Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

D4.1 – symbIoTe Smart Space Middleware Tools, Protocols and Core Mechanisms

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: SymbIoTe Smart Space Middleware Tools, Protocols and Core Mechanisms

Type: Public

Editor(s): Gianluca Insolvibile, Matteo Pardi (NXW)

E-mail: g.insolvibile@nextworks.it, m.pardi@nextworks.it

Author(s): Gianluca Insolvibile, Matteo Pardi (NXW), Pavle Skočir (UNIZG-FER), João Garcia (UW), Daniele Croce, Domenico Garlisi, Fabrizio Giuliano, Michele Gucciardo, Ilenia Tinnirello (CNIT), Karl Kreiner (AIT), Nemanja Ignjatov, Svenja Schröder (UNIVIE)

Doc ID: D4.1-v2.1

Amendment History

Version	Date	Author	Description/Comments
v0.0	August 19, 2016	Matteo Pardi (NXW)	Initial Table of Contents
v0.1	October 07, 2016	Matteo Pardi (NXW)	Partner contribution assignment proposal
v0.2	November 30, 2016	Matteo Pardi (NXW)	Added NXW contents
v0.3	December 8, 2016	Matteo Di Fraia (UNIDATA)	Added UNIDATA contents
v0.3	December 8, 2016	Pavle Skočir (UNIZG-FER)	Added UNIZG-FER contents
v0.3	December 8, 2016	João Garcia (UW)	Added UW contents
v0.3	December 12, 2016	Daniele Croce, Domenico Garlisi, Fabrizio Giuliano, Michele Gucciardo, Ilenia Tinnirello (CNIT)	Added CNIT contents in sec 3.2, 4.3 and 4.8
v0.3	December 22, 2016	Matteo Di Fraia (UNIDATA)	Updated UNIDATA contents
v0.3	December 22, 2016	Pavle Skočir (UNIZG-FER)	Updated UNIZG-FER contents
v0.3	December 23, 2016	Daniele Croce (CNIT)	Added paragraphs in sec 4.8 on LoRa optimization and on the integration of heterogeneous wireless technologies
v0.4	December 30, 2016	Gianluca Insolvibile (NXW)	Merged contributions and reorganized ToC.
v0.4	January 4, 2017	João Garcia (UW)	Added UW contents
v0.5	January 9, 2017	Gianluca Insolvibile (NXW)	Merged contributions
v0.5	January 9, 2017	Kar Kreiner (AIT)	Added AIT contents
v0.5	January 9, 2017	Pavle Skočir (UNIZG-FER)	Updated UNIZG-FER contents
v0.5	January 10, 2017	Matteo Di Fraia (UNIDATA)	Updated UNIDATA contents
v0.6	January 11, 2017	Matteo Pardi (NXW)	Merged contributions
v0.6	January 12, 2017	Nemanja Ignjatov (UNIVIE)	Added UNIVIE contents (par 4.7)
v0.6	January 13, 2017	Nemanja Ignjatov (UNIVIE)	Added UNIVIE contents (par 4.5)
v0.7	January 13, 2017	Matteo Pardi (NXW)	Merged contributions
v1.0rc1	January 13, 2017	Gianluca Insolvibile (NXW)	First rc version

v1.0rc2	January 27, 2017	Gianluca Insolvibile, Matteo Pardi (NXW)	Second rc version
V2.0	January 31, 2017	Matteo Pardi (NXW)	Final version
V2.1	February 6, 2017	Sergios Soursos (ICOM)	Submission ready version

Table of Contents

1 Executive Summary 7

2 Introduction 8

3 Objectives and scope 9

3.1 Smart Spaces 9

3.1.1 Dynamic adaption to changing topology 12

3.1.2 IoT platform interoperability at the SSP level 12

3.1.3 Cloud dependency 13

3.1.4 Architectural mapping to existing IoT platforms 14

3.2 Smart Devices 16

3.2.1 Objects roaming 17

3.2.2 Level3 vs. Level4 compliance 18

3.2.3 LoRaWAN Technology 19

3.3 System requirements 21

4 Smart Space and Smart Device 24

4.1 Smart Space Architecture 24

4.2 SymbIoTe middleware 25

4.2.1 Innkeeper 26

4.2.2 Resource Scanner 27

4.2.3 SSP RAP 27

4.2.4 SDEV RAP 28

4.2.5 Local AAM 28

4.2.6 IoT platform's symbIoTe agent 29

4.3 Smart Device Architecture 29

4.4 Relationship with Core and Cloud components 31

4.5 Security considerations 33

4.6 Registration, discovery and interoperability of smart objects 35

4.7 Smart Device event notification mechanism 39

4.7.1 Events structure and integration 41

4.8 Wireless technologies 41

4.8.1 Detect interfering technologies by means of Wi-Fi interfaces 43

4.8.2 Set-up of cross-technology communication channels 44

4.8.3 Support cross-network TDMA 44

4.8.4 Transmit to multiple gateways 45

4.8.5 Support device configuration protocols 45

4.8.6 Fast attach *46*

5 Components interaction **47**

5.1 App joining Smart Space 47

5.2 Smart Device joining Smart Space 48

5.3 Access to SDEV resources from within the SSP 50

5.4 Access to SDEV resources from outside the SSP 51

6 Conclusions and Next Steps **52**

7 References **53**

8 Acronyms **55**

(This page is left blank intentionally.)

1 Executive Summary

Smart Spaces and Smart Devices complete the symbIoTe architectural view of IoT platforms at the earthly level, as opposed and complementary to the cloud level, by enclosing physical resources in virtual spaces.

A Smart Space (SSP) is an environment where one or more IoT platforms coexist, each of them providing some kind of service. Such environments are typically identified with physical locations, which can range from wide spaces to small areas; a Smart Space defines abstract boundaries for the IoT services and platforms it embraces, and acts as a sort of gateway from local resources to the rest of the symbIoTe environment.

A Smart Device (SDEV) relies on the functions provided by the SSP, in order to roam, associate, and be accessible from the symbIoTe Core or by any symbIoTe app. As will be described in this document, any device capable of complying with the symbIoTe SSP interface for registration and resource access can be considered an SDEV.

The design of the SSP and SDEV domains will bring to the development of a set of proper software components (the symbIoTe Smart Space Middleware, S3M). Such components will provide symbIoTe functions inside the SSP, locally handling the mechanisms of resources registration and access, replicating the paradigm adopted in the Cloud domain. Therefore, the design of the S3M has been influenced by the analysis of the relationships with the Core and the Cloud components and the complex interactions of software modules both inside and outside the SSP.

The main technical objectives that have been pursued in the design and implementation of the Smart Space and Smart Device Domains are the dynamic discovery and automatic configuration of resources, the IