalities. In this case, the security functionalities include: • Application authentication; • Managing and forwarding of home tokens to obtain core tokens or foreign tokens • Performing of cryptographic operations using the private key in a challenge-response procedure • Validation of X.509 certificates provided by foreign components. It must verify the authenticity of components or applications that try to access to the component that uses its functionalities. In this case, the security functionalities include: • Validation of core tokens, through the check of the expiration date and the sign of the token; • Initiation of the challenge-response procedure using the public key of the requesting entity, included in the token; • Checking of any asynchronous revocation of the token with the Core AAM • Comparison between the set of attributes included in a set of tokens and the access policy required to access a given resource. It must validate X.509 certificates when an entity belonging to an IoT platform federated with symbIoTe sends a message to a component in the core layer that uses its functionalities. This is necessary to authenticate that entity. It must check the matching between attributes stored in a set of tokens and the access policy associated to a given resource.
Provided functionalities	• Authenticates with the Core AAM or foreign AAM on behalf of the entity that uses its functionalities. • Manages core tokens and foreign tokens assigned to the entity that uses its functionalities. • Performs the "validate access tokens" procedure when one or more core tokens are provided to the entity that uses its functionalities. • Performs the "check revocation procedure" with the Core AAM when one or more core tokens are provided to the entity that uses its functionalities. • Initiates the "Challenge-Response Procedure" to verify that the component or application using the core tokens is effectively the component or application for which they have been released by the Core AAM, in case one or more core

	tokens are provided to the entity that uses its functionalities. • Manages cryptography operations on behalf of a component using its functionalities, when such component or application provides a set of tokens to a component in a given IoT platform federated with symbIoTe. • Performs the "Check Access Policy" procedure to verify that the tokens supplied by the applications satisfy the access policies of the resources when the Search Engine is used by an application. • Performs the "Validate Certificate" procedure on behalf of the component or application that uses its functionalities.
Relation to other components	Core AAM, Core Resource Monitor, Search Engine
Related use cases	All
Related requirements	14, 34, 35, 37, 40, 42, 43, 52, 59 and S1, S2, S3, S5, S7, S19, S20

Table 11 Core Bartering & Trading Component

Component	**Core Bartering & Trading Component** (Core BarT)
Compliance Level	L2
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 (needs confirmation)

Related requirements	46-53

5.2 *Cloud Domain*

5.2.1 General concepts

The Cloud Domain (CLD) components will enable IoT Platform Providers to register desired resources to symbIoTe Core Services. In addition, the components will enable applications/enablers to access those resources, found through the symbIoTe Core Services, in a unified and secure way. These functionalities are a prerequisite for achieving symbIoTe L1 Compliance. Furthermore, the components will implement specific functionality for the exchange of information between two collaborating IoT platforms, thus enabling IoT Platform Federation and symbIoTe L2 Compliance.

The components within CLD are shown in Figure 8. Each IoT Platform Provider will need to install a set of tools containing these components on the platform side in order to integrate their resources with symbIoTe. 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
- **Authentication and Authorization Manager (AAM):** enables a common authentication and authorization mechanism for all L2 Compliant Platforms and applications
- **Bartering and Trading Manager (BTM):** manages bartering and trading actions in advance to establishing federations
- **Monitoring:** monitors status of IoT Devices and records when applications/enablers access IoT Devices/Composite IoT services
- **Federation Manager (SLA):** offers SLA (Service Level Agreement) between IoT Platforms to create a federation, monitors if SLA is being respected
- **Security Handler (SH):** provides a set of libraries implementing security functionalities

The aforementioned tools will be an umbrella to the underlying platform specific components and translate the interaction using the symbIoTe core information model defined in WP2. The tools will provide a unified access to symbIoTe core and hide the heterogeneity from the IoT platform. The platforms might have their own monitoring system, their own SLA, etc. 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.

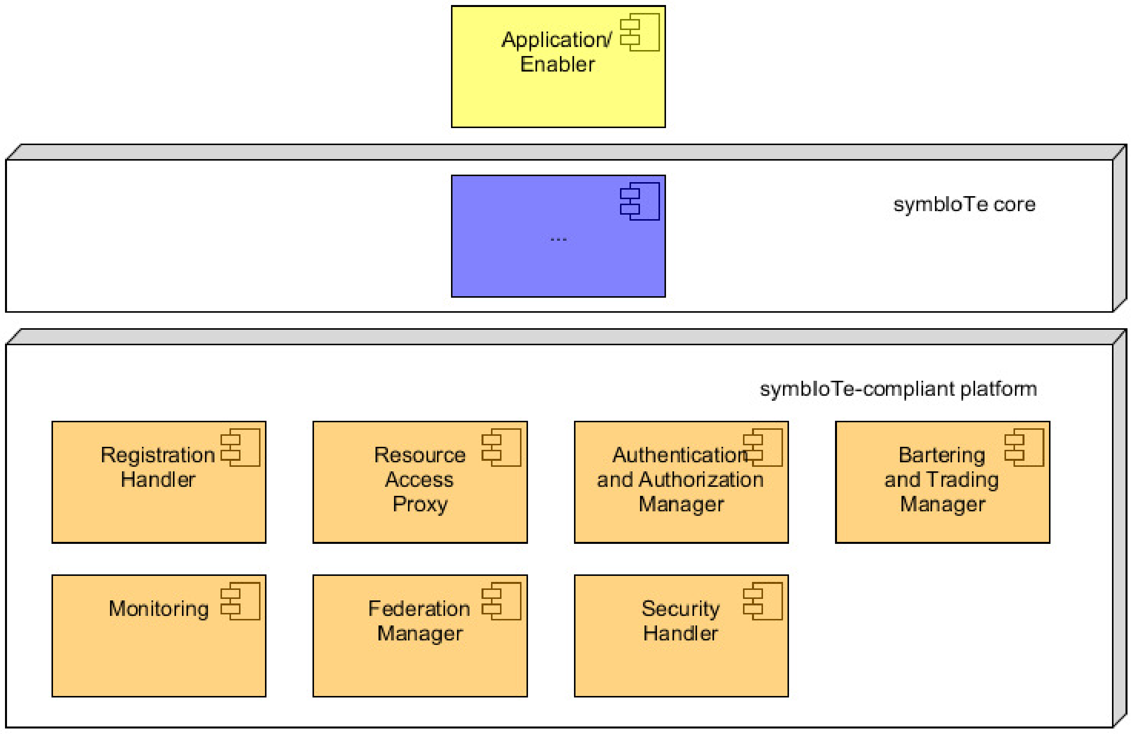

Figure 8 symbIoTe CLD components

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 that will have an intensive interaction with the 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 Cloud Domain 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 identify the components for Cloud Domain.

Table 12 Registration Handler

Component	**Registration Handler** (RH)
Compliance Level	L1, L2
symbIoTe Domain	CLD
Description	This component will drive an IoT Platform Provider through the step of registering resources into the symbIoTe Core. Registration Handler should monitor platform resources so that it can register them into the symbIoTe (eco)system. These resources can be an IoT Device or a Composite IoT Service and must be described using the symbIoTe information model. The Registration Handler component should map the platform information model to the symbIoTe information model. Some domain-specific properties of a resource can be reported during the registration process by using the platform information model, but they have to be formatted in a manner of an extension of the symbIoTe information model. Registration Handler needs to provide the following information from the symbIoTe

	information model: IoT Device or Composite IoT Service description, Location and its properties and Observed Property description and name. Registration Handler signs into the symbIoTe system through the Core Authentication & Authorization Manager with security credentials obtained during the platform registration process, so that the Registry can verify the origin of the registration request. Registry accepts the registration requests, and replies to the Registration Handler with a list of IDs generated by symbIoTe system for those resources. A similar process is occurring in the event of unregistration of a resource or update of the resource information. Both processes can be triggered by the Registration Handler. Additionally, the Registration Handler can invoke (force) resource availability updating and checking, by triggering the appropriate command in the Core Resource Monitor which eventually leads to the resource availability check and status update of a resource within the symbIoTe (eco)system. The IoT Platform Provider must be able to define which of its resources are available to use, and under which conditions. A platform does not have to expose all of its resources to symbIoTe. Instead, they may opt to reveal just a subset. An IoT Platform Provider might also set the rules for access to its resources (e.g. a maximum of 10 times per day, only from 7p.m. to 7a.m., only to administrators of other platforms etc.). The platform provider can set-up access rules and access scope in the Authentication & Authorization Manager. Rules regarding the bid and usage of a resource are configured through Bartering and Trading Manager, while set-up of SLA of a resource is done in cooperation with Federation Manager. The Registration Handler serves as mediator in communication of IoT Platform Provider with symbIoTe (and its specific components). All updates about resources and its features should also be handled by Registration Handler.
Provided functionalities	• Registers resources to the symbIoTe core, virtual and physical, using the symbIoTe information model • Updates resource status and unregistered resources • Registers SLA information of a resource • Registers pricing of a resource • Registers security information (access rules) • Handles configuration of exposition from resources/service • Synchronizes the information with symbIoTe core
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 (SLA): stores data regarding SLA of a resource 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, 61

Table 13 Resource Access Proxy

Component	**Resource Access Proxy** (RAP)
Compliance Level	L1, L2
symbIoTe Domain	CLD
Description	This component enables symbIoTe-compliant access to resources within an IoT platform or (enabler acting as a platform). 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. Also, if an SLA is involved, it should be verified. 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. Access to the resource will be provided when all previous conditions are satisfied. The component may check the type of resource which is being accesses (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 information model. 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 should be supported.
Provided functionalities	• Enables authorized access to platform/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
Relation to other components	Bartering and Trading Manager (L2) Authentication and Authorization Manager (AAM) (L1, L2) Federation Manager (SLA) (L2)
Related use cases	ALL
Related requirements	13, 14, 27, 38, 56

Table 14 Authentication and Authorization Manager

Component	**Authentication and Authorization Manager** (AAM)
Compliance Level	L1, L2
symbIoTe Domain	CLD
API	Interworking API
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
Relation to other components	Application/Enabler Registration Handler (L1, L2) Core Authentication & Authorization Manager (L1) Core Resource Access Monitor (L1) Resource Access Proxy (L1, L2) Federation Manager (L2) Monitoring (L1, L2)
Related use cases	All use cases
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 15 Bartering and Trading Manager

Component	**Bartering and Trading Manager** (BTM)
Compliance Level	L2
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 will be able to set up SLAs for its resources (including charges for using them and further rules to 'sell' them). It should be able to set different rules for the same resources depending on which platform is trying to access to the resource, kind of access (prioritized or not), time that is accessed (daytime, nighttime), 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 register the access done to the different resources from the platform and associate a price to this access. Therefore, it will need the information of who and from which platform is accessed. Bartering algorithm is applied when a user from another IoT platform tries to access a resource or just trading when the user is from an application. 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 (L1, L2) Registration Handler (L1, L2) 3rd part Bartering and Trading Manager (L2)

	Central Bartering and Trading component (L2)
Related use cases	ALL
Related requirements	41, 42, 46-53

Table 16 Monitoring

Component	**Monitoring**
Compliance Level	L1, L2
symbIoTe Domain	CLD
Description	This component is responsible for two different tasks: monitoring the status of resources and monitoring the access to resources from applications/enablers. 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 Resource Monitor at symbIoTe (eco)system. Regarding the access to resources, the component will record when applications/enablers use the IoT Devices/(Composite) IoT services. The Resource Access Proxy will emit a signal when the actual access is started and finished. This way it might be possible to check the response times from the resources.
Provided functionalities	• Resource monitoring • Record of start and end of the access to a resource
Relation to other components	Registration Handler (L1, L2) Resource Access Proxy. (L1, L2)
Related use cases	ALL
Related requirements	5, 6, 32

Table 17 Federation Manager

Component	**Federation Manager** (SLA)
Compliance Level	L2
symbIoTe Domain	CLD
Description	This component must be able to handle SLA offerings and SLA agreement. The SLA offering is a proposal done by the IoT Platform Provider. These IoT Platform Providers should be able to create generic offerings e.g: they don't need to specify which sensor will comply to which conditions; the IoT Platform Provider should be able to specify that a temperature sensor will comply to some specific condition like a response time lower than 10ms for example. Prior to the actual access to a resource, an agreement has to be created. The agreement will be between the IoT Platform Provider and an application/enabler/user. The agreement will contain the specific resources the application/enabler/user will access.

	When an agreement has been created, it must be monitored that no violations are taking place. If violations take place the Bartering and Trading Manager can be informed in order to offer a reward/punishment to the user/platform/application.
Provided functionalities	• SLA offering creation • SLA agreement creation when access is requested to the resources • SLA agreement monitoring
Relation to other components	Bartering and Trading Manager (L2) Monitoring (L1, L2)
Related use cases	ALL
Related requirements	33, 35

Table 18 Security Handler

Component	**Security Handler** (SH)
Compliance Level	L2
symbIoTe Domain	CLD
Description	This component provides a set of libraries implementing security functionalities. It MUST manage security material (i.e., usernames, passwords, certificates, received tokens etc.) of the component or the application that uses its functionalities. It MUST perform authentication procedures on behalf of the component and application that uses its functionalities. In this case, the security functionalities include: • Application authentication; • Managing and forwarding of home tokens to obtain core tokens or foreign tokens; • Performing of cryptography operations using the private key in a challenge-response procedure; • Validation of X.509 certificates provided by components in the core layer or in a foreign IoT platform. It MUST verify the authenticity of components or applications that try to access to the component that uses its functionalities. In this case, the security functionalities include: • Validation of home and foreign tokens, through the check of the expiration date and the sign of the token; • Initiation of the challenge-response procedure using the public key of the requesting entity, included in the token; • Checking of any asynchronous revocation of the token with the home and/or foreign AAM; • Comparison between the set of attributes included in a set of tokens and the access policy required to access a given resource. It MUST validate X.509 certificates when a component belonging to another IoT platform federated with symbIoTe or to symbIoTe core sends a message to the component that uses its functionalities. This is necessary to authenticate that entity. When a RAP uses its functionalities, it MUST check the matching between attributes stored in a set of tokens and the access policy associated to a given resource.
Provided functionalities	• The L2 Security Handler module authenticates with the home AAM, Core AAM or foreign AAM on behalf of the component or application that uses its functionalities. • The L2 Security Handler module manages home tokens, core tokens and foreign

	tokens assigned to the component or application that uses its functionalities. • When one or more home or foreign tokens are provided to the component that uses its functionalities, the L2 Security Handler module performs the “validate access tokens” procedure. • When one or more home or foreign tokens are provided to the component that uses its functionalities, the L2 Security Handler module performs the “check revocation procedure” with the Home AAM. • When one or more home or foreign tokens are provided to the component that uses its functionalities, the L2 Security Handler module initiate the “Challenge-Response Procedure” to verify that the component or application using the core tokens is effectively the component or application for which they have been released by the home AAM. • When a component or application using its functionalities provides a set of tokens to a component hosted in a given IoT platform federated with symbIoTe, the L2 Security Handler manages cryptography operations on behalf of this component using its functionalities. • When the RAP is used by an application, the L2 Security Handler module performs the “Check Access Policy” procedure to verify that the tokens supplied by the applications satisfy the access policies of the resources. • The L2 Security Handler module performs the “Validate Certificate” procedure on behalf of the component or application that uses its functionalities.
Relation to other components	Registration Handler (L1, L2) Resource Access Proxy (L1, L2) Authentication and Authorization Manager (L1, L2) Federation Manager (L2)
Related use cases	ALL
Related requirements	35, 37, 40, 42, 43, 53, 60, 61 and S1, S2, S3, S5, S6, S7, S8, S9, S17, S18, S19, S20

5.3 Smart Space Domain and Smart Device Domain

5.3.1 Vision

Smart Spaces are environments (residence, campus, vessel, …) 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 a split local / 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 designing a software architecture that can be deployed in either way with no significant modifications. The functionality is duplicated at various domains, in Cloud Domain and Smart Space Domain (e.g. device management in the CLD and SSP): what differs is the scope and the available hardware.

More specifically, the goal is:

- to keep the same architecture for software components breakdown, interfaces and communications paradigm

- to maximize code reuse (e.g. by creating libraries)
- to reuse entire software components, if possible
- to keep components as modular as possible, to be able to just plug platform specific code for each specific IoT platform.

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 SS will also be exposed directly, that is without being "mediated" by any of the local platforms. symbIoTe Compliant local IoT platforms within a SS will be able to access all the resources associated to that SS, provided that the required AA policies are in place. This includes both resources provided by any co-located 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 SS level. When more than one IoT platform is active in a SS, the S3 Middleware shall thus be acting as a local resource interchange.

5.3.2 Smart Space and visiting entities

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.

5.3.3 Smart Device

In the context of symbIoTe, a “Smart Device” (SD) is a device that can directly interact with a Smart Space:

- “directly” means that the device should not require any external components to communicate with the symbIoTe Smart Space. Third-party devices which are not symbIoTe-enabled can be considered an SD, if taken as a whole with the additional components can interact with a SS. (e.g.a commercial IP controlled electric plug coupled with a custom symbIoTe gateway is an SD);
- “interact” refers to a bidirectional exchange of information with the Smart Space, more specifically implementing the sequence of actions described in the following paragraphs.

According to this definition, any mobile device (smartphone, tablet) running a proper symbIoTe app can be considered an SD. The containing SS shall allow the SD to be accessed by components (apps, enablers) outside the SS itself, and by components inside it (an app associated to the same SS): in the latter case, no Internet connectivity shall be required. SymbIoTe Compliant local IoT platforms shall also be able to access an SD residing in the same SS, without the assistance of any cloud component.

5.3.4 L3 vs. L4 Compliance

As regards L3 vs. L4 Compliance, the distinguishing features of the two levels are dynamic configuration (L3) and roaming (L4). Hence, an SD will be considered L3 Compliant if it can be attached to a Smart Space and be dynamically configured (i.e.

registered to a symbIoTe Compliant platform and thus become searchable and accessible). The SD will be considered L4 Compliant if it can also move across different Smart Spaces and still be recognized as the same object it was before. In other words:

- device *identity* is the main characterizing aspect of an L4 SD: if the device can be uniquely recognized independently of the Smart Space it is connected to, it is actually roaming across platforms. Otherwise, there is no way to distinguish said device from another identical device;
- maintaining identity only makes sense if there is a service (symbIoTe enabler) or application which needs to track the device, either because it is associated with a specific service or user, or because the history of the device is meaningful for the purpose it is used for.

The only strict requirement for an SD to be L4 Compliant, thus, is having a unique identifier (e.g. MAC address or symbIoTe-generated UUID). The choice of using the SD as an L3 or L4 device, though, is up to the specific service and may even change over time, as the SD is being used for different purposes. Hence, an L4 Compliant SD does not necessarily always roam.

An L3 SD (identified by a given UUID, accessed through a given URL) will be accessed via a different URL each time it connects to a new Smart Space, whereas an L4 SD will always keep the same URL.

5.3.5 General architecture

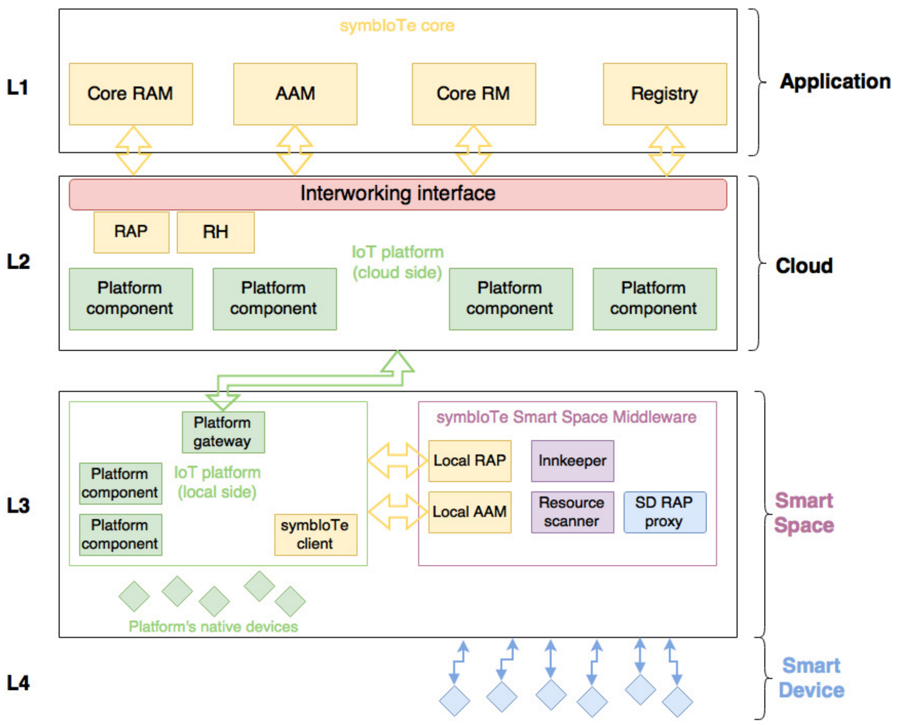

Figure 9 Architecture of Smart Space and Smart Device Domains

5.3.6 Components for L3/L4 Compliance

- **Innkeeper**: it is used to connect app or Smart Device to the Smart Space.
  - It notifies a new app that it has entered a symbIoTe Compliant Smart Space.

    - It registers a new device to the local SS.
    - It keeps a register of locally registered apps and Smart Devices.
    - It maps global URIs to local URIs.
- **Resource scanner**: it has the task of scanning the local network for well-known devices and device gateways.
- **Local RAP**: it 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.
- **Local AAM**: it authenticates and authorize symbIoTe applications and Smart Devices to allow access the Smart Space when no Internet connectivity is available.
- **SD RAP proxy**: it provides a resource access point for SDs to be addressable from outside the Smart Space and it provides a local RAP to access SDs from within the SS.
- **SymbIoTe client**: 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.

The Resource Scanner is in charge of scanning the local network for well-known devices and device gateways. Once a new (not already registered) device/gateway is detected, an association procedure is started and the device is attached to the symbIoTe middleware. So the Innkeeper (if possible) queries the symbIoTe core to know if the device shall be registered as a new device (L3 behavior) or as a roaming device (L4 behavior), and then registers it with the Core Registration Handler. The Innkeeper also provides the SD with the URI for the SD RAP Proxy, to which the SD connects immediately afterwards. If the Smart Space could be accessed only locally, the new Smart Devices or the new app can authenticate and obtain the authorization from the Local AMM.

When an app or a Smart Device is associated to the Smart Space, it can access to the Smart Space's resources through Local Rap, without passing through the cloud. There is also the possibility that an application outside the Smart Space may want to access the Smart Device: the SD RAP proxy is the module which is in charge to provide a resource access point for that Smart Devices.

To be able to access to Smart Devices registered in the same Smart Space of an IoT-platform, a symbiote client can be installed locally into the platform itself, becoming a client to access to that foreign resources.

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 interconnect with applications, while devices are managed by different IoT platforms. Attribute based access control (ABAC) fulfills 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 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 entity in the system. An ‘attribute’ is defined as a particular property, role or permission associated to an entity 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 through Access Control Policies. An access policy defining a specific combination of attributes needed to grant access to resources is assigned to each resource by the producer of that resource. Therefore, a client application may be granted access to a resource only if it possesses a set of attributes that match the predefined access policy. In symbIoTe this policy can contain at the same time attributes assigned to users and objects and also particular environment conditions connected to the 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 the system, with an enhanced flexibility especially in distributed environments, where the specific domains can avoid any form of synchronization to create consistent access control policies.

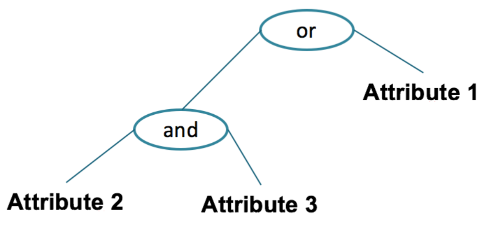

Figure 10 An example of access policy enforced by three attributes

For instance, with reference to the access policy depicted in the figure below, an application may access to the 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 symbIoTe, we have that:

- attributes are stored in token(s) and are generated by the AAMs in the Home Platform;
- a token ensures the authenticity of attributes it contains;
- attributes could be modified when accessing a foreign IoT platform, according to the 'attribute mapping function' implemented by the AAM;
- the "access policy checking" between attributes of the consumer and access policy established by the producer is handled by the RAP.

Further details related to symbIoTe approach to security will be provided in deliverable D3.1 “Basic Resource Trading Mechanisms and Access Scopes” due in M11.

5.5 Achieving Level 1 Compliance

As already stated in Section 3.3, L1 Compliance 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 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 will be analyzed in D2.1. 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 Compliant Platforms is presented in Section 5.5.1. It also identifies component interfaces. Sequence diagrams specifying L1 functionalities are included in 5.5.2, while Section 5.5.3 correlates messages exchanged between components with defined interfaces.

5.5.1 Component diagram

Component diagram with specified interfaces is shown in Figure 11. For Level 1 Compliance, 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
- **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)

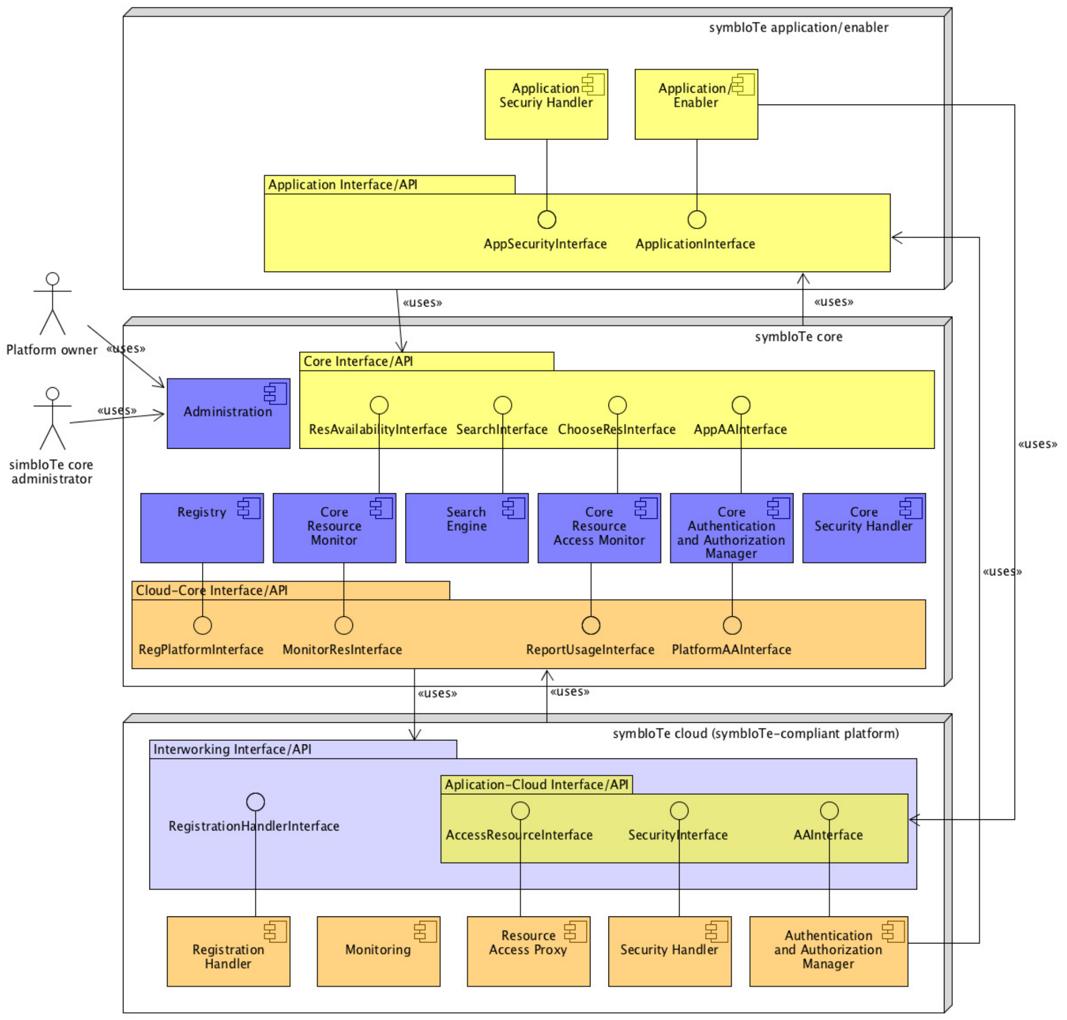

Figure 11 Component diagram for Level 1 Compliance

5.5.2 Sequence diagrams

The functionalities defined for symbIoTe Level 1 Compliance are the following:

- Platform registration
- Resource registration
- Resource unregistration
- Resource update
- Monitoring resource availability
- Search
- Access to resources (with and without registration)
- Monitoring

Hereafter all functionalities are presented in the form of UML sequence diagrams, with detailed description of the exchanged messages. Figure 12 shows the legend for messages used in the diagrams.

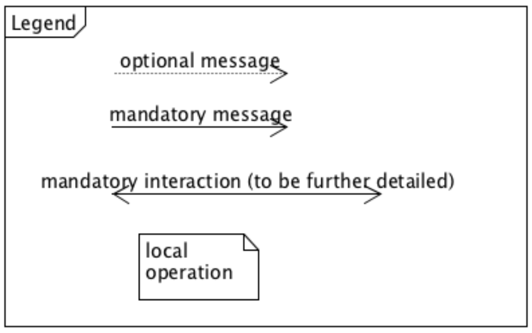

Figure 12 Legend – messages used in the following diagrams

5.5.2.1 Platform registration

Platform registration is executed by a symbIoTe core administrator or IoT platform provider by using the administrative web application.
An enabler has two roles in the registration process:

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

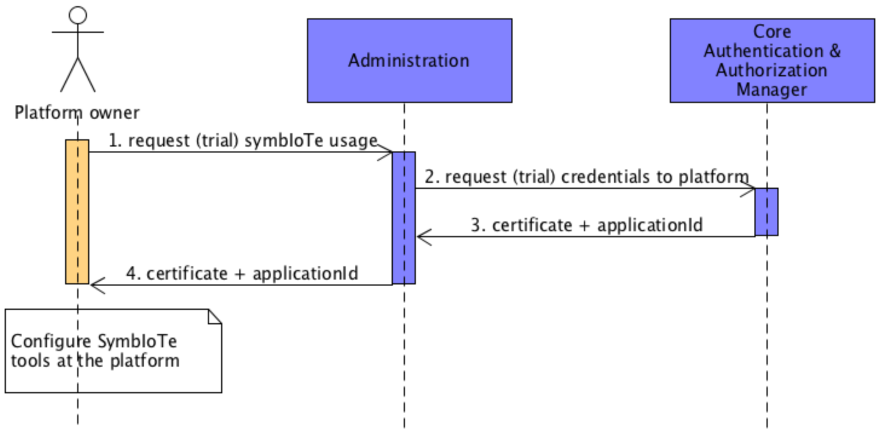
Platform owner
Administration
Core Authentication & Authorization Manager
1. request (trial) symbIoTe usage
2. request (trial) credentials to platform
3. certificate + applicationId
4. certificate + applicationId
Configure SymbIoTe tools at the platform

Figure 13 Platform registration

Description:

- Message 1: IoT Platform Provider sends a request for symbIoTe usage by using the Administration web application. The request is either for a trail or normal registration.
- Message 2: Administration sends request to the Core Authentication and Authorization Manager which requests credentials for the platform.

- Message 3: Core Authentication and Authorization Manager returns the generated certificate and applicationId to Administration.
- Message 4: Administration web application returns certificate and applicationId to IoT Platform Provider. The Platform Provider can subsequently configure the platform to become L1 Compliant.

5.5.2.2 Resource registration

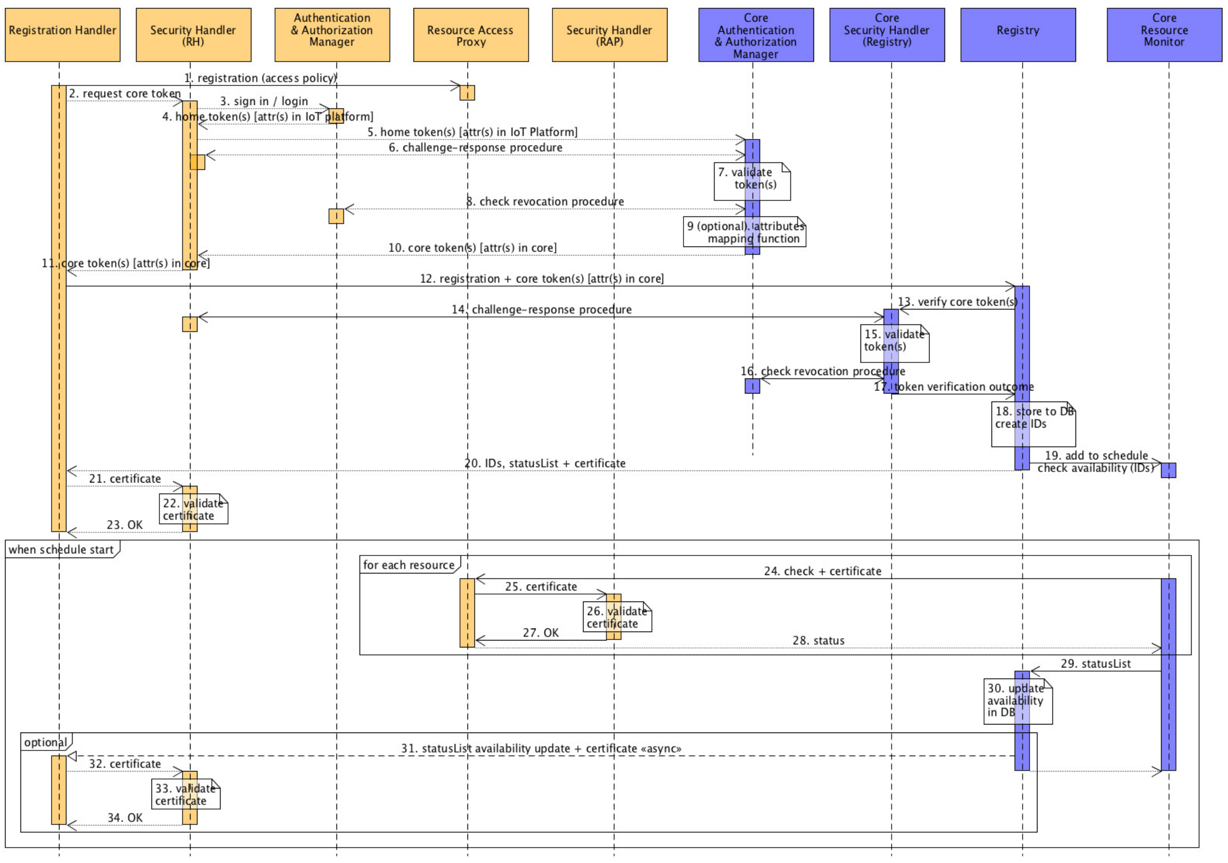
Registration Handler
Security Handler (RH)
Authentication & Authorization Manager
Resource Access Proxy
Security Handler (RAP)
Core Authentication & Authorization Manager
Core Security Handler (Registry)
Registry
Core Resource Monitor
1. registration (access policy)
2. request core token
3. sign in / login
4. home token(s) [attr(s) in IoT platform]
5. home token(s) [attr(s) in IoT Platform]
6. challenge-response procedure
7. validate token(s)
8. check revocation procedure
9 (optional). attributes mapping function
10. core token(s) [attr(s) in core]
11. core token(s) [attr(s) in core]
12. registration + core token(s) [attr(s) in core]
13. verify core token(s)
14. challenge-response procedure
15. validate token(s)
16. check revocation procedure
17. token verification outcome
18. store to DB create IDs
19. add to schedule check availability (IDs)
20. IDs, statusList + certificate
21. certificate
22. validate certificate
23. OK
when schedule start
for each resource
24. check + certificate
25. certificate
26. validate certificate
27. OK
28. status
29. statusList
30. update availability in DB
optional
31. statusList availability update + certificate «async»
32. certificate
33. validate certificate
34. OK

Figure 14 Resource registration

Description:

- Message 1: generated by the Registration Handler and sent to the Resource Access Proxy in the same IoT platform. It is used to register the resource on the Resource Access Proxy, along with the access policy to access it;
- Message 2 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, this step is not needed.
- Message 3 (optional): generated by the Security Handler and sent to the home (platform) AAM in which the Registration Handler is registered. It is used to

authenticate the Registration Handler. If the Registration Handler is already logged in, this step is not needed.

- Message 4 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 5 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 6 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 7 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8(optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 9(optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 10(optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 11(optional): generated by the Security Handler and sent to the Registration Handler. It is used to forward the core token generated at the previous step.
- Message 12 (RegPlatformInterface): generated by the Registration Handler and sent to the Registry. It's main purpose is to provide the metadata describing a resource or a set of resources which the platform exposes to the Registry. In addition to the registration message, it also provides the core token(s) containing the attributes assigned to the Registration Handler.
- Message 13: generated by the Registry and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 14 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).

- Procedure 15: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 16: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 17: generated by the Security Handler in the core layer and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 18: stores registrations to database and generates ID for that resource
- Message 19: Registry send message to Core Resource Monitor to add to the schedule task for checking availability of registered resources. Core Resource Monitor will in the future check availability (messages 24-28) and asynchronously inform Resource Handler about availability with updated status list (message 31).
- Message 20: Registry returns: IDs of registered resources, status list and certificate (used to demonstrate the identity of the entity generating the message, for authentication purposes, the certificate must be validated by the Security Handler of the component)
- Message 21: Registration Handler forwards certificate to Security Handler for validation
- Procedure 22: Security Handler validates certificate
- Message 23: Security Handler returns status of validation
- Messages 24-28 is checking of availability of each resource
- Message 24 (AccessResourceInterface): Core Resource Monitor sends message to Resource Access Proxy in order to check availability. It includes certificate as well (used to demonstrate the identity of the entity generating the message, for authentication purposes, the certificate must be validated by the Security Handler of the component).
- Message 25: Resource Access Proxy sends certificate to Security Handler for validation
- Procedure 26: Security Handler validates certificate
- Message 27: Security Handler returns status of validation
- Message 28: Resource Access Proxy returns availability status
- Message 29: Core Resource Monitor collects all availability statuses, make a status list and send it to Registry
- Procedure 30: Updates availability in database
- Message 31 (optional): Registry sends asynchronous message with availability list and certificate to Registration Handler

- Message 32 (triggered by 31) (RegistrationHandlerInterface): Registration Handler forwards certificate to Security Handler for validation
- Procedure 33: Security Handler validates certificate
- Message 34: Security Handler returns status of validation

5.5.2.3 *Resource unregistration*

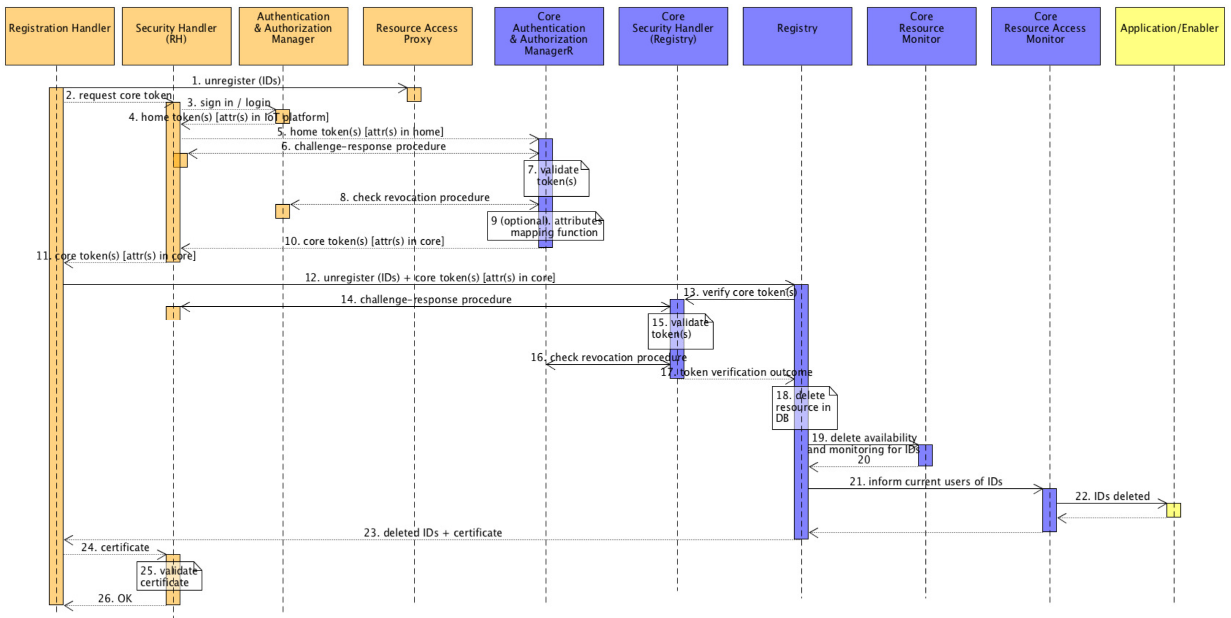
Registration Handler
Security Handler (RH)
Authentication & Authorization Manager
Resource Access Proxy
Core Authentication & Authorization ManagerR
Core Security Handler (Registry)
Registry
Core Resource Monitor
Core Resource Access Monitor
Application/Enabler
1. unregister (IDs)
2. request core token
3. sign in / login
4. home token(s) [attr(s) in IoT platform]
5. home token(s) [attr(s) in home]
6. challenge-response procedure
7. validate token(s)
8. check revocation procedure
9 (optional). attributes mapping function
10. core token(s) [attr(s) in core]
11. core token(s) [attr(s) in core]
12. unregister (IDs) + core token(s) [attr(s) in core]
13. verify core token(s)
14. challenge-response procedure
15. validate token(s)
16. check revocation procedure
17. token verification outcome
18. delete resource in DB
19. delete availability and monitoring for IDs
20
21. inform current users of IDs
22. IDs deleted
23. deleted IDs + certificate
24. certificate
25. validate certificate
26. OK

Figure 15 Resource unregistration

Description:

- Message 1: generated by the Registration Handler and sent to the Resource Access Proxy in the same IoT platform. It is used to unregister the resource on the Resource Access Proxy;
- Message 2 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.
- Message 3 (optional): generated by the Security Handler and sent to the home AAM in which the Registration Handler is registered. It is used to authenticate the Registration Handler. If the Registration Handler is already logged in, it is not necessary.
- Message 4 (optional): generated by the home AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 5 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.

- Procedure 6 (optional) (Security Interface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 7 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 9 (optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 10(optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 11 (optional): generated by the Security Handler and sent to the Registration Handler. It is used to forward the core token generated at the previous step.
- Message 12 (RegPlatformInterface): generated by the Registration Handler and sent to the Registry. It is used to provide, along with the unregistration message, the core token(s) containing the attributes assigned to the Registration Handler.
- Message 13: generated by the Registry and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 14 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).
- Procedure 15: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 16: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 17: generated by the Core Security Handler and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 18: Registry deletes resource in database

- Message 19: Registry sends message to Core Resource Monitor to delete availability and possibly scheduled task that check availability of specified resource
- Message 20: returns call
- Message 21: Registry informs Core Resource Access Monitor that specific source is unregistered and that the users of that resource needs to be informed
- Message 22 (optional) (ApplicationInterface): Core Resource Access Monitor informs each reachable (open connection or registered endpoint) Application/Enabler that uses specific resource about deletion of resource
- Message 23: Registry returns deleted IDs and certificate to Registration Handler (certificate is used to demonstrate the identity of the entity generating the message, for authentication purposes, certificate must be validated by the Security Handler of the component)
- Message 24: Registration Handler forwards certificate to Security Handler for validation
- Procedure 25: Security Handler validates certificate
- Message 26: Security Handler returns status of validation

5.5.2.4 *Resource update*

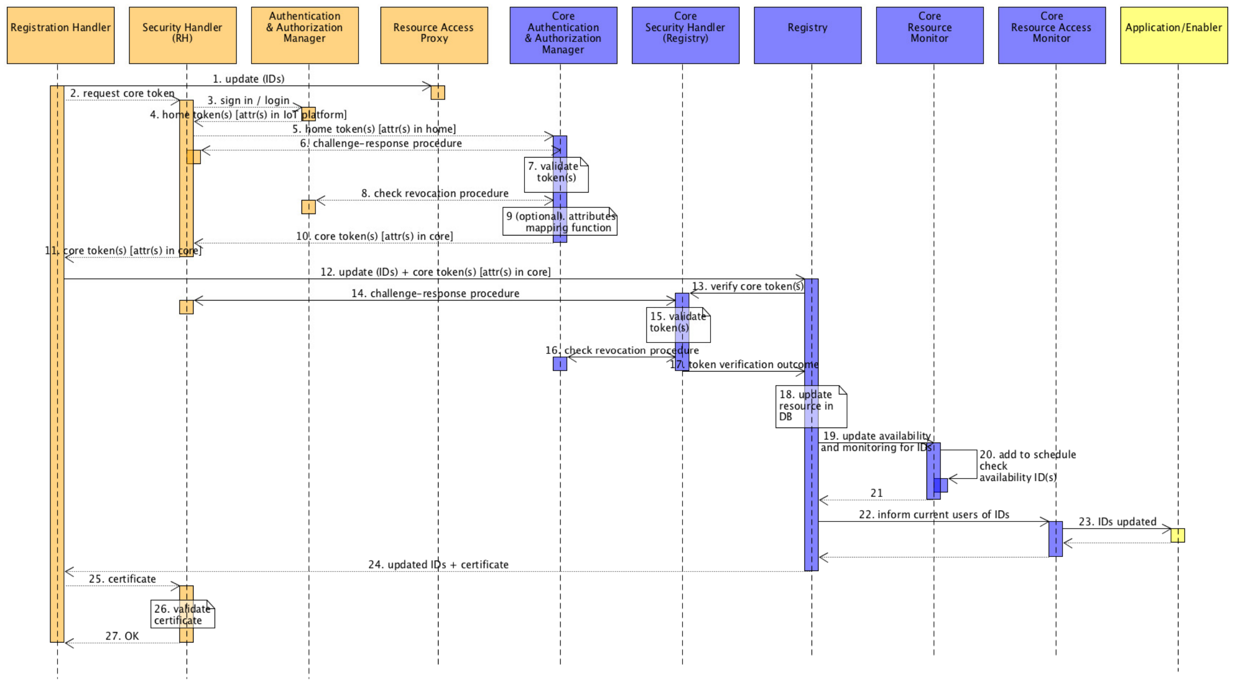
Registration Handler
Security Handler (RH)
Authentication & Authorization Manager
Resource Access Proxy
Core Authentication & Authorization Manager
Core Security Handler (Registry)
Registry
Core Resource Monitor
Core Resource Access Monitor
Application/Enabler
1. update (IDs)
2. request core token
3. sign in / login
4. home token(s) [attr(s) in IoT platform]
5. home token(s) [attr(s) in home]
6. challenge-response procedure
7. validate token(s)
8. check revocation procedure
9 (optional). attributes mapping function
10. core token(s) [attr(s) in core]
11. core token(s) [attr(s) in core]
12. update (IDs) + core token(s) [attr(s) in core]
13. verify core token(s)
14. challenge-response procedure
15. validate token(s)
16. check revocation procedure
17. token verification outcome
18. update resource in DB
19. update availability and monitoring for IDs
20. add to schedule check availability ID(s)
21
22. inform current users of IDs
23. IDs updated
24. updated IDs + certificate
25. certificate
26. validate certificate
27. OK

Figure 16 Resource update

Description:

- Message 1: generated by the Registration Handler and sent to the Resource Access Proxy in the same IoT platform. It is used to update the resource on the Resource Access Proxy;

- Message 2 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.
- Message 3 (optional): generated by the Security Handler and sent to the home AAM in which the Registration Handler is registered. It is used to authenticate the Registration Handler. If the Registration Handler is already logged in, it is not necessary.
- Message 4 (optional): generated by the home AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 5 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 6 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 7 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 9 (optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 10 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 11 (optional): generated by the Security Handler and sent to the Registration Handler. It is used to forward the core token generated at the previous step.
- Message 12 (RegPlatformInterface): generated by the Registration Handler and sent to the Registry. It is used to provide, along with the update message, the core token(s) containing the attributes assigned to the Registration Handler.
- Message 13: generated by the Registry and sent to the Security Handler in the core layer. It is used to ask to the security handler to verify the complete validity of the token.

- Procedure 14 (SecurityInterface):_procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).
- Procedure 15:_verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 16: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 17: generated by the Security Handler in the core layer and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 18: Registry updates resource in database
- Message 19: Registry ends request to Core Resource Monitor to update availability and to schedule availability check
- Message 20: Core Resource Monitor schedules task for checking availability for specified resources
- Message 21: returns call
- Message 22: Registry sends message to Core Access Resource Access Monitor to inform current user of updated resources
- Message 23 (optional) (ApplicationInterface): Core Resource Access Monitor informs each reachable (open connection or registered endpoint) Application/Enabler that uses specific resource about resource update
- Message 24: Registry returns updated IDs including certificate
- Message 25: Registration Handler forwards certificate to Security Handler for validation
- Procedure 26: Security Handler validates certificate
- Message 27: Security Handler returns status of validation

5.5.2.5 Monitoring resource availability

Monitoring can be started by Application (yellow part of sequence diagram - messages 1-12) or by Registration Handler (orange part of sequence diagram - messages 13-33). The result is scheduled task for checking availability. Blue part of sequence diagram shows what happened when scheduled task starts executing (messages 34-41).

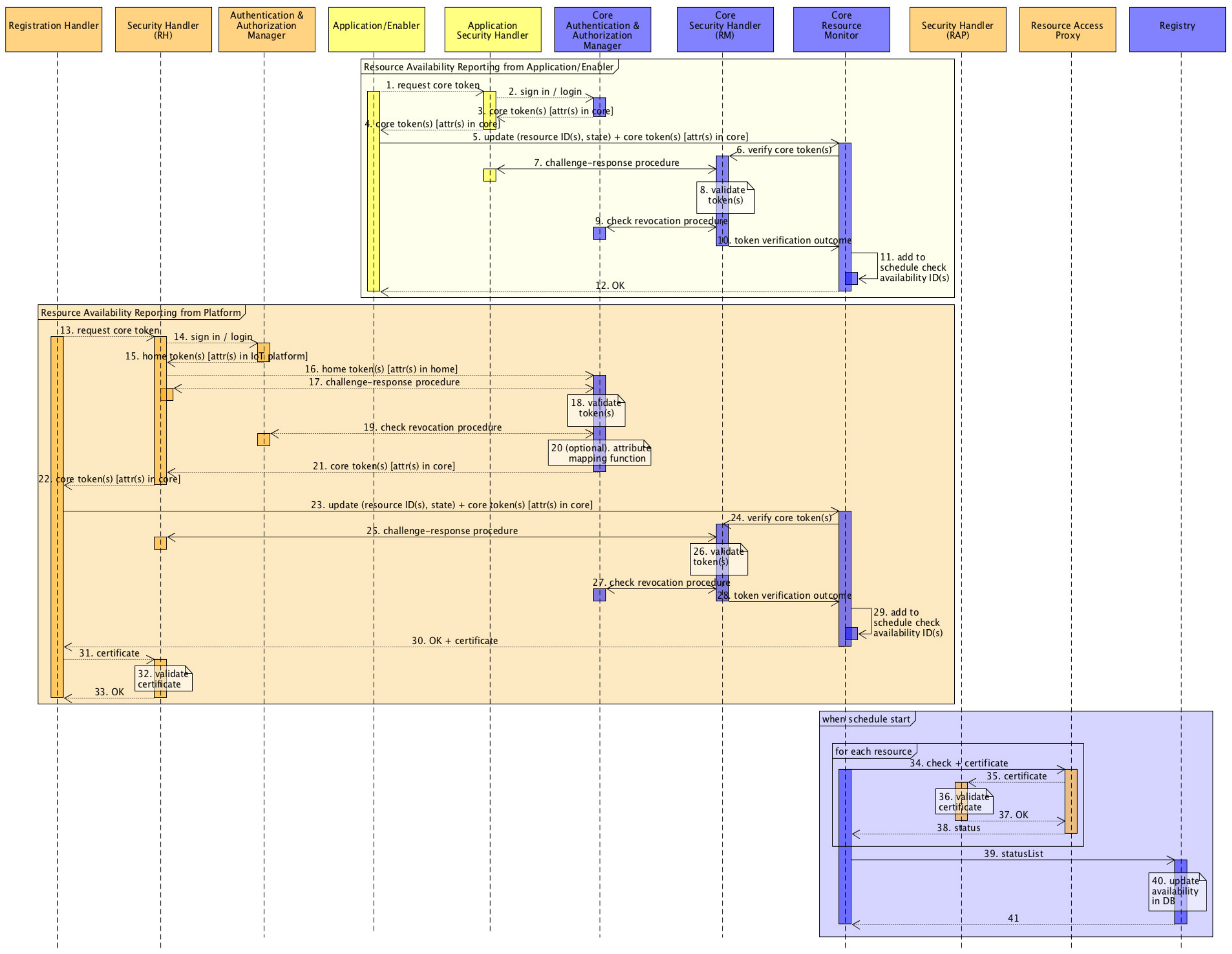
Registration Handler
Security Handler (RH)
Authentication & Authorization Manager
Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Security Handler (RM)
Core Resource Monitor
Security Handler (RAP)
Resource Access Proxy
Registry
Resource Availability Reporting from Application/Enabler
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
5. update (resource ID(s), state) + core token(s) [attr(s) in core]
6. verify core token(s)
7. challenge-response procedure
8. validate token(s)
9. check revocation procedure
10. token verification outcome
11. add to schedule check availability ID(s)
12. OK
Resource Availability Reporting from Platform
13. request core token
14. sign in / login
15. home token(s) [attr(s) in IoT platform]
16. home token(s) [attr(s) in home]
17. challenge-response procedure
18. validate token(s)
19. check revocation procedure
20 (optional). attribute mapping function
21. core token(s) [attr(s) in core]
22. core token(s) [attr(s) in core]
23. update (resource ID(s), state) + core token(s) [attr(s) in core]
24. verify core token(s)
25. challenge-response procedure
26. validate token(s)
27. check revocation procedure
28. token verification outcome
29. add to schedule check availability ID(s)
30. OK + certificate
31. certificate
32. validate certificate
33. OK
when schedule start
for each resource
34. check + certificate
35. certificate
36. validate certificate
37. OK
38. status
39. statusList
40. update availability in DB
41

Figure 17 Monitoring resource availability

Description:

- Message 1 (optional): generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.
- Message 2 (optional) (AppAAInterface): generated by the Application Security Handler and sent to the Core (home) AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.
- Message 3 (optional): generated by the Core (home) AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.
- Message 4 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It it is used to deliver the core token(s).
- Message 5 (ResAvailabilityInterface): generated by the Application/Enabler and sent to the Core Resource Monitor. It is used to forward the update message and the core token(s) to the Core Resource Monitor.

- Message 6: generated by the Core Resource Monitor and sent to the Core Security Handler in the core layer. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 7 (AppSecurityInterface): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).
- Procedure 8: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 9: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 10: generated by the Core Security Handler in the core layer and sent to the Core Resource Monitor. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 11: Core Resource Monitor schedules task for checking availability of specifies IDs
- Message 12: returns status of scheduling task for checking availability
- Message 13 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.
- Message 14 (optional): generated by the Security Handler and sent to the home AAM in which the Registration Handler is registered. It is used to authenticate the Registration Handler. If the Registration Handler is already logged in, it is not necessary.
- Message 15 (optional): generated by the home AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 16 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 17 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 18 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 19 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the

expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.

- Procedure 20 (optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 21 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Messages 22: returns core token to Registration Handler
- Message 23 (MonitorResInterface): generated by the Registration Handler and sent to the Core Resource Monitor. It is used to provide, along with the update message, the core token(s) containing the attributes assigned to the Registration Handler.
- Message 24: generated by the Core Resource Monitor and sent to the Core Security Handler. It is used to ask the security handler to verify the complete validity of the token.
- Procedure 25 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).
- Procedure 26: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 27: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 28: generated by the Core Security Handler and sent to the Core Resource Monitor. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 29: schedules task for checking availability of specified resources (IDs)
- Message 30: returns status of availability scheduling and certificate (used to demonstrate the identity of the entity generating the message, for authentication purposes, the certificate must be validated by the Security Handler of the component)
- Message 31: Registration Handler sends certificate to Security Handler for validation
- Procedure 32: validate certificate
- Message 33: returns result of certificate validation

- Message 34 (AccessResourceInterface): when the availability checking task is executed it starts with this message from Core Resource Monitor to Resource Access Proxy (includes certificate)
- Message 35: Resource Access Proxy sends certificate for validation to Security Handler
- Procedure 36: validates certificate
- Message 37: returns result of certificate validation
- Message 38: Resource Access Proxy returns result of availability to Core Resource Monitor
- Message 39: Core Resource Monitor collects all availability results, creates status list and send it to Registry
- Procedure 40: updates availability in database
- Message 41: returns call

5.5.2.6 Search

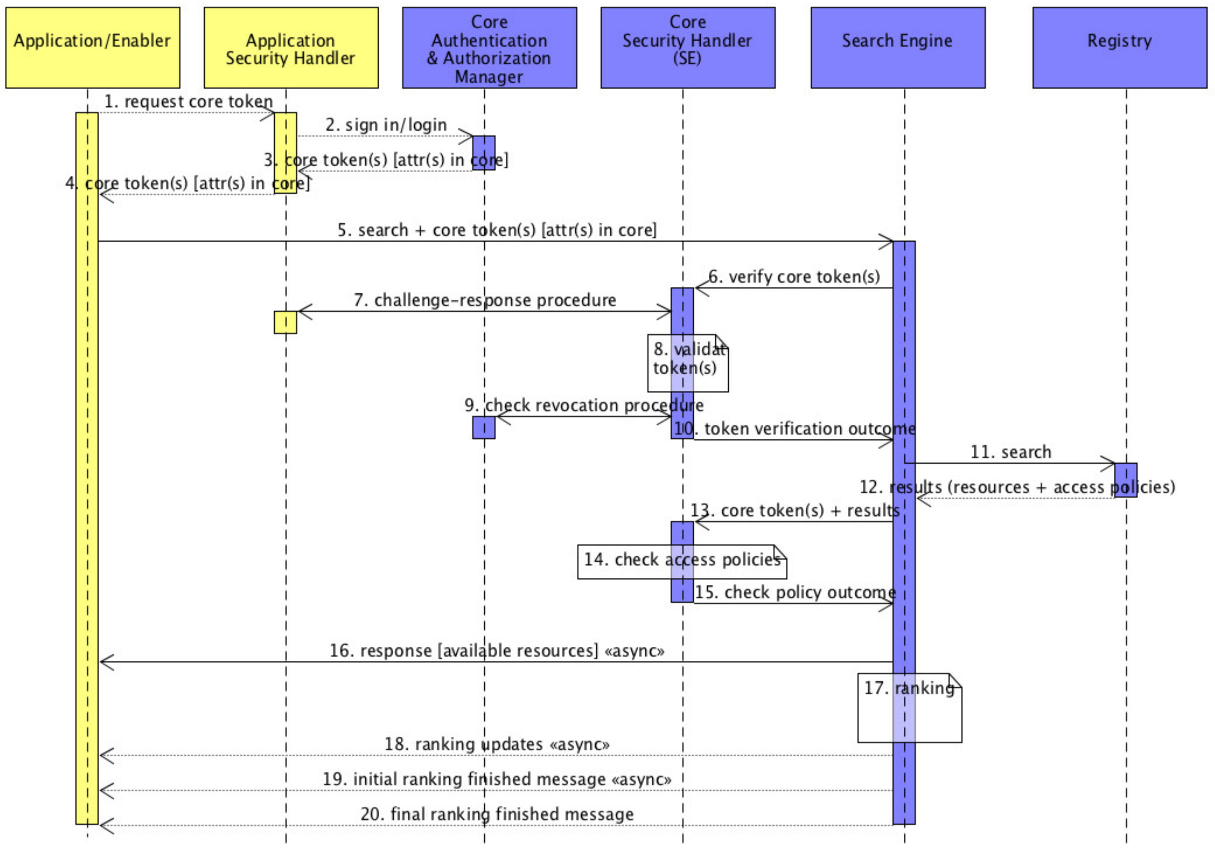
Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Security Handler (SE)
Search Engine
Registry
1. request core token
2. sign in/login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
5. search + core token(s) [attr(s) in core]
6. verify core token(s)
7. challenge-response procedure
8. validat token(s)
9. check revocation procedure
10. token verification outcome
11. search
12. results (resources + access policies)
13. core token(s) + results
14. check access policies
15. check policy outcome
16. response [available resources] «async»
17. ranking
18. ranking updates «async»
19. initial ranking finished message «async»
20. final ranking finished message

Figure 18 Search

Description:

- Message 1 (optional): generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.
- Message 2 (optional) (AppAAInterface): generated by the Application Security Handler and sent to the Core AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.
- Message 3 (optional): generated by the Core AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.
- Message 4 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to deliver the core token(s).
- Message 5 (SearchInterface): generated by the Application/Enabler and sent to the Search Engine. It sends search query and the core token(s) to the Search Engine.
- Message 6: generated by the Search Engine and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 7 (AppSecurityInterface): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).
- Procedure 8: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 9: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 10: generated by the Core Security Handler and sent to the Search Enginge. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 11: generated by the Search Engine and sent to the Registry. It is used to search available resources.
- Message 12: generated by the Registry and sent to the Search Engine. It is used to return the result of the search operation, containing resources and associated access policies.
- Message 13: generated by the Search Engine and sent to the Core Security Handler. It is used to deliver the core token(s) previously verified and the results of the search operation.

- Procedure 14: procedure that checks, for each resource, if the attributes contained in the core token(s) satisfy the access policy associated to that resource.
- Message 15: generated by the Core Security Handler and sent to the Search Engine. It is used to deliver the result of the previous procedure.
- Message 16: generated by the Search Engine and sent to the Application/Enabler asynchronously. It is used to deliver the result of the search operation (available resources).
- Procedure 17: executes ranking of food resources
- Message 18 (ApplicationInterfce): asynchronously sends ranking update to Application/Enabler
- Message 19 (ApplicationInterfce): asynchronously sends message about the end of initial ranking
- Message 20 (ApplicationInterfce): synchronously sends message of the end of final ranking

5.5.2.7 Access to resources

Access to resources is defined for two possible cases. In the first case, shown in Figure 19, Application or Enabler attempts to access resources without reservation, while in the second case, shown in Figure 20, Application or Enabler attempts to access resources with reservation.

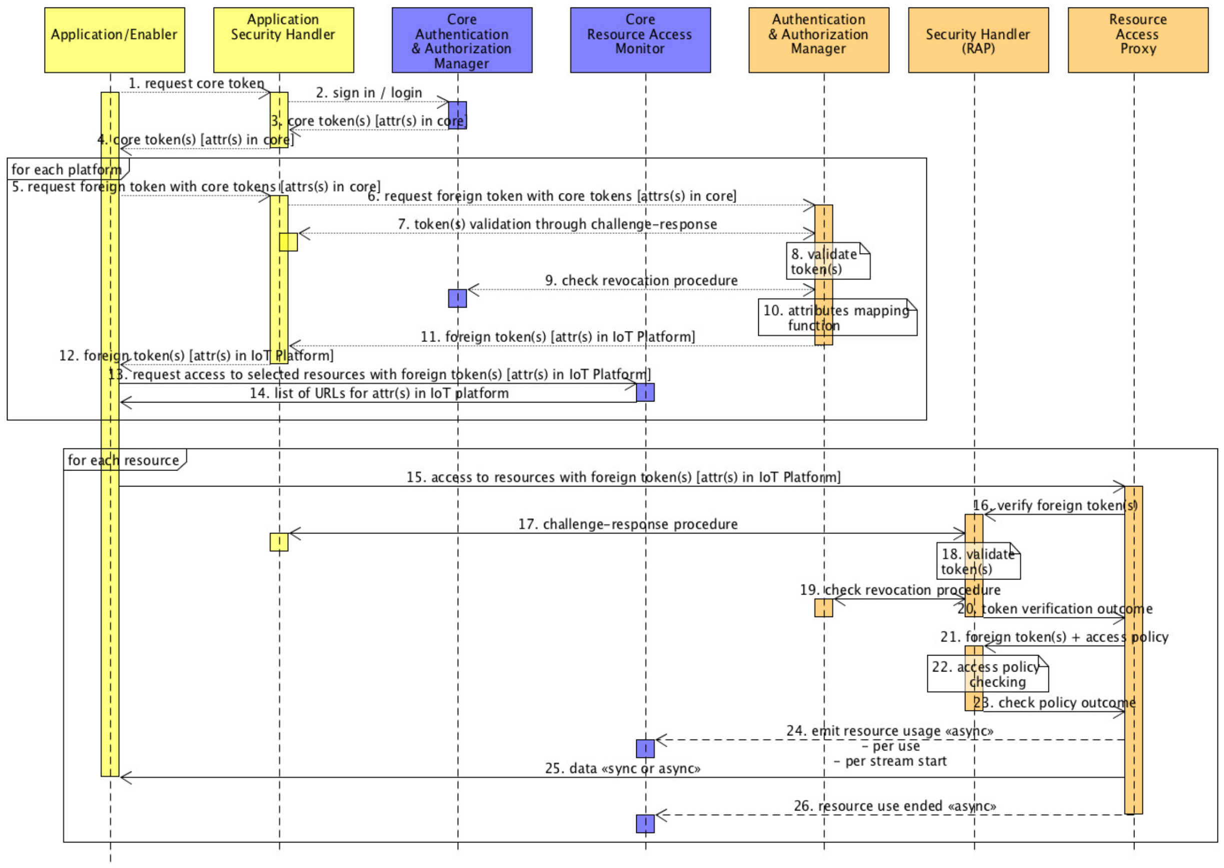
Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Resource Access Monitor
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. request foreign token with core tokens [attrs(s) in core]
6. request foreign token with core tokens [attrs(s) in core]
7. token(s) validation through challenge–response
8. validate token(s)
9. check revocation procedure
10. attributes mapping function
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
13. request access to selected resources with foreign token(s) [attr(s) in IoT Platform]
14. list of URLs for attr(s) in IoT platform
for each resource
15. access to resources with foreign token(s) [attr(s) in IoT Platform]
16. verify foreign token(s)
17. challenge–response procedure
18. validate token(s)
19. check revocation procedure
20. token verification outcome
21. foreign token(s) + access policy
22. access policy checking
23. check policy outcome
24. emit resource usage «async»
– per use
– per stream start
25. data «sync or async»
26. resource use ended «async»

Figure 19 Access to resources without reservation

Description:

- Message 1 (optional): generated by the Application/Enabler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.
- Message 2 (optional) (AppAAInterface): generated by the Security Handler and sent to the home AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.
- Message 3 (optional): generated by the home AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.
- Message 4(optional): generated by the Security Handler and sent to the Application/Enabler. It it is used to deliver the core token(s).
- Message 5 (optional): generated by the Application/Enabler and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform. If the Application/Enabler already has valid foreign token(s), it is not necessary.

- Message 6 (optional) (AAInterface): generated by the Application Security Handler and sent to the foreign AAM in IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 7 (optional) (AppSecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 8 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 9 (optional) (PlatformAAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 10 (optional): procedure that, in case it is needed, translates attributes that the Application/Enabler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Application/Enabler already has valid foreign token(s), it is not necessary.
- Message 11 (optional): generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Message 12 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to forward the foreign token generated at the previous step.
- Message 13 (ChooseResInterface): Application/Enabler sends request access to selected resources to Core Resource Access Monitor. Message includes foreign token obtained in previous message
- Message 14: Core Recoure Access Monitor returns list of URLs for selected resources in IoT platform
- Message 15 (AccessResourceInterface): generated by the Application/Enabler and sent to the Resource Access Proxy in the foreign IoT platform. It is used to access resources, while providing the foreign token previously obtained.
- Message 16: generated by the Resource Access Proxy and sent to the Security Handler in the foreign IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 17 (AppSecurityInterface): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).
- Procedure 18: verification of the time validity, authenticity and integrity of the provided token(s).

- Procedure 19: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself).
- Message 20: generated by the Security Handler in the foreign IoT platform and sent to the Resource Access Proxy. It is used to communicate the outcome of the token validation procedures performed by the Foreign Security Handler.
- Message 21: generated by the Resource Access Proxy and sent to the Security Handler. It is used to deliver the core token(s) previously verified and the access policy of the requested resource to the Security Handler.
- Procedure 22: it is used to check if the attributes included in the core token(s) satisfy the access policy associated to the requested resource.
- Message 23: generated by the Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- Message 24: (ReportUsageInterface) asynchronously emit resource usage per use/per stream start
- Message 25: (ApplicationInterface) this message can be synchronous, then Resource Access Proxy returns data. If it is asynchronously then it can emit async messages for some time
- Message 26: (ReportUsageInterface) if previous message is asynchronous then this message informs Core Resource Access Monitor when the stream is ended

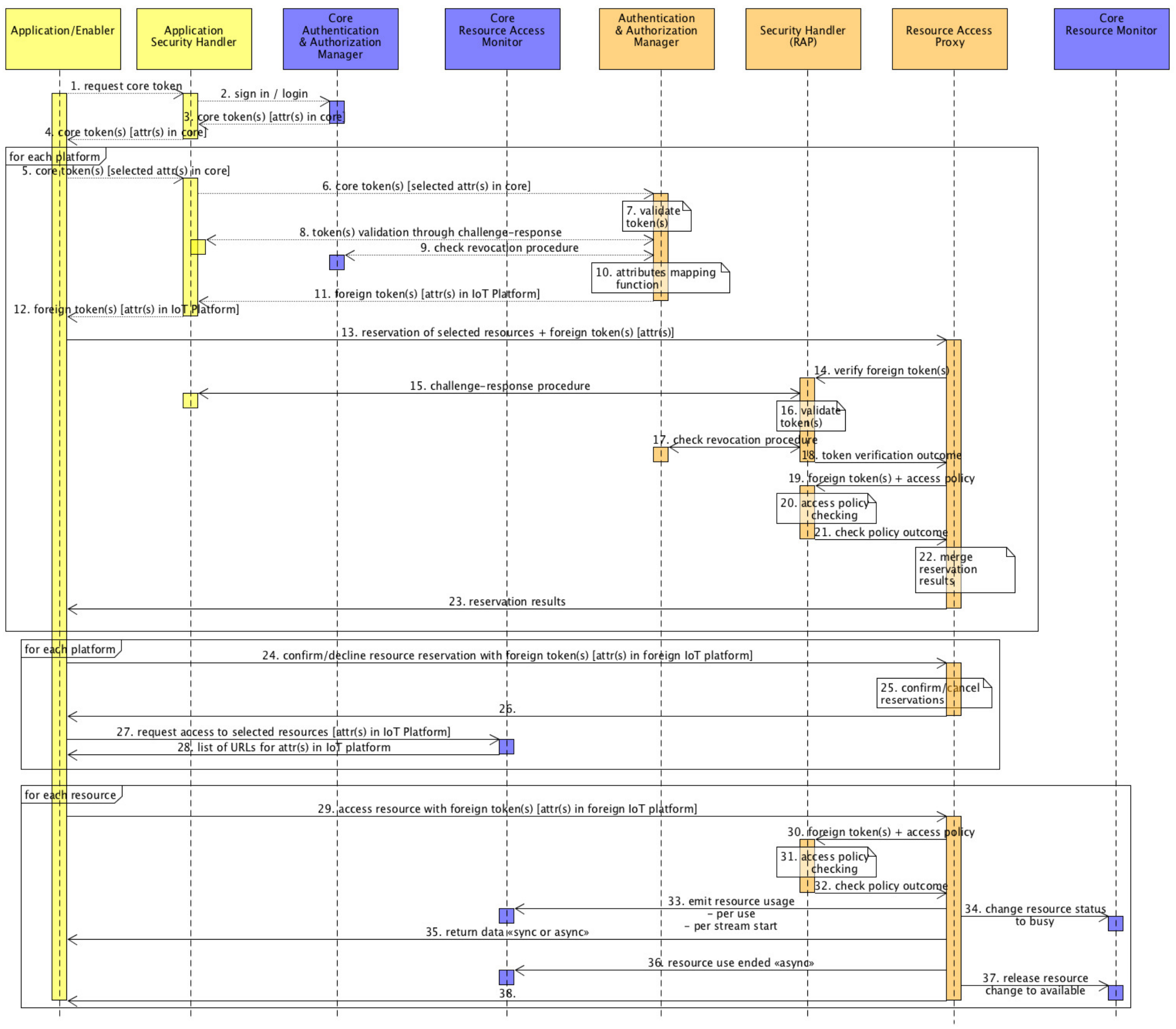
Application/Enabler
Application Security Handler
Core Authentication & Authorization Manager
Core Resource Access Monitor
Authentication & Authorization Manager
Security Handler (RAP)
Resource Access Proxy
Core Resource Monitor
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
for each platform
5. core token(s) [selected attr(s) in core]
6. core token(s) [selected attr(s) in core]
7. validate token(s)
8. token(s) validation through challenge-response
9. check revocation procedure
10. attributes mapping function
11. foreign token(s) [attr(s) in IoT Platform]
12. foreign token(s) [attr(s) in IoT Platform]
13. reservation of selected resources + foreign token(s) [attr(s)]
14. verify foreign token(s)
15. challenge-response procedure
16. validate token(s)
17. check revocation procedure
18. token verification outcome
19. foreign token(s) + access policy
20. access policy checking
21. check policy outcome
22. merge reservation results
23. reservation results
for each platform
24. confirm/decline resource reservation with foreign token(s) [attr(s) in foreign IoT platform]
25. confirm/cancel reservations
26.
27. request access to selected resources [attr(s) in IoT Platform]
28. list of URLs for attr(s) in IoT platform
for each resource
29. access resource with foreign token(s) [attr(s) in foreign IoT platform]
30. foreign token(s) + access policy
31. access policy checking
32. check policy outcome
33. emit resource usage
- per use
- per stream start
34. change resource status to busy
35. return data «sync or async»
36. resource use ended «async»
37. release resource change to available
38.

Figure 20 Access to resources with reservation

Description:

- Message 1 (optional): generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the recovery of the core token(s). If the Application/Enabler is already logged in, it is not necessary.
- Message 2 (optional) (AppAAInterface): generated by the Application Security Handler and sent to the Core AAM in which the Application/Enabler is registered. It is used to authenticate the Application/Enabler. If the Application/Enabler is already logged in, it is not necessary.
- Message 3 (optional): generated by the Core AAM in the IoT platform and sent to the Application Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Enabler is already logged in, it is not necessary.
- Message 4 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It it is used to deliver the core token(s).

- Message 5 (optional): generated by the Application/Enabler and sent to the Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Message 6 (optional) (AAInterface): generated by the Application Security Handler and sent to the foreign AAM in the foreign IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 7 (optional): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 8 (optional) (AppSecurityInterface): verification of the time validity, authenticity and integrity of the provided token(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 9 (optional) (PlatformAAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Procedure 10 (optional): procedure that, in case it is needed, translates attributes that the Application/Enabler has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Application/Enabler already has valid foreign token(s), it is not necessary.
- Message 11 (optional): generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Enabler already has valid foreign token(s), it is not necessary.
- Message 12 (optional): generated by the Application Security Handler and sent to the Application/Enabler. It is used to forward the foreign token generated at the previous step.
- Message 13 (Access Resource Interface): generated by the Application/Enabler and sent to the Resoruce Access Proxy in the foreign IoT platform. It is used to reserve resources, while providing the foreign token previously obtained.
- Message 14: generated by the Resource Access Proxy and sent to the Security Handler in the foreign IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 15 (AppSecurityInterface): procedure that allows the Application Security Handler that is acting on behalf of the Application/Enabler to demonstrate that it is the real owner of the token(s).
- Procedure 16: verification of the time validity, authenticity and integrity of the provided token(s).

- Procedure 17: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the AAM before the expiration time indicated within the token itself).
- Message 18: generated by the Security Handler in the foreign IoT platform and sent to the Resource Access Proxy. It is used to communicate the outcome of the token validation procedures performed by the foreign Security Handler.
- Message 19: generated by the Resource Access Proxy and sent to the Security Handler. It is used to deliver the core token(s) previously verified and the access policy of the requested resources to the Security Handler.
- Procedure 20: it is used to check if the attributes included in the core token(s) satisfy the access policy associated to the requested resources.
- Message 21: generated by the Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- Message 22: merge reservation results
- Message 23: Resource Access Proxy returns reservation results to Application/Enabler
- Message 24 (AccessResourceInterface): Application/Enabler sends confirm/decline resource reservation message to Resource Access Proxy. The message contains foreign token(s) and attributes.
- Procedure 25: Resource Access Proxy confirms or cancels reservation in the platform
- Message 26: return call
- Message 27 (ChooseResInterface): Application/Enabler sends request access to selected resources to Core Resource Access Monitor
- Message 28: Core Recoure Access Monitor returns list of URLs for selected resources in IoT platform
- Message 29 (AccessResourceInterface): generated by the Application/Enabler and sent to the Resource Access Proxy in the foreign IoT platform. It is used to access resources, while providing the foreign token previously obtained.
- Message 30: generated by the Resource Access Proxy and sent to the Security Handler in the foreign IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 31: it is used to check if the attributes included in the core token(s) satisfy the access policy associated to the requested resource.
- Message 32: generated by the Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- Message 33 (ReportUsageInterface): emit resource usage per use/per stream start

- Message 34 (MonitorResInterface): Resource Access Proxy informs Core Resource Monitor that the resource is busy
- Message 35 (ApplicationInterface): this message can be synchronous, then Resource Access Proxy returns data. If it is asynchronously then it can emit async messages for some time
- Message 36 (ReportUsageInterface): if previous message is asynchronous then this message informs Core Resource Access Monitor when the stream is ended
- Message 37 (MonitorResInterface): Resource Access Proxy informs Core Resource Monitor that the resource is released and status can be changed to available
- Message 38: returns call

5.5.2.8 Monitoring

This process is triggered by the platform at suitable times.

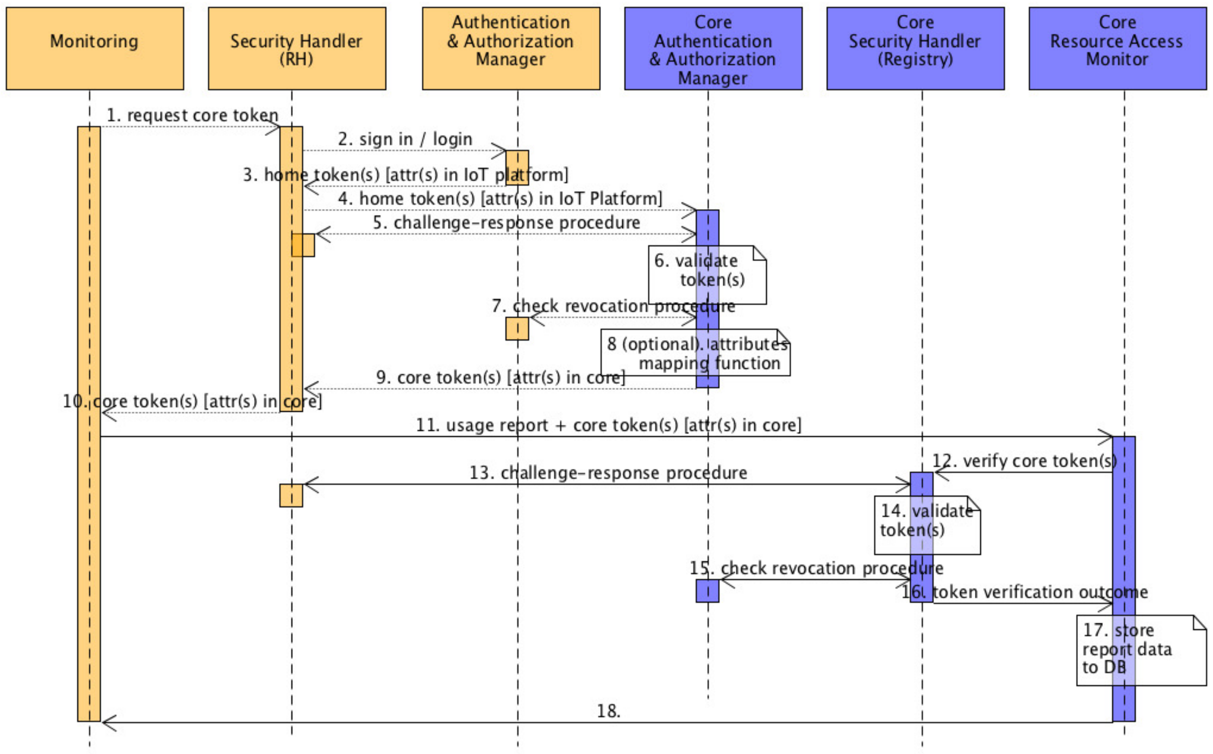
Monitoring
Security Handler (RH)
Authentication & Authorization Manager
Core Authentication & Authorization Manager
Core Security Handler (Registry)
Core Resource Access Monitor
1. request core token
2. sign in / login
3. home token(s) [attr(s) in IoT platform]
4. home token(s) [attr(s) in IoT Platform]
5. challenge-response procedure
6. validate token(s)
7. check revocation procedure
8 (optional). attributes mapping function
9. core token(s) [attr(s) in core]
10. core token(s) [attr(s) in core]
11. usage report + core token(s) [attr(s) in core]
12. verify core token(s)
13. challenge-response procedure
14. validate token(s)
15. check revocation procedure
16. token verification outcome
17. store report data to DB
18.

Figure 21 Monitoring

Description:

- Message 1 (optional): generated by the Monitoring and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Monitoring is already logged in, it is not necessary.

- Message 2 (optional): generated by the Security Handler and sent to the home (platform) AAM in which the Monitoring is registered. It is used to authenticate the Monitoring. If the Monitoring is already logged in, it is not necessary.
- Message 3 (optional): generated by the home (platform) AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Monitoring is already logged in, it is not necessary.
- Message 4 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 5 (optional) (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Monitoring to demonstrate that it is the real owner of the token(s). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 6 (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 7 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 8 (optional): procedure that, in case it is needed, translates attributes that the Monitoring has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Monitoring already has valid core token(s), it is not necessary.
- Message 9 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Monitoring already has valid core token(s), it is not necessary.
- Message 10(optional): generated by the Security Handler and sent to the Monitoring. It is used to forward the core token generated at the previous step.
- Message 11 (ReportUsageInterface): Monitoring generates usage report and sent it to the Core Resource Access Monitoring.
- Message 12: generated by the Core Resource Access Monitoring and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 13 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Monitoring to demonstrate that it is the real owner of the token(s).
- Procedure 14: verification of the time validity, authenticity and integrity of the provided token(s).

- Procedure 15: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 16: generated by the Security Handler in the core layer and sent to the Core Resource Access Monitoring. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 17: stores report data to database
- Message 18: returns call

5.5.3 Interfaces

Interfaces have been defined for the components of the Application and Cloud Domain enabling symbIoTe Level 1 Compliance, as it can be seen in Figure 11. Table 19 shows those interfaces, components within which they are implemented, as well as messages with input and output data exchanged when using those interfaces.

Table 19 Interfaces

Component	Interface	Message	Input	Output
Application/Enabler	ApplicationInterfce	ID deleted	ID	
		ID updated	ID	
		ranking update	ranking update, query id	
		initial ranking finished message	query id	
		final ranking finished message	query id	
		data	resource ID, data	
Application Security Handler	AppSecurityInterface	challenge-response procedure	security challenge data	security response data
Registry	RegPlatformInterface	registration	registration resource data, core token	resource IDs, resource status list, certificate
		unregister	resource IDs, core token	deleted resource IDs, certificate
		update	resource IDs, core token	updated resource IDs, certificate
Core Resource Monitor	ResAvailabilityInterface	update	resources IDs, state, core token, attributes in core	
	MonitorResInterface	update	resources IDs, state, core token, attributes in core	OK, certificate

Search Engine	SearchInterface	search	query, core token, attributes in core	available resources, query id
Core Resource Access Monitor	ChooseResInterface	update	resource ID, state, core token, attributes in core	
		request access	resource IDs, foreign token, attributes in IoT platform	list of access URLs in IoT platform
	ReportUsageInterface	resource usage	resource IDs, usage status	
		resource usage ended	resource IDs	
		usage report	report, core token, attributes in core	
Core Authentication and Authorization Manager	AppAAInterface	sign in application/enabler	username, password	core token, attributes in core
	PlatformAAInterface	request core token	home token, attributes in home	core token, attributes in core
		check revocation	core token	status
Registration Handler	RegistrationHandlerInterface	availability update	status list, certificate	
Resource Access Proxy	AccessResourceInterface	check availability	resource ID, certificate	availability status
		access resources	resource ID, foreign token, attributes in IoT platform	resource data
		reservation	resource IDs, foreign token, attributes in IoT platform	reservation status
		confirm/decline resource reservation	resource IDs with confirm/decline status, foreign token, attributes in IoT platform	
Security Handler	SecurityInterface	challenge-response procedure	security challenge data	security response data
Authentication and Authorization Manager	AAInterface	check revocation	home token	status
		request foreign token	core token, attributes in core	foreign token, attributes in IoT platform

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) to the symbIoTe architecture. 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 in 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 (Section 10.1), while hereafter we concentrate on mapping of the AIOTI layers to the symbIoTe architecture.

Figure 22 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.

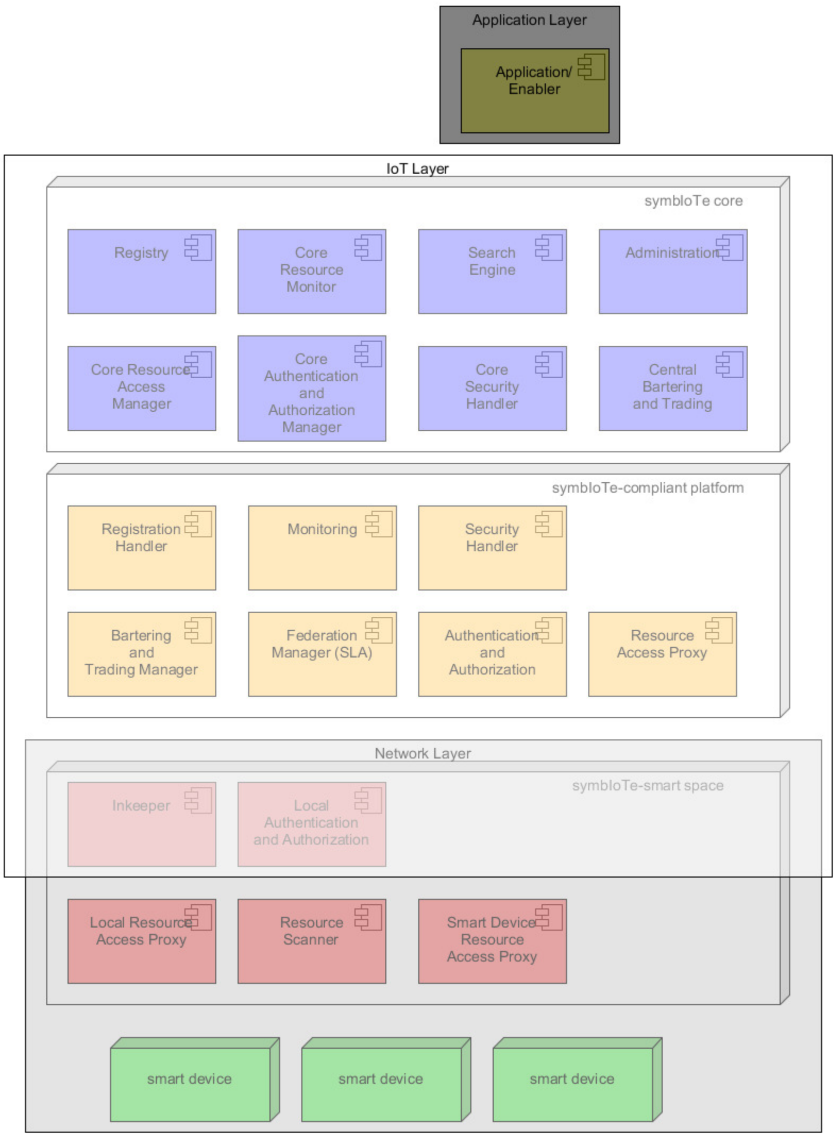

Figure 22 Mapping between the AIOTI HLA and symbIoTe architecture

Figure 23 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 23. 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.

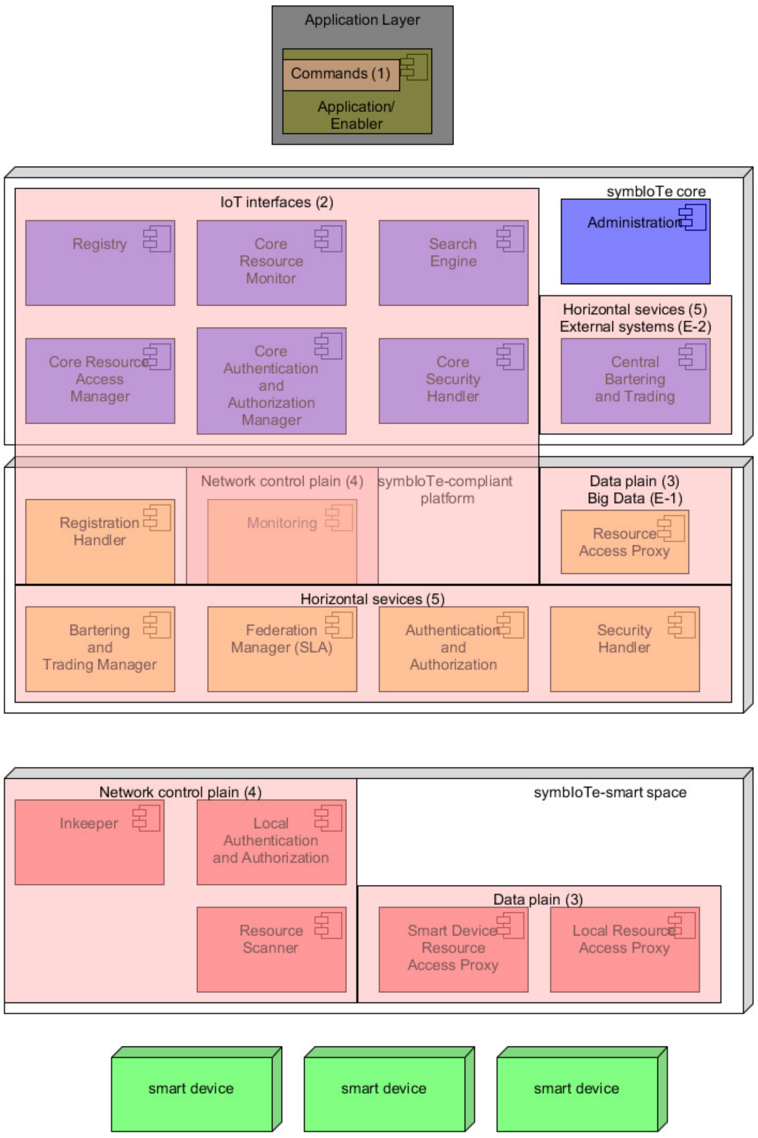

Figure 23 Mapping between the AIOTI HLA interfaces and symbIoTe architecture

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 an updated edition in March 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 (10.2).

The symbIoTe architecture is motivated by the oneM2M functional architecture, and thus it is straightforward to map symbIoTe architectural domains to oneM2M nodes Figure 24 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.

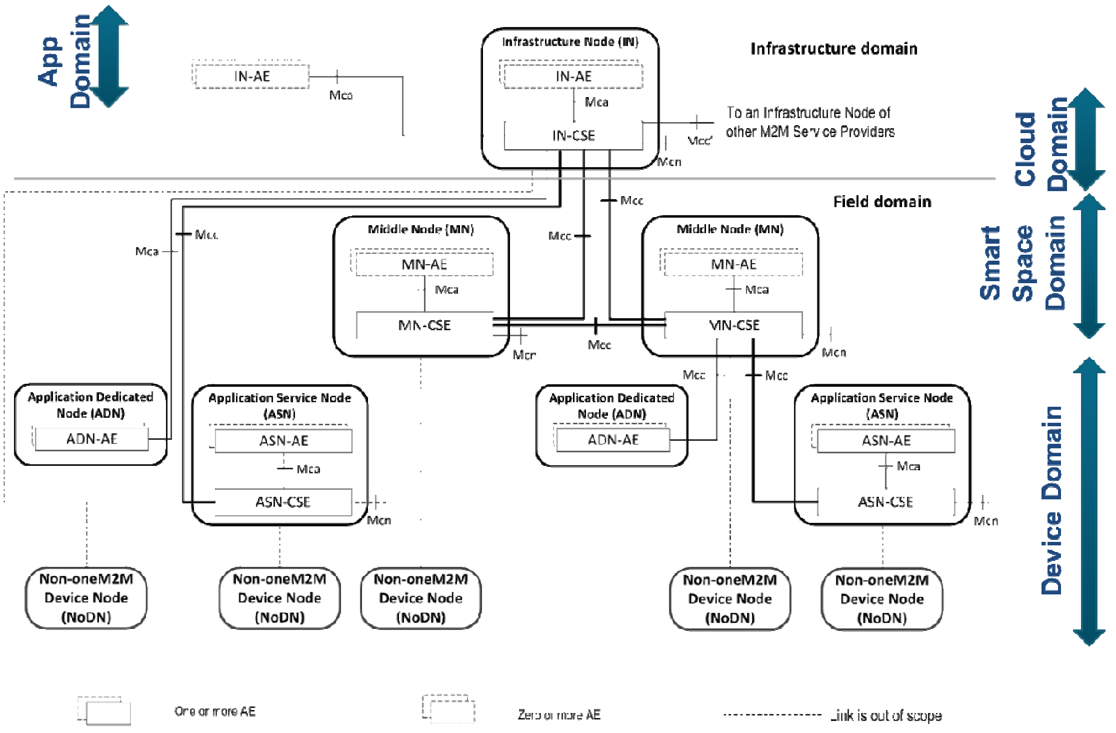
App Domain
IN-AE
Mca
Infrastructure Node (IN)
IN-AE
Mca
IN-CSE
Infrastructure domain
To an Infrastructure Node of other M2M Service Providers
Mcc'
Mcn
Cloud Domain
Field domain
Mcc
Mca
Mcc
Middle Node (MN)
MN-AE
Mca
MN-CSE
Mcn
Mcc
Mcc
Middle Node (MN)
MN-AE
Mca
MN-CSE
Mcn
Smart Space Domain
Mcc
Mcc
Application Dedicated Node (ADN)
ADN-AE
Application Service Node (ASN)
ASN-AE
Mca
ASN-CSE
Mcn
Application Dedicated Node (ADN)
ADN-AE
Application Service Node (ASN)
ASN-AE
Mca
ASN-CSE
Mcn
Device Domain
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
One or more AE
Zero or more AE
Link is out of scope

Figure 24 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. The current focus is on functionalitis related to symbIoTe L1 and L2 compliance. In the case of L1, a symbIoTe application is running outside the platform, while using CSEs provided by the platform, as shown in Figure 25. 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 Infrastrusture Node-Application Entity (IN-AE).

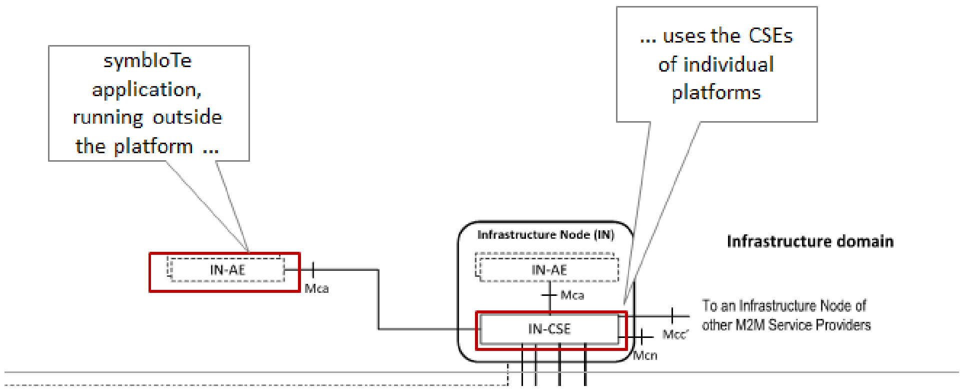
symbIoTe application, running outside the platform ...
... uses the CSEs of individual platforms
IN-AE
Mca
Infrastructure Node (IN)
IN-AE
Mca
IN-CSE
Mcc'
Mcn
Infrastructure domain
To an Infrastructure Node of other M2M Service Providers

Figure 25 Mapping symbIoTe L1 Compliance 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 26 shows the symbIoTe Core Services and their relation to oneM2M CSEs (listed in Appendix, Section 10.2). 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 and Core Security Handler – Security.

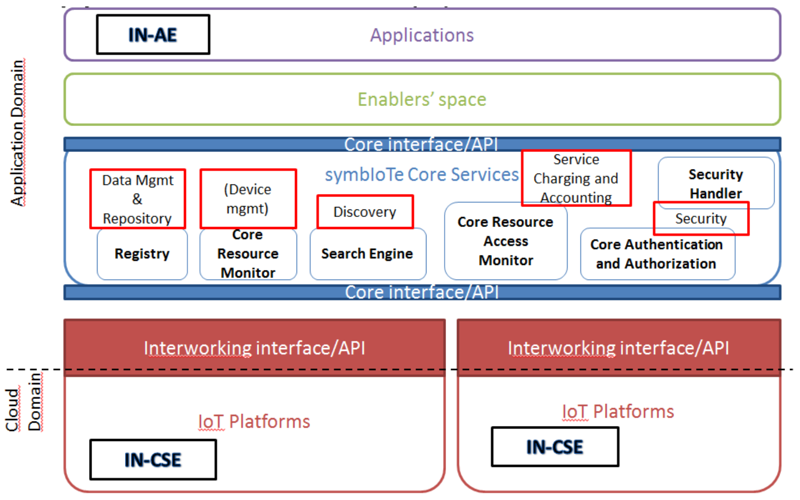

Figure 26 Mapping of symbIoTe Core Services to oneM2M CSEs

In the case of L2, existing (native) applications use CSEs within the platform space and reuse resources of other platforms within a platform federation, as shown in Figure 27. oneM2M already identifies an interaction between two platforms and names this interface

Mcc', but no further details are currently provided regarding the functionaility enabled though this interface. symbIoTe CLD components which are required for L2 Compliance are not yet envisioned in oneM2M standards, and could thus be used to extend the existing CSEs of oneM2M-compliant platforms, especialy the ones related to bartering and trading.

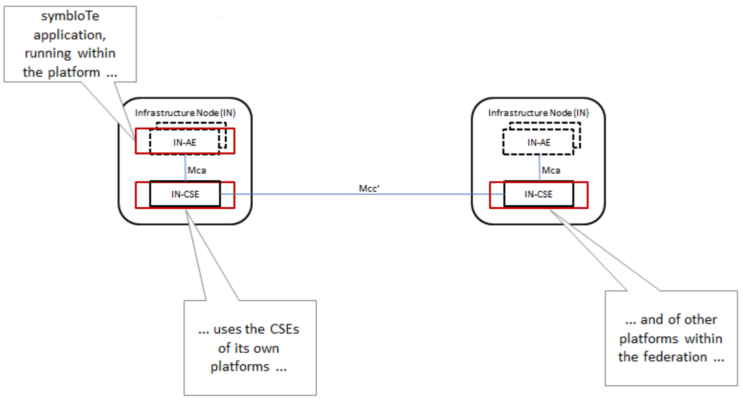
symbIoTe application, running within the platform ...
Infrastructure Node (IN)
IN-AE
Mca
IN-CSE
Mcc'
Infrastructure Node (IN)
IN-AE
Mca
IN-CSE
... uses the CSEs of its own platforms ...
... and of other platforms within the federation ...

Figure 27 Mapping symbIoTe L1 Compliance to oneM2M

Figure 28 shows the symbIoTe CLD components and their mappings to oneM2M CSEs. Those components would need to be integrated within existing CSEs of a oneM2M-compliant platform and to extend their functionalities. They can be mapped to oneM2M CSEs as shown in Figure 28: Registration Handler – Registration; Resource Access Proxy – Communication management & delivery handling, and Subscription and notification; Authentication and Authorization Manager, and Core Security Handler – Security.

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

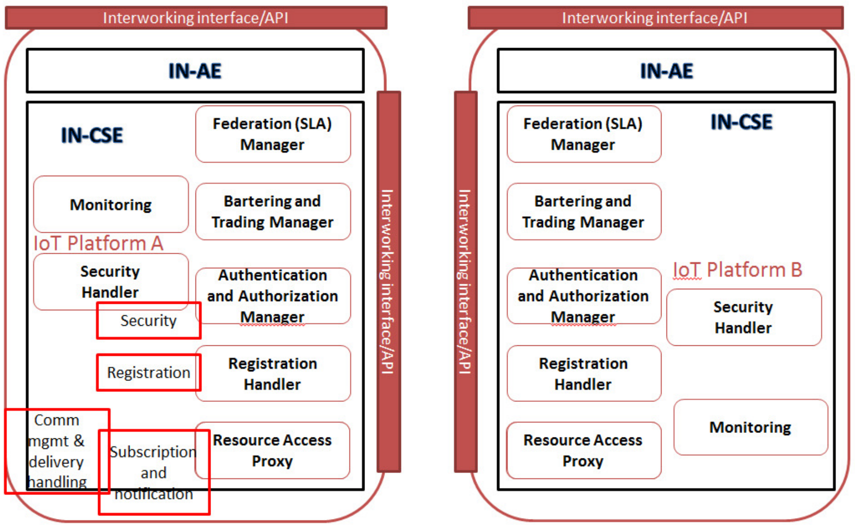
Interworking interface/API
IN-AE
IN-CSE
Federation (SLA) Manager
Monitoring
Bartering and Trading Manager
IoT Platform A
Security Handler
Authentication and Authorization Manager
Security
Registration
Registration Handler
Comm mgmt & delivery handling
Subscription and notification
Resource Access Proxy
Interworking interface/API
Interworking interface/API
IN-AE
Federation (SLA) Manager
IN-CSE
Bartering and Trading Manager
Authentication and Authorization Manager
IoT Platform B
Security Handler
Registration Handler
Monitoring
Resource Access Proxy
Interworking interface/API

Figure 28 Mapping of symbIoTe CLD components to oneM2M CSEs

As it can be seen in Figure 27, 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 is currently not specified in the oneM2M functional architecture, and thus symbIoTe components for Bartering and Trading or Federation cannot be mapped to oneM2M CSEs. This functionality proposed by symbIoTe has potential to be proposed for standardization within the oneM2M standardization process.

Further analysis of the oneM2M standardization and its relation to the symbIoTe Smart Space and Smart Device Domain will be provided in the next and final version of this deliverable (Deliverable 1.4, "Final report on system requirements and architecture").

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 to 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).

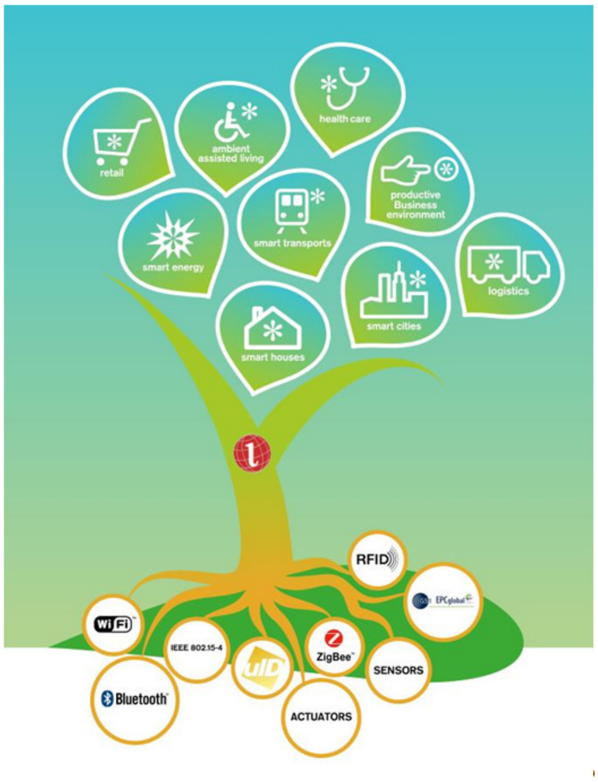

Figure 29 The IoT-A tree [8]

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).

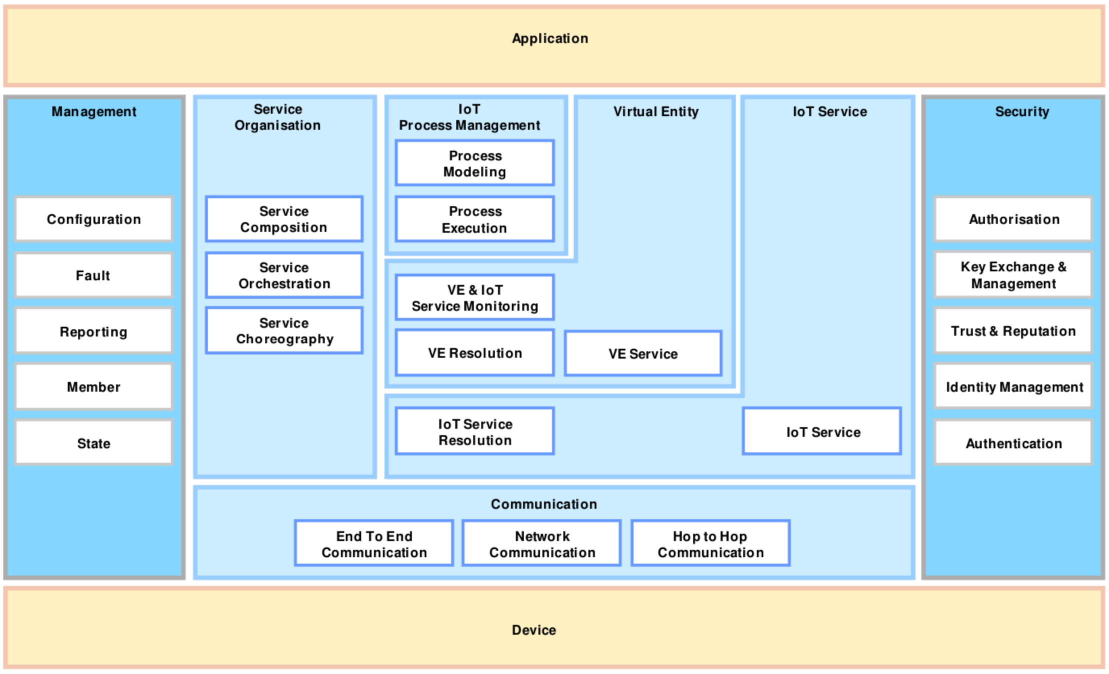

Figure 30 IOT-A reference architecture [8]

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 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 31.

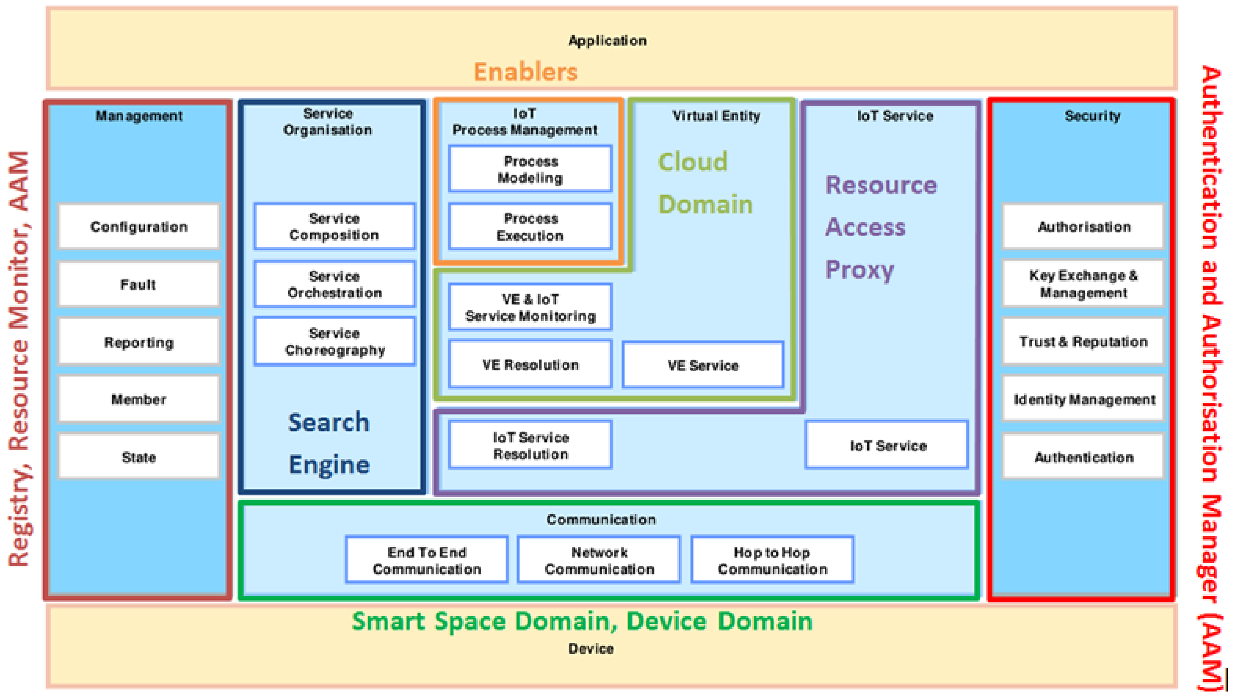

Figure 31 Mapping of symbIoTe to IoT-A reference architecture functional groups

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, as are 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[8] 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.

[8] 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 SmartSpace and symbIoTe 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.

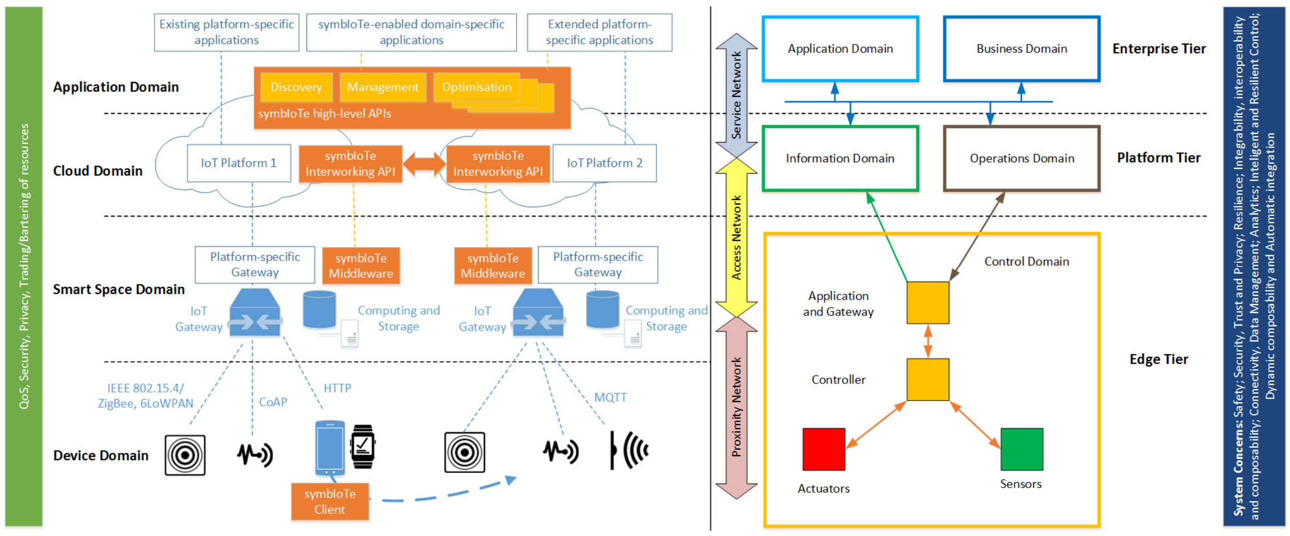
QoS, Security, Privacy, Trading/Bartering of resources
Existing platform-specific applications
symbIoTe-enabled domain-specific applications
Extended platform-specific applications
Application Domain
Discovery
Management
Optimisation
symbIoTe high-level APIs
Cloud Domain
IoT Platform 1
symbIoTe Interworking API
symbIoTe Interworking API
IoT Platform 2
Smart Space Domain
Platform-specific Gateway
symbIoTe Middleware
symbIoTe Middleware
Platform-specific Gateway
IoT Gateway
Computing and Storage
IoT Gateway
Computing and Storage
IEEE 802.15.4/ ZigBee, 6LoWPAN
CoAP
HTTP
MQTT
Device Domain
symbIoTe Client
Service Network
Access Network
Proximity Network
Application Domain
Business Domain
Enterprise Tier
Information Domain
Operations Domain
Platform Tier
Control Domain
Application and Gateway
Controller
Actuators
Sensors
Edge Tier
System Concerns: Safety; Security, Trust and Privacy; Resilience; Integrability, interoperability and composability; Connectivity; Data Management; Analytics; Inteligent and Resilient Control; Dynamic composability and Automatic integration

Figure 32 Comparison between symbIoTe and IIRA achitecture

Further details on IIRA are given in Section 10.3.

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.

6.2 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**[9] provides NGSI9[10] and NGSI10[11] 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.

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

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.

- **KeyRock**[12] 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**[13] is a backend device management. It translates protocols specific to the IoT into NGSI contect 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.

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

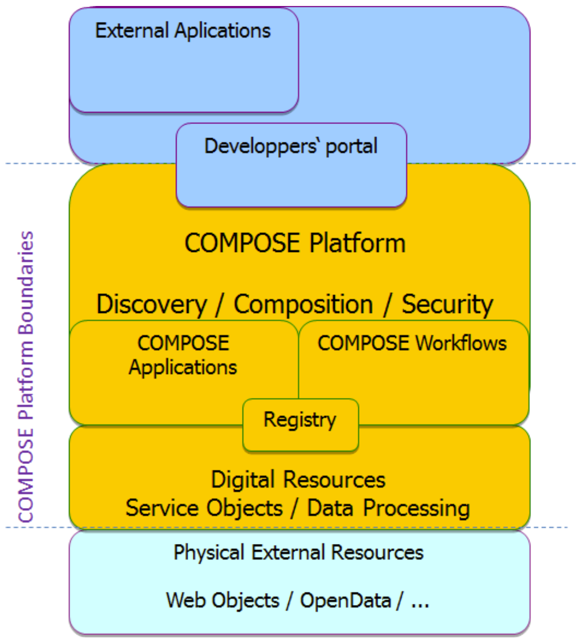
External Aplications
Developpers' portal
COMPOSE Platform
Discovery / Composition / Security
COMPOSE Applications
COMPOSE Workflows
Registry
Digital Resources
Service Objects / Data Processing
Physical External Resources
Web Objects / OpenData / ...
COMPOSE Platform Boundaries

Figure 33 COMPOSE high level architecture [15]

The architecture of COMPOSE system is shown in Figure 33. 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 33 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 34. 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.

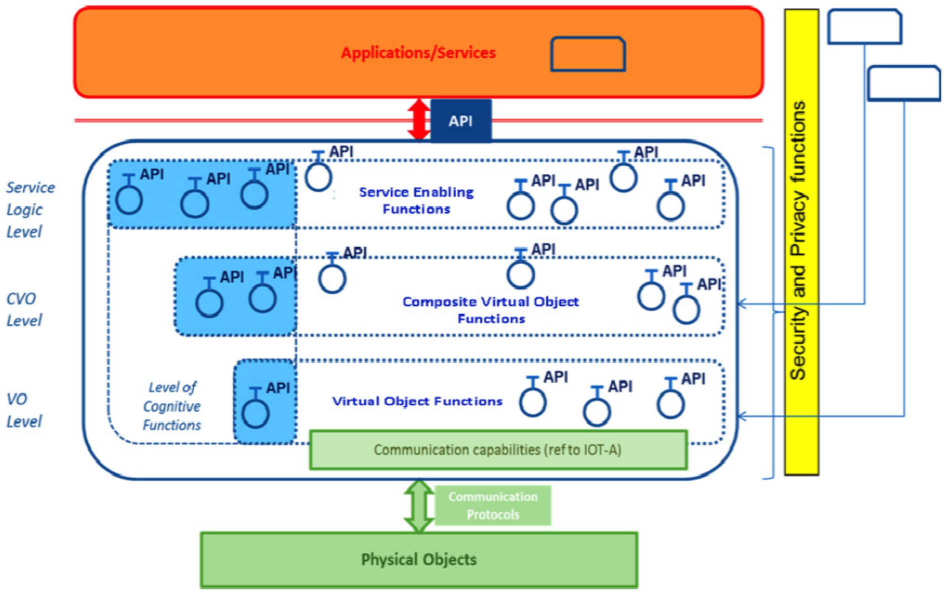
Applications/Services
API
Service Logic Level
CVO Level
VO Level
Service Enabling Functions
Composite Virtual Object Functions
Virtual Object Functions
Level of Cognitive Functions
Communication capabilities (ref to IOT-A)
Communication Protocols
Physical Objects
Security and Privacy functions

Figure 34 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

At this point in time it is possible only to provide a preliminary analysis and comparison of the symbIoTe architecture to other IoT-EPI projects, namely AGILE, BIG-IoT, bIoTope, VICINITY, Inter-IoT and TagItSmart[14], since the corresponding architectures are still being defined and documented. 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. 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 galetway which supports various devices and communication protocols. A preliminary 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*

Platforms by 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 includes also sensor middleware that eases the collection of data from virtually any sensor, while at the same time ensuring their proper semantic annotation. It o
ers visual tools that enable the development and deployment of IoT applications with almost zero programming. Another 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[15], 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 L1 Compliance. Additionally, OpenIoT will be integrated within the symbIoTe IoT federation, enabling L2 Compliance.

[14] For short project descriptions visit http://iot-epi.eu/.
[15] https://www.blackducksoftware.com/about/news-events/releases/black-duck-announces-open-source-rookies-of-year-winners

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 the platform a commercial product whose IPR belongs to Nextworks, internal details cannot be disclosed.

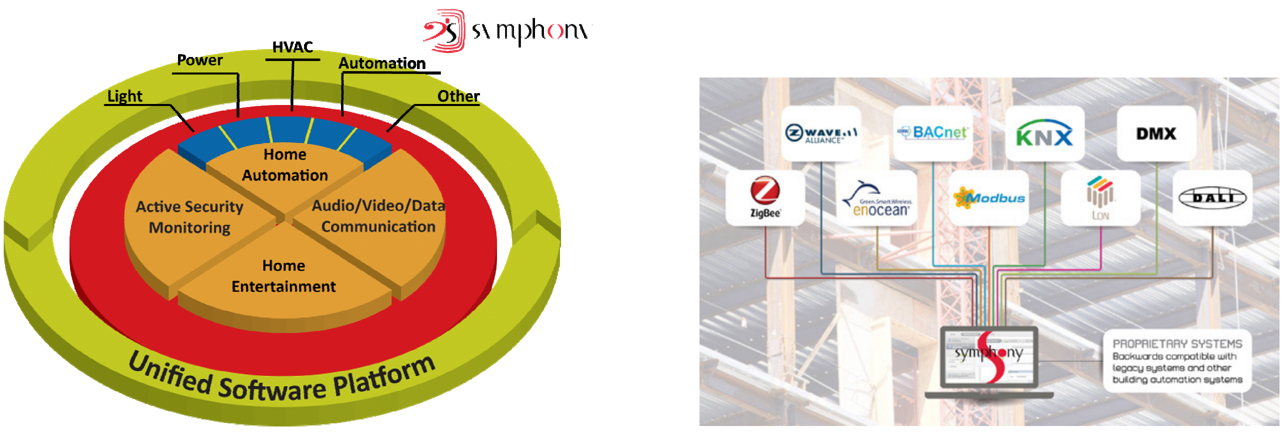
symphony
Power
HVAC
Automation
Light
Other
Home Automation
Active Security Monitoring
Audio/Video/Data Communication
Home Entertainment
Unified Software Platform
Z-WAVE ALLIANCE
BACnet
KNX
DMX
ZigBee
enocean
Modbus
LON
DALI
symphony
PROPRIETARY SYSTEMS
Backwards compatible with legacy systems and other building automation systems

Figure 35 Symphony platform concept

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 Level-3 Compliance, 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, these 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 L1 Compliance, used in smart city use-case. It enables integration of data from many sources, focusing on the mobility aspect of the city. Figure 36 shows which types of services the MoBaaS offers and which kind of devices can be connected to it.

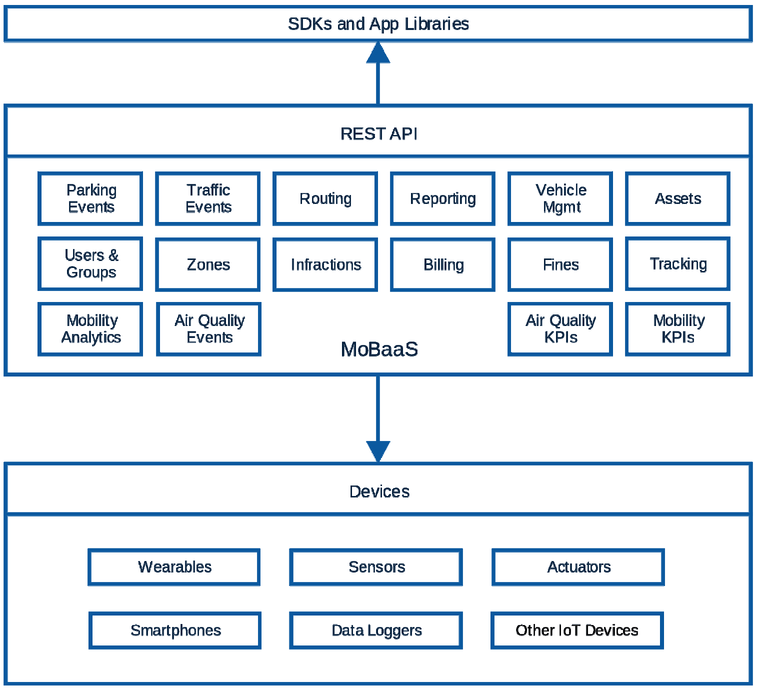

Figure 36 MoBaaS overall architecture

Figure 36 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 BETaaS Platform

BETaaS platform is an *open-source platform* that aims to overcome the limitations of cloud-centric IoT architectures through the design and development of a horizontal platform that leverages a distributed architecture to move the intelligence *to the edge*, i.e., close to the physical environment. BETaaS overcomes the limitations of current M2M applications platforms. It provides a runtime that simplifies the deployment and execution of content-centric M2M applications with a horizontal abstraction, which relies on a local cloud of gateways, enabling the use of Things as a Service over heterogeneous things. The platform offers built-in support for several non-functional requirements: *Context Awareness, Quality of Service, Security, Big Data Management* and *Virtualization*. The software is based on Java OSGi, a framework specifically designed to build modular platforms with a highly dynamic structure.

BETaaS will not create federations as other IoT platforms. Atos will use the experience gained while developing this platform to contribute to the symbIoTe platform architecture and the developed components. BETaaS was designed to integrate heterogeneous devices, and within symbIoTe we will have heterogeneous platforms, so similarities exist within both solutions. symbIoTe can take advantage of the experience gained while developing BETaaS and its adopted solution.

6.3.5 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 Space Domain for the practical deployment of demos, pilots and trials in the context of the Smart Homes services, and in the creation of IoT Federations.

6.3.6 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 that has compared 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 AIOTI 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 covered by symbIoTe. Projects such as COMPOSE, CRYSTAL, iCore and a recent project FIESTA-IOT[16] focus on syntactic and semantic interoperability, but not on organizational interoperability. symbIoTe considers original features related to organizational interoperability than the aforementioned projects 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

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

7 Conclusion and Future Work

This document presents the initial collection of the symbIoTe system requirements and reports a first version of 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. The functional architecture is built around a layered stack in accordance with the AIOTI 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 based on the corresponding requirements and have modeled system behavior supporting syntactic and semantic interoperability (L1 compliance). In the next phase we will define sequence diagrams describing component interactions between platforms forming federations and provide further details about the components for the Smart Space and Smart Device domain. The corresponding sequence diagrams will also be specified to depict the interaction 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, 2015, URL: http://www.onem2m.org/images/files/deliverables/TS-0001-Functional_Architecture-V1_6_1.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

9 Abbreviations

AAM	Authentication and Authorization Manager
APP	Application
BarT	Bartering & Trading component
CLD	Cloud
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
Lx (1-4)	Level x (1 to 4) symbIoTe Compliance
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
SDEV	Smart Device
SLA	Service Level Agreement
SOTA	State of the art
SSP	Smart Space
QoE	Quality of Experience

10 Appendix

10.1 AIOTI high level architecture overview

The AIOTI HLA architecture consists of three layers (application, IoT and network layer) and five interfaces defined between layer entities as shown on Figure 37. The App Entity implements IoT application logic. It is part of the Application layer but it can be deployed on devices, gateways or servers. The IoT Entity exposes IoT functionalities through two interfaces: the interface to IoT capabilities is used by the App Entity, while the Horizontal services interface is used by other IoT Entity (i.e. by other IoT platform). Interfaces for data acquisition (labeled as 3) and interface for network control (labeled as 4) are used in communication between the IoT Entity and Networks. The Networks is an entity in the Network layer that transparently comprise lower layers regarding the network technologies and device management. In the document v. 2.1 [4] the AIOTI HLA is extended with two additional interfaces used to integrate with third-party solutions. The E-1 interface is used for integration with BigData frameworks, while the E-2 interface is used to exchange context information with other systems, such as location, maps, banking, etc.

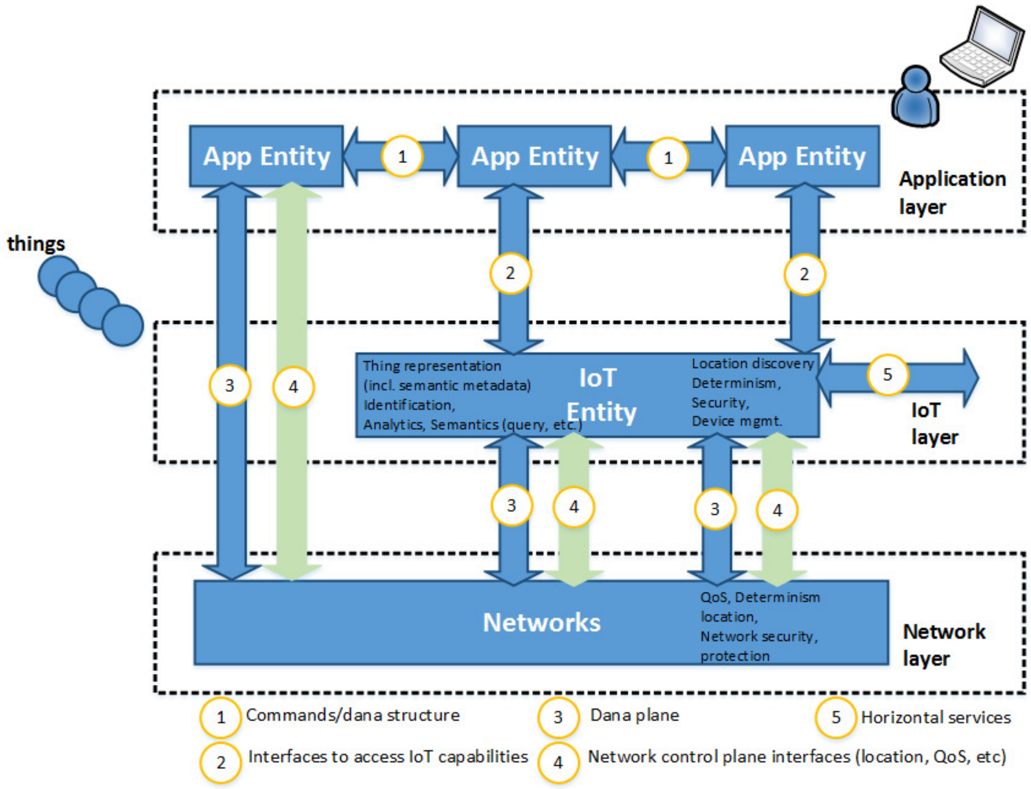
App Entity
App Entity
App Entity
Application layer
things
Thing representation (incl. semantic metadata) Identification, Analytics, Semantics (query, etc.)
IoT Entity
Location discovery Determinism, Security, Device mgmt.
IoT layer
Networks
QoS, Determinism location, Network security, protection
Network layer
1 Commands/dana structure
2 Interfaces to access IoT capabilities
3 Dana plane
4 Network control plane interfaces (location, QoS, etc)
5 Horizontal services

Figure 37 AIOTI high level architecture [4]

10.2 oneM2M functional architecture overview

oneM2M is a standardisation initiative whose main goal is to develop standards for M2M and the Internet of Things. It was established as an association of major standardization organizations from all over the world: ARIB (Japan), ATIS (USA), CCSA (China), ETSI (Europe), TIA (USA), TSDSI (India), TTA (Korea) and TTC (Japan). Apart from standardization organizations, a large number of technical companies (Cisco, Intel, Huawei, ...) also participates in standards development.

As officially stated, the purpose and goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

oneM2M is involved in the preparation of a large number of specifications and reports, a few of the most important ones are listed below:

- use cases and requirements for a common set of Service Layer capabilities;
- Service Layer aspects with high-level and detailed service architecture, in light of an access independent view of end-to-end services;
- protocols/APIs/standard objects based on this architecture (open interfaces&protocols);
- security and privacy aspects; and
- information models and data management

This brief overview of oneM2M efforts presents oneM2M functional architecture, i.e. Service Layer aspects with high-level and detailed service architecture, in light of an access independent view of end-to-end services.

oneM2M Layered Model for supporting end-to-end (E2E) M2M Services comprises three layers: Application Layer, Common Services Layer and the underlying Network Services Layer. oneM2M functional architecture, shown in Figure 38, comprises the following functions:

- **Application Entity (AE)** that implements application service logic (e.g. instance of a fleet tracking application, a power metering application etc.)
- **Common Services Entity (CSE)** that represents an instantiation of a set of common service functions, i.e. services exposed to other entities through Mca and Mcc reference points. Examples of such services include Data management, Device management, Service subscription management.
- **Network Services Entity (NSE)** provides services from the underlying network to CSE. Examples of such services include Device management, Location services, Device triggering. Data transport services from underlying network are not included in NSE.

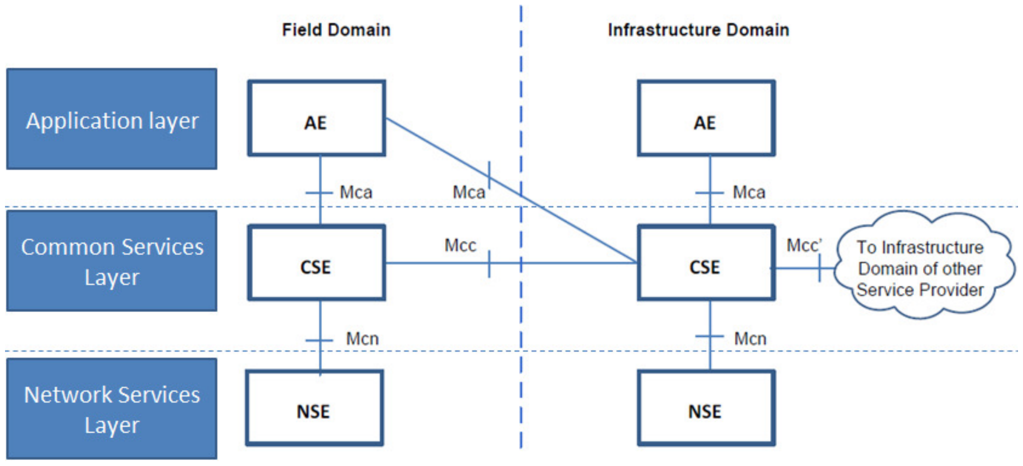

Figure 38 oneM2M functional architecture [7]

Reference points defined between entities of functional architecture (Mca, Mcc, Mcn, Mcc') consist of one or more interfaces of any kind. They enable communication between entities, forwarding data or using the functionalities of other entities. The standard defines various procedures that can be used through those reference points, which will not be elaborated within the scope of this overview.

Configurations supported by oneM2M architecture are shown in Figure 39. Specified nodes (IN, MN, ADN, ASN, ...) are logical entities. Their physical mapping is the following:

- Infrastructure Node (IN): M2M Service Infrastructure
- Middle Node (MN): M2M Gateway
- Application Service Node (ASN) and Application Dedicated Node (ADN): M2M Device

Nodes are either CSE-capable or non-CSE-capable. A CSE-capable node is a logical entity that contains at least one oneM2M CSE. Examples of such nodes are ASN, IN and MN. A non-CSE-capable node contains no oneM2M CSEs. CSEs that reside in different nodes can be different and are dependent on the services supported by the CSE and the characteristics of the physical entity that contains the CSE's node.

Within CSE reside services provided by the Common Services Layer in the M2M System, and are referred to as Common Service Functions (CSFs).

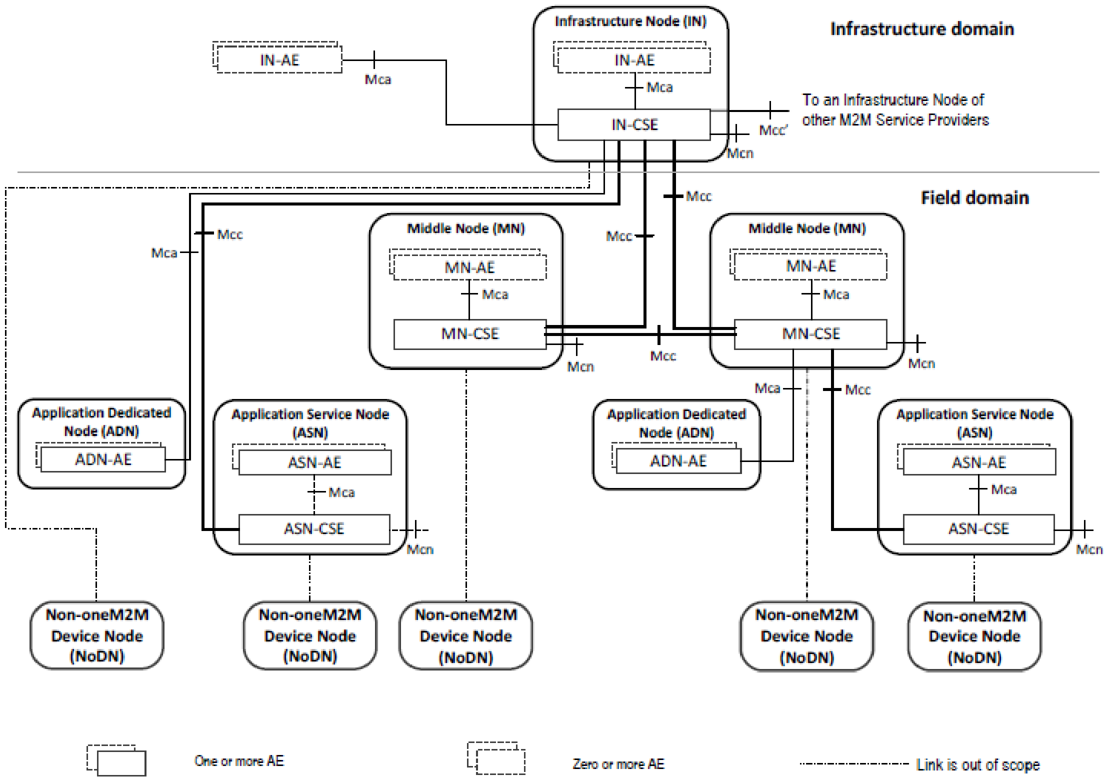
Infrastructure Node (IN)
Infrastructure domain
IN-AE
Mca
IN-AE
Mca
IN-CSE
To an Infrastructure Node of other M2M Service Providers
Mcc'
Mcn
Field domain
Mcc
Middle Node (MN)
MN-AE
Mca
MN-CSE
Mcn
Mcc
Middle Node (MN)
MN-AE
Mca
MN-CSE
Mcn
Mcc
Mca
Mcc
Application Dedicated Node (ADN)
ADN-AE
Application Service Node (ASN)
ASN-AE
Mca
ASN-CSE
Mcn
Mca
Mcc
Application Dedicated Node (ADN)
ADN-AE
Application Service Node (ASN)
ASN-AE
Mca
ASN-CSE
Mcn
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
Non-oneM2M Device Node (NoDN)
One or more AE
Zero or more AE
Link is out of scope

Figure 39 Configurations supported by oneM2M architecture [7]

Common Service Functions (CSFs) residing within CSE are shown in Figure 40. They provide services to the AEs via the Mca reference point and to other CSEs via the Mcc reference point. Underlying network functions are accessed through Mcn reference point.

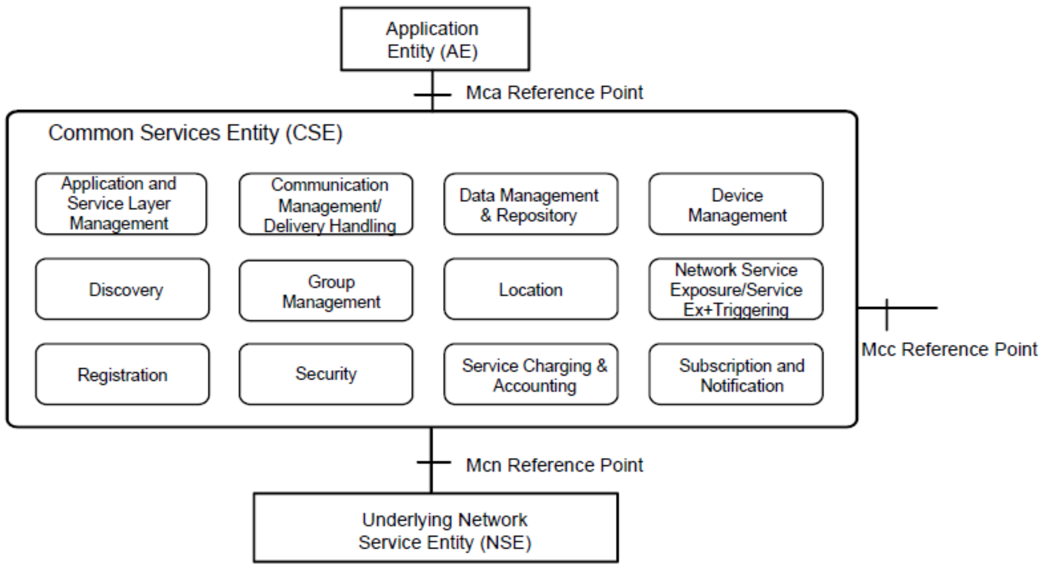
Application Entity (AE)
Mca Reference Point
Common Services Entity (CSE)
Application and Service Layer Management
Communication Management/ Delivery Handling
Data Management & Repository
Device Management
Discovery
Group Management
Location
Network Service Exposure/Service Ex+Triggering
Registration
Security
Service Charging & Accounting
Subscription and Notification
Mcc Reference Point
Mcn Reference Point
Underlying Network Service Entity (NSE)

Figure 40 Common service functions [7]

Common Services Functions are the following:

- **Application and Service Layer Management** is responsible for management of AEs and CSEs on the ADNs, ASNs, and INs. The functionalities it provides are configuring, troubleshooting and upgrading of CSEs and AEs.
- **Communication Management and Delivery Handling** is responsible for communication with other CSEs, AEs and NSEs. It chooses the appropriate communication connection for delivering data, manages buffering and forwarding, and delivery handling policies.
- **Data Management and Repository** is responsible for collecting, converging and storing data for analytics and semantic processing. Some of the data types stored in this repository are contextual information, semantic information, time stamps and location data.
- **Device Management** function is responsible for management of device capabilities on MNs, ASNs and ADNs that reside within an M2M Area Network. It defines specific protocols and data models according to the needs of AEs and CSEs. In order to manage entities in the field domain, Management Server needs to be deployed on IN, while Management Proxy and Management Client need to be deployed on MN/ASN
- **Discovery** function is responsible for searching information about applications and services. It depends on filter criteria which is subject to access control policy.
- **Group management** enables bulk operations on multiple devices, applications or resources. It is responsible for creating, retrieving, updating and deleting the group, adding and removing group members, creating subscriptions and data forwarding.
- **Location** function allows AEs to obtain geographical location information of the nodes. Location server in the network is responsible for storing and updating this information, while GPS modules need to be deployed on devices.
- **Network Service Exposure, Service Execution and Triggering** manages communication with the underlying networks for accessing network service functions. It is responsible for device triggering, small data transmission, location notification, policy rule settings and device management.

- **Registration** function is responsible for delivering AE or CSE information to another CSE in order to use M2M services. AEs register with local CSEs. CSEs register with higher-level CSEs (ASN CSE with MN CSE, MN CSE with IN CSE).
- **Security** function is responsible for sensitive data handling, access control (identification, authentication, authorization, identity management).
- **Service Charging and Accounting** is responsible for charging functions for the Service Layer such as online real-time credit control and charging policies. It captures chargeable events, and generates charging records. The function supports subscription and event based charging.
- **Subscription and Notificiation** handles subscriptions from AEs and CSEs, and provides notifications. Hosting CSE sends notifications to subscribers when modifications to a resource are made.

oneM2M specifies information flows for some of the functionalities of the CSF, but they are not considered in this brief overview.

10.3 Industrial Internet Reference Architecture overview

The Industrial Internet Architecture Framework builds on top of ISO/IEC/IEEE 40201:2011, which aims to standardize the practice of architecture description by defining standard terms and vocabulary and presents a conceptual foundation for expressing, communicating and reviewing architectures and specifying requirements that apply to architecture descriptions, architecture frameworks and architecture description languages. Vocabulary definitions based on 40201:2011 are noted in *italics*.

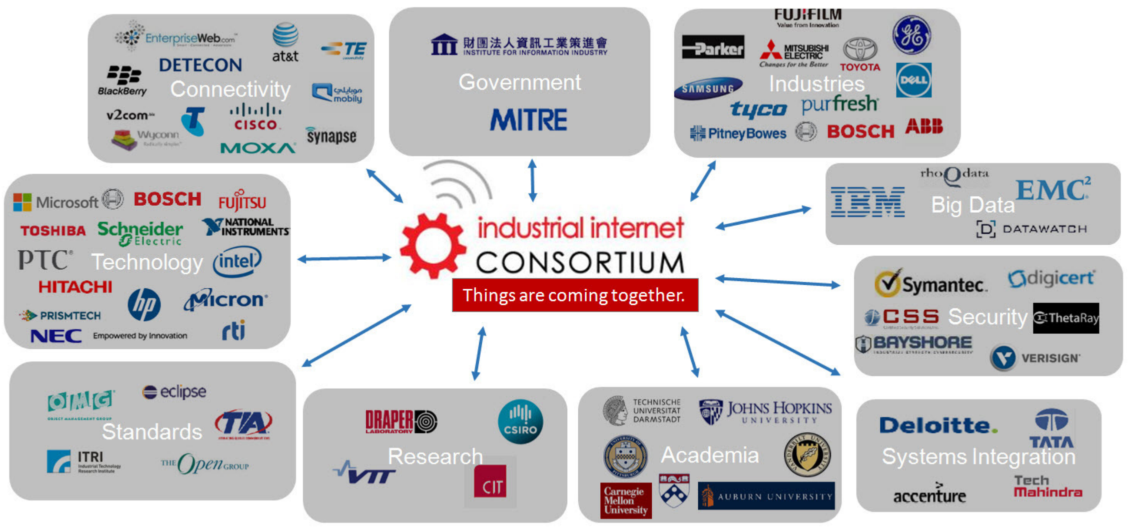

Figure 41 Industrial Internet Consortium members [12]

One aspect is that, by solving the so-called symbol-grounding problem in the IT-domain, the OT domain can benefit from the explicit and transparent provision of the meaning and the context of data. Symbol grounding in that context is a paraphrase for the problem to give symbols used in computer programs or communication their unambiguous meaning in an explicitly defined and preferably computer processable way. This observation stresses the importance of the capability to deal with semantics.

Various topics of interest (*concerns*) related to identified stakeholders are grouped in four *architecture viewpoints,* which deal with different aspects of the concerns, as stated by various *stakeholders*:

- *Business viewpoint* – concerns on business visions, values and objectives.
- *Usage viewpoint* – concerns of system engineers to achieve system capabilities.
- *Functional viewpoint* – interactions and interfaces to other systems.
- *Implementation viewpoint* – technologies needed to implement functions.

In all viewpoints, special care has been given to security by defining security concerns, activities, functions and secure implementations, as the issue of security is crucial in the Industrial Internet due to new nature of openness of previously closed Industrial Control Systems (ICS).

10.3.1 Functional viewpoint

The Functional Viewpoint explains that Industrial Control Systems (ICS), widely deployed to enable industrial automation are currently in a major transformation, from usually closed systems operated by Supervisory Control and Data Acquisition (SCADA) systems towards open systems, utilizing Internet Protocol and Internet architecture, capable of communicating and collaborating with other distributed systems. Two major themes of transformation are identified:

- *Increasing local collaborative autonomy*: New sensing and detection technologies provide more, and more accurate, data. Greater embedded computational power enables more advanced analytics of these data and better models of the state of a physical system and the environment in which it operates. Moreover, ubiquitous connectivity between peer systems enables a level of fusion and collaboration that was previously impractical.
- *Increasing system optimization through global orchestration:* Collecting sensor data from across the control systems and applying analytics, including models developed through machine learning, to these data, we can gain insight to a business's operations. With these insights, we can improve decision-making and optimize the system operations globally through automatic and autonomous orchestration.

Functionalities of the system are further grouped in five *functional domains* which are top-level functional decompositions of any Industrial Internet System, each providing a distinct functionalities in the overall system. General model of functional domains does not necessarily fit all industrial verticals, thus different implementations are possible due to emphasis of any industrial vertical – functional domains can all be implemented within a single system, or can be implemented split up over several sub-systems.

- Control (asset) domain
- Operations domain
- Information domain
- Application domain
- Business domain

Relationships between Functional domains are given in the Figure 42 The functional domains in the IIRA [11] (Green arrows indicate data/information flow, grey/white arrows indicate decision flows and red arrows indicate command/request flows).

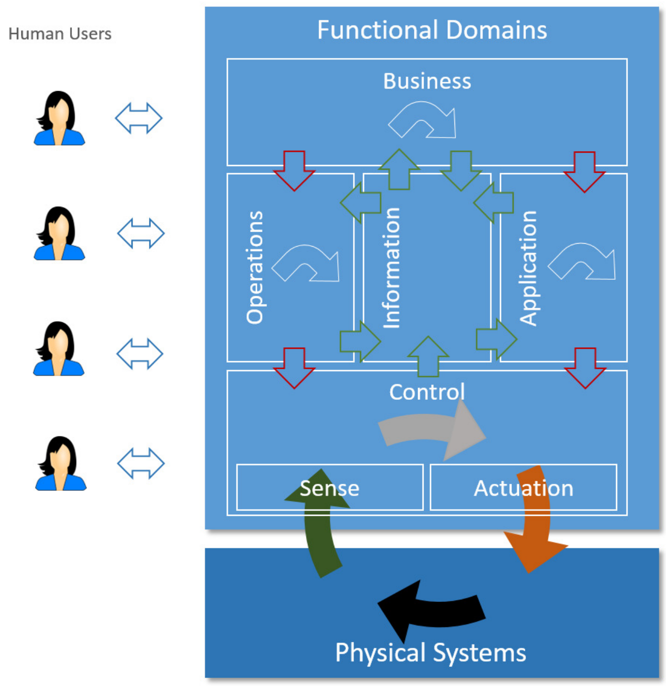

Figure 42 The functional domains in the IIRA [11]

10.3.2 Implementation viewpoint

The Implementation Viewpoint defines general architecture of Industrial Internet System, its structure and the distribution of components, and the topology by which they are interconnected. It also includes technical description of the components, including interfaces, protocols, behaviours and ther properties. Provides implementation map for the key system characteristics and ativities as defined and indetified in *Usage Viewpoint*.

General architecture defines a three-tier architecture, consisting of:

- Edge tier – collects data from the edge nodes, using the proximity network (network connecting the edge nodes to edge gateways)
- Platform tier – receives and forwards control commands between the tiers, it consolidates processes and and analyzes data frlow from edge to enterprise tier.
- Enterprise tier – implements domain-specific applications, decision support systems and interfaces to end-users and other systems.

Architecture also defines three different network environments, each one with its specific functions:

- Proximity network - connection between edge nodes (sensors, actuators, devices, control systems and assets) and edge gateways – bridging short-range communication towards edge nodes towards other networks.

- Access network – connection between Edge gateways and Platforms – can include closed corporate network, overlay private network or public Internet (both fixed or mobile access).
- Service network – connectivity between Platform and Applications, usually an overlay private network over public Internet or Internet network itself.

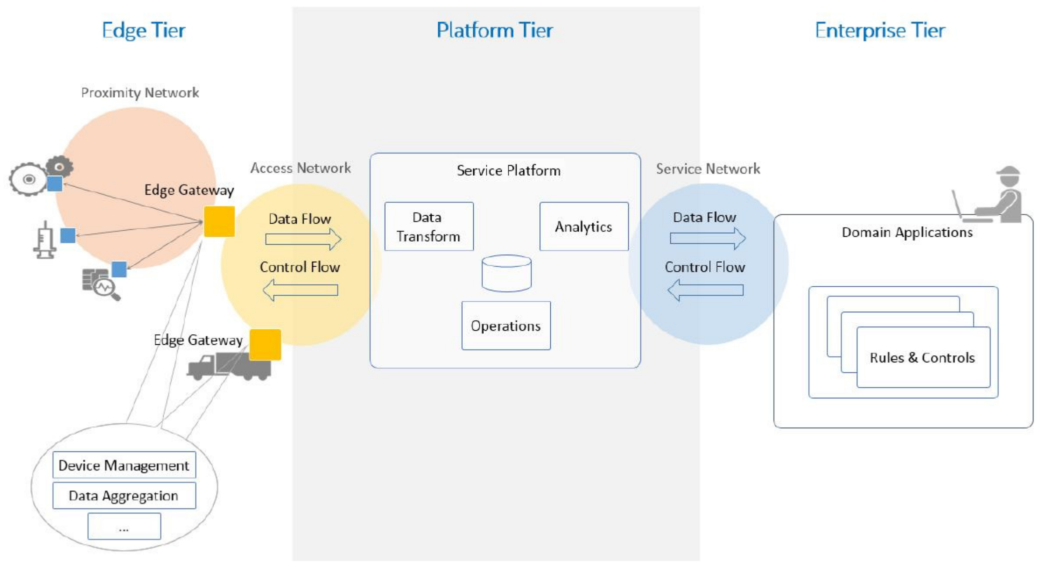

Figure 43 Three-tier general architecture [11]

A Nice view provided by the IIRA incporates concepts of Functional domains from Functional Viewpoint and three-tier general architecture from Implementation Viewpoint, providing additional layer of definition of composition of Functional domains and data flows between domains and accross platform tiers.

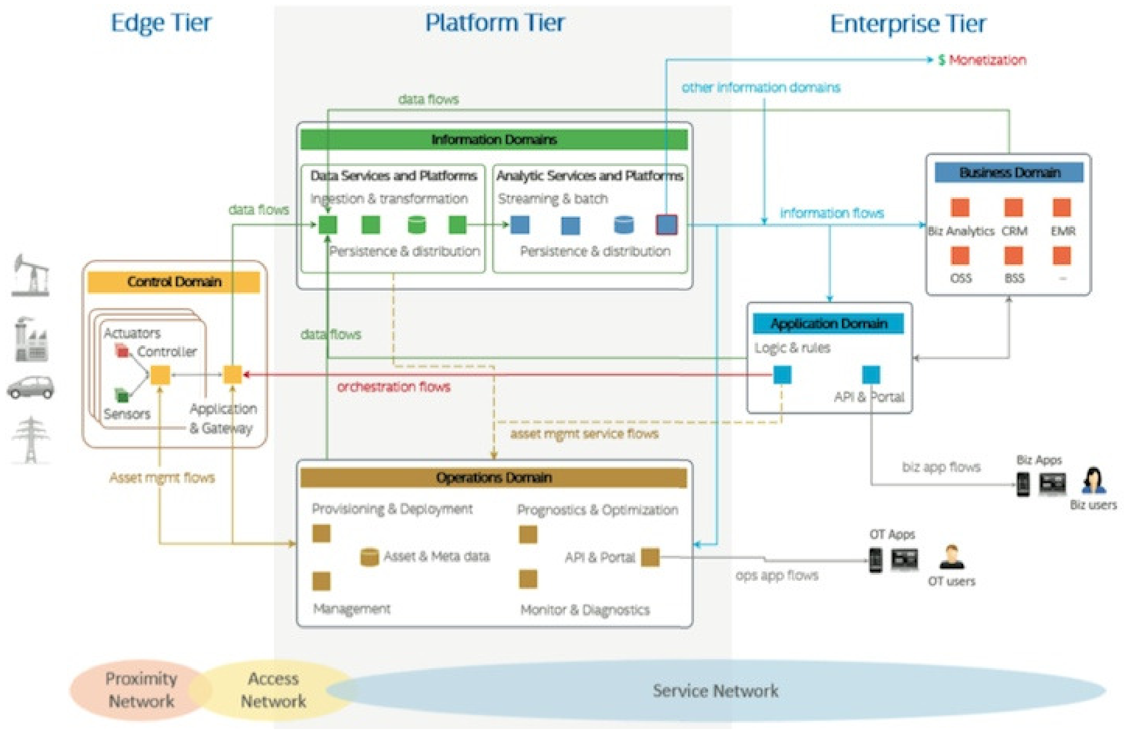

Figure 44 Mapping of the three-tier architecture to functional domains [11]

10.3.3 Key system concerns

Apart from domains and general architecture, IIRA also defines (in Part II of the IIRA) key system concerns as groups of problems that need to be taken in consideration when designing or developing IIS system, in the perspective of the transition of IIS systems from closed system to open and interoperable systems:

- Safety
- Security, trust and privacy
- Resilience
- Integrability, interoperability and composability
- Connectivity
- Data management
- Analytics
- Intelligent and resilient control
- Dynamic composability and automatic integration

10.4 COMPOSE platform overview

COMPOSE is an open-source ecosystem that aims at enabling new services that can seamlessly integrate real and virtual worlds through the convergence of the Internet of Services (IoS) with the Internet of Things (IoT). This is achieved through the provisioning of a scalable marketplace infrastructure, 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 virtualising 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.
COMPOSE project will end by November 2015.

IoT aspects covered:

- Object virtualization: enabling the creation of standardized service objects
- Interaction virtualization: abstract heterogeneity while offering several interaction paradigms
- Knowledge aggregation: creating information from data
- Discovery and advertisement of semantically-enriched objects and services

- Data Management: handle massive amounts and diversity of data/metadata
- Ad hoc creation, composition, and maintenance of service objects and services
- Security, heterogeneity, scalability, and resiliency incorporated throughout the layers

The COMPOSE platform consists of:

- *Service Objects*: an internal digital representation of the *Smart Object* (real-world internet connected physical objects – presented as Digital Resources [15]). COMPOSE service objects communicate with the smart object via a web object, which is a smart object component that interacts with the COMPOSE platform using the COMPOSE protocol.
- *Registry:* holds semantic metadata in the RDF store for the different entities hosted by the platform. The ultimate goal of the registry is to enable *discovery* of COMPOSE components to developers and end-users. In addition, the platform provides an *assisted composition services engine* to help external developers combine the base service objects and *applications* into *workflows* and external applications.The *security architecture* of COMPOSE is based on security metadata which captures security policies of users specifying the privacy level the system must maintain for them.
- *COMPOSE Applications*: re-usable assets built by external developers using the COMPOSE platform IDE and SDK.
- *COMPOSE Workflows*: contain business logic, provided by a developer, and may combine data stemming from Service Objects, with capabilities provided by various COMPOSE applications.
- *Developer's portal*: the main component which external developers encounter when dealing with COMPOSE. Contains basic components such as a SDK, an IDE, a GUI and a marketplace for the final IoT applications.
- *External applications*: not really COMPOSE entities but rather can be viewed as consumers of COMPOSE entities.

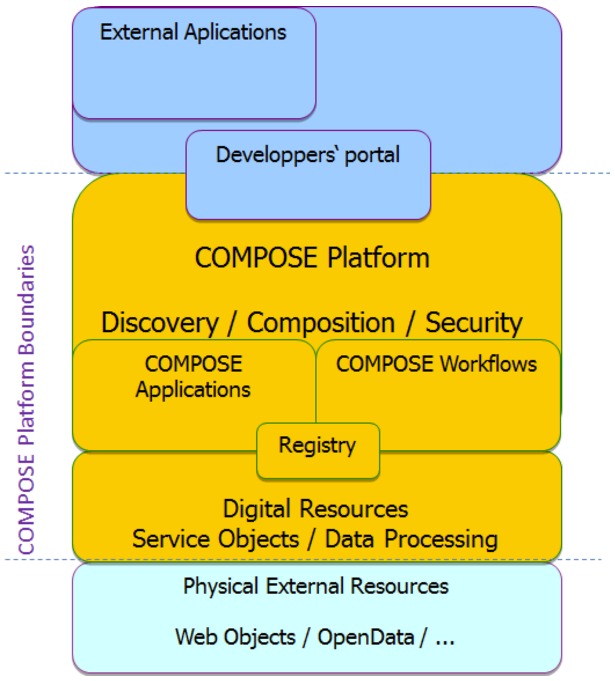
External Aplications
Developpers' portal
COMPOSE Platform
Discovery / Composition / Security
COMPOSE Applications
COMPOSE Workflows
Registry
Digital Resources
Service Objects / Data Processing
Physical External Resources
Web Objects / OpenData / ...
COMPOSE Platform Boundaries

Figure 45 COMPOSE high level architecture [15]

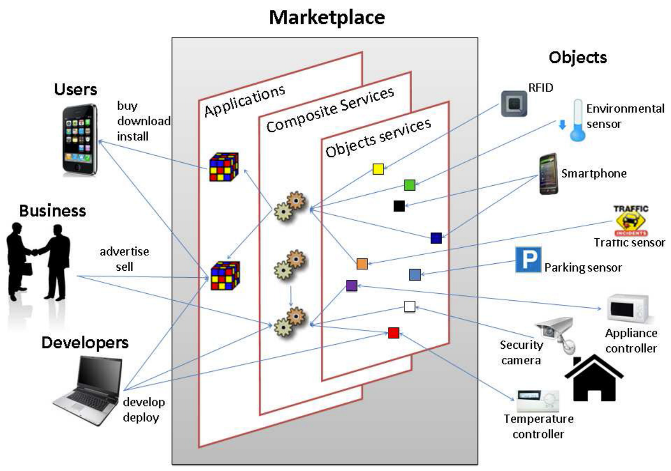

Figure 46 The COMPOSE open marketplace approach [15]

How can symbIoTe take advantage/complement/extend this platform:
Advantages:

- layered architecture
- support for semantic representation and discovery of smart objects
- data management and real-time analytics on incoming data streams
- support for security and standardisation protocols

Disadvantages:

- it is not clear how and which real-time analytics on incoming data streams can be performed

10.5 Crystal platform overview

CRYSTAL aims at the Establishment of Recognized International Open Standards of Lifecycle Tool & Data Integration Platforms for Systems Engineering. Integration and interconnection of tools is a required prerequisite in order to support the collaboration within a development process as well as with customers and suppliers. Interoperability is therefore getting more and more crucial for successful and efficient product engineering. The main technical challenge in addressing this problem is the lack of open and common interoperability technologies.

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.

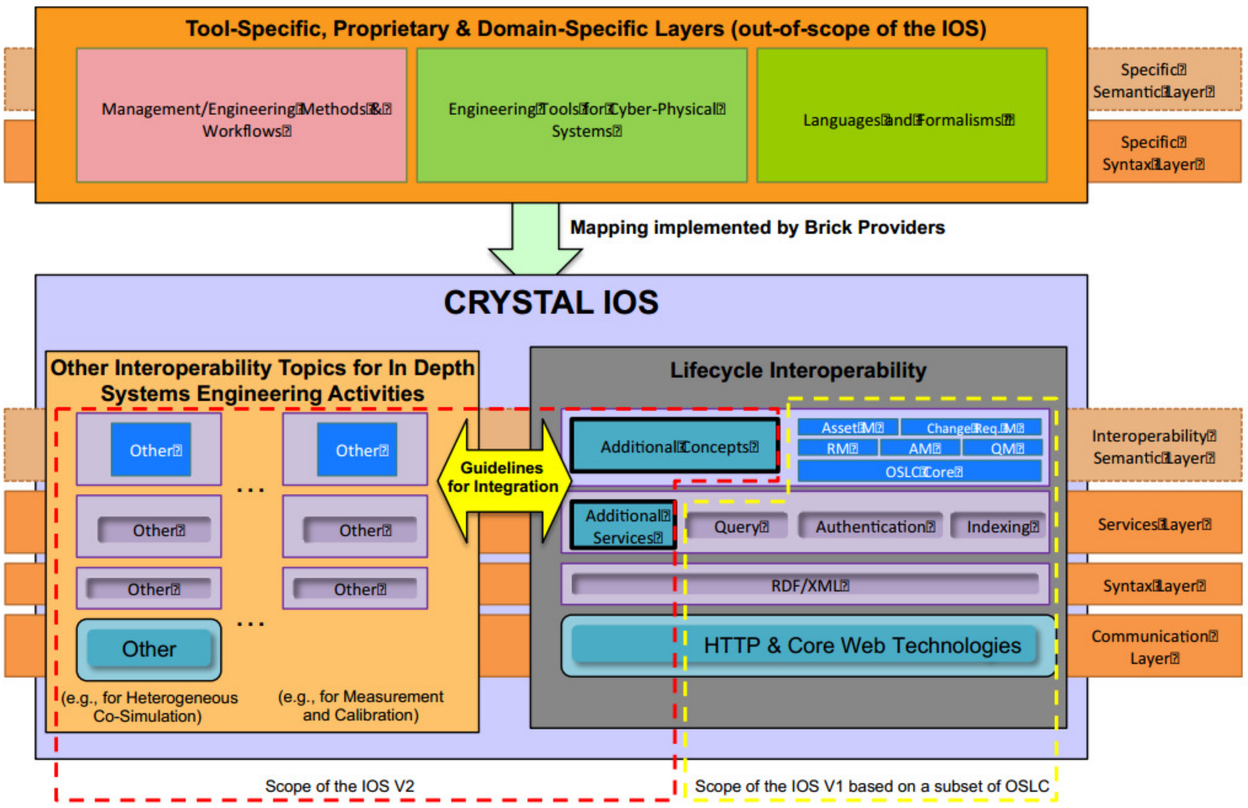
Tool-Specific, Proprietary & Domain-Specific Layers (out-of-scope of the IOS)
Management/Engineering Methods & Workflows
Engineering Tools for Cyber-Physical Systems
Languages and Formalisms
Specific Semantic Layer
Specific Syntax Layer
Mapping implemented by Brick Providers
CRYSTAL IOS
Other Interoperability Topics for In Depth Systems Engineering Activities
Other
Other
Other
Other
Other
Other
Other
(e.g., for Heterogeneous Co-Simulation)
(e.g., for Measurement and Calibration)
Guidelines for Integration
Lifecycle Interoperability
Additional Concepts
Asset M
Change Req. M
RM
AM
QM
OSLC Core
Additional Services
Query
Authentication
Indexing
RDF/XML
HTTP & Core Web Technologies
Interoperability Semantic Layer
Services Layer
Syntax Layer
Communication Layer
Scope of the IOS V2
Scope of the IOS V1 based on a subset of OSLC

Figure 47 The Crystal IOS layered architecture [23]

CRYSTAL is not focusing on IoT platforms but rather on engineering processes that allow embedded systems and management platforms to share the same data across different engineering processes, mainly handled by proprietary and close systems. The vision of CRYSTAL is not to create APIs across system to exchange data but rather to create a data repository that allows different systems to access the data, providing also the necessary semantics for each system to use the data. In this sense, CRYSTAL is not close to IoT, but the general concept of sharing data among different systems, which in the IoT domain are the smart objects and the external Cloud services, can be worth investigating.

CRYSTAL refers to several approaches with respect to the interoperability of data across different systems. Such examples are i) Open Services Lifecycle Collaboration (OSLC) approach (http://open-services.net/specifications/) and ii) the ReqIF data exchange format (www.omg.org/spec/ReqIF/).

Moreover, CRYSTAL claims that it builds upon and extends the Interoperability Specifications (IOS), initially defined by former ARTEMIS projects (i.e., iFEST (http://www.artemis-ifest.eu), MBAT (http://www.mbat-artemis.eu)). IOS consists of a specification for achieving common Tool & Data Interoperability in heterogeneous Systems Engineering development environments. In particular, it encompasses the specification of three main aspects:

- The specification of communication paradigms and protocols to be used for exchanging information between integrated Tools and Data repositories,
- The specification of data exchange formats (or syntax, referring to the formats used for serializing data as strings, e.g., RDF/XML, XMI/XML, JSON, etc.), and
- The specification of the semantics of the information to be interpreted and exchanged across these Tools and Data repositories (or abstract syntax, referring to the definition of sets of concepts for lifecycle integration, defined with their properties and relationships).

Since resource semantics are important to symbIoTe, it may be worth investigating further into the aforementioned approaches. However, a paradigm shift will be required since these technologies do not refer to the IoT and smart objects domain.

10.6 iCore platform overview

The iCore platform provides a cognitive management framework that facilitates and supports Internet-connected objects. The platform conceals the technological heterogeneity of IoT resources, while supporting application provision and business integrity in line with the perspective of various users/stakeholders. The framework is based on the layered architecture, consisting of virtual objects (VOs), composite virtual objects (CVOs) and functional blocks (service level) for representing the user perspectives, where VOs are cognitive virtual representations of legacy real-word objects and digital objects, CVOs are cognitive mash-ups of semantically interoperable VOs that combine information and services from the lower layer in order to provide building relationships and achieve added value, while the topmost layer provides unified functional parts for each service. Each layer offers scalable mechanisms for the registration, look-up and discovery of entities, and the composition of services handles the management of events as well as cooperation between objects. The iCore platform is equipped with security protocols/functionality, which span all levels of the framework, and consider the ownership and privacy of data, as well as controlling the actual access to objects. To validate the proposed solutions iCore addresses the following application areas: ambient assisted living, smart office, smart transportation and supply chain management.

The EU iCore project won the Best Exhibit Award at IoT360 Summit in 2014.

IoT aspects covered:

- Virtual representation of physical sensors/digital devices
- Dynamic sensor discovery
- Semantic representation of virtual objects (RDF triplets)
- Intelligent decisions based on cognitive properties

The iCore architecture comprises several levels of functionality:

- **Virtualization and composition of objects.** Real World Objects (RWO) can be represented as simple Virtual Objects (VOs) that can be aggregated and merged in order to create new Virtual Composite Objects (VCOs) that extend and generalize RWOs functionalities and features. RWOs are semantically described using RDF triplets.
- **Segmentation and aggregation of functions.** Objects are framed in three levels. At each level, the iCore architecture envisages an increasing number of functionalities and systematic entities used to support the architecture.
- **Functional and systematic view of objects.** Any single object provides interfaces and functions that support integration of the object into the system.
- **Cognition.** iCore is structured as a cognitive cycle in which knowledge is derived from observing the external environment which is continually evolving and decisions are inferring its future behavior based on different criteria. iCore uses different machine learning techniques, semantic reasoning and pattern recognition techniques.
- **Security and privacy.** iCore can authenticate users and authorize them against VOs/CVOs.

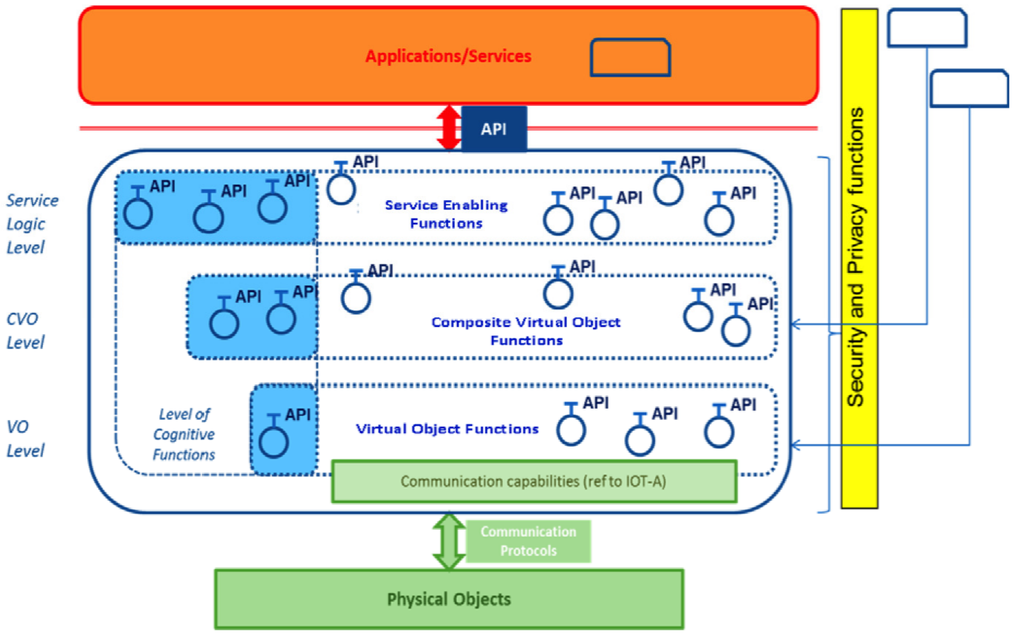
Applications/Services
API
Service Logic Level
CVO Level
VO Level
Level of Cognitive Functions
Service Enabling Functions
Composite Virtual Object Functions
Virtual Object Functions
Communication capabilities (ref to IOT-A)
Communication Protocols
Physical Objects
Security and Privacy functions

Figure 48 iCore architecture [16]

How can symbIoTe take advantage/complement/extend this platform:
Advantages:

- layered organization with virtual objects and composite virtual objects
- support for semantic representation of virtual objects
- various cognitive mechanisms.

Disadvantages:

- the layered organization does not easily map to the symbIoTe hierarchical organization with local IoT domains (cloudlets) and the cloud domain
- it is unclear which "cognitive mechanisms" are available and whether they are indeed scalable or not

10.7 FIWARE platform overview

FIWARE is an open initiative aiming to create a sustainable ecosystem to grasp the opportunities that will emerge with the new wave of digitalization caused by the integration of recent Internet technologies. The high-level goal of the FIWARE project is to build the Core Platform of the Future Internet that will dramatically increase the global competitiveness of the European ICT economy by introducing an innovative infrastructure for cost-effective creation and delivery of versatile digital services, providing high QoS and security guarantees. As such, it will provide a powerful foundation for the Future Internet, stimulating and cultivating a sustainable ecosystem for (a) innovative service providers delivering new applications and solutions meeting the requirements of established and emerging Usage Areas; and (b) end users and consumers actively participating in content and service consumption and creation. The entire FI WARE platform is based upon different elements (i.e. General Enablers, GEs) which offer reusable and commonly shared functions serving a multiplicity of Usage Areas across various sectors. Key goals of the FI-WARE project are the identification and specification of GEs, together with the development and demonstration of reference implementations of identified GEs. Any implementation of a GE comprises a set of components and will offer capabilities and functionalities which can be flexibly customized, used and combined for many different Usage Areas, enabling the development of advanced and innovative Internet applications and services.

FIWARE initiative consists of several parts: FIWARE platform, FIWARE Lab, FIWARE Ops, FIWARE Accelerate and FIWARE Mundus.

IoT aspects covered:

- scalable discovery and look-up of IoT resources
- data management
- semantic data annotation
- scalable, elastic and high performing management of the IoT
- mobility of IoT resources
- QoS and security

The Core Platform provided by the FIWARE project is based on GEs linked to the following areas:

- Cloud Hosting – the fundamental layer which provides the computation, storage and network resources, upon which services are provisioned and managed.
- Data/Context Management – the facilities for effective accessing, processing, and analyzing massive volume of data, transforming them into valuable knowledge available to applications. Main components are Publish/Subscribe Broker GE, Complex Event Processing GE, Big Data Analysis GE, Query Broker GE, Semantic Annotation GE, Meta-data pre-processing GE, etc.
- Applications, Services and Data Delivery – the infrastructure to create, publish, manage and consume FI services across their life cycle, addressing all technical and business aspects.
- Internet of Things (IoT) Service Enablement – the bridge whereby FI services interface and leverage the ubiquity of heterogeneous, resource-constrained devices in the Internet of Things. Main IoT GEs are IoT Communication, IoT Resources Management, IoT Data Handling and IoT Process Automation.
- Interface to Networks and Devices (I2ND) – open interfaces to networks and devices, providing the connectivity needs of services delivered across the platform.
- Security – the mechanisms which ensure that the delivery and usage of services is trustworthy and meets security and privacy requirements.
- Advanced Web-based User Interface – new user input and interaction capabilities, such as interactive 3D graphics and immersive interaction with the real and virtual world.

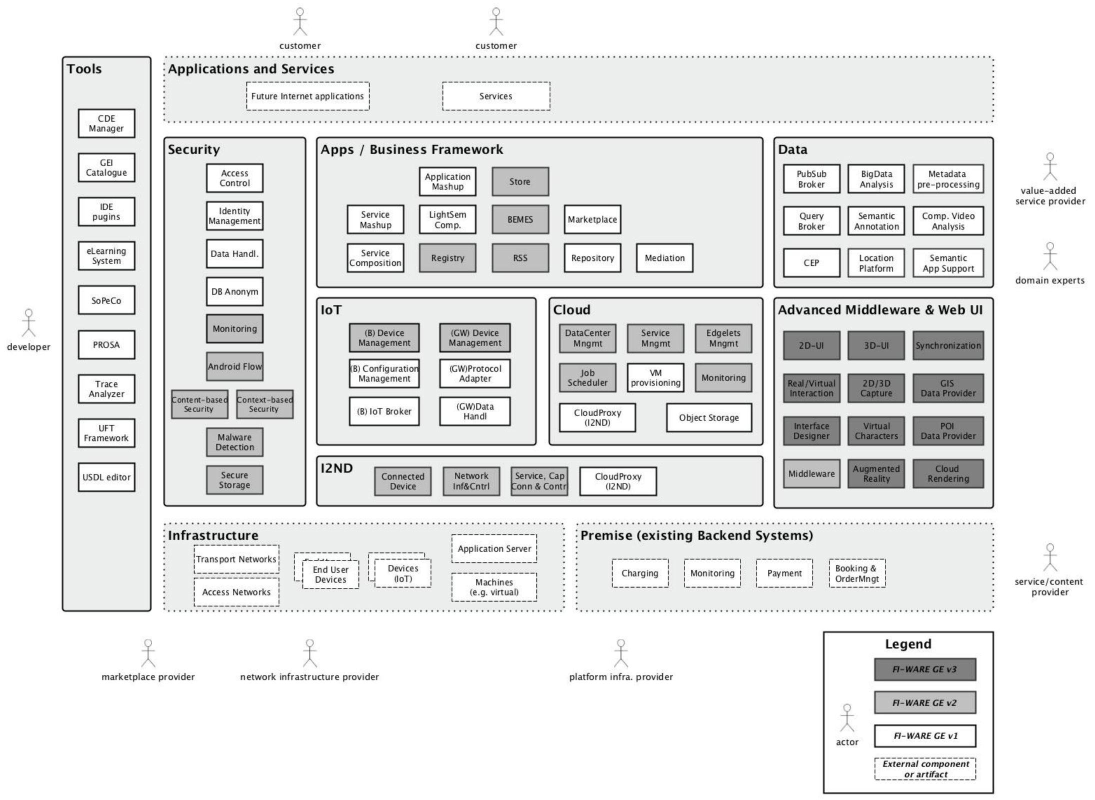

Figure 49 FIWARE platform architecture [24]

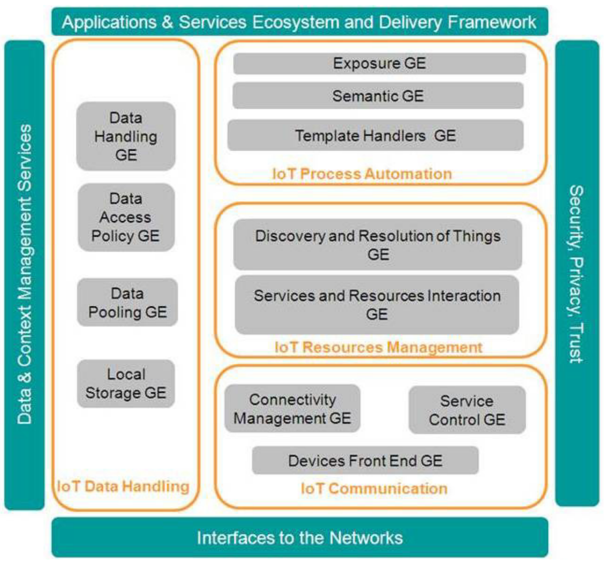

Figure 50 FIWARE IoT service enablement platform [24]

How can symbIoTe take advantage/complement/extend this platform:
Advantages: data management and real-time event processing capabilities, support for semantic data annotation, security, reusable GEs

Disadvantages: it is not intuitive how to use GEs implemented within the FIWARE Lab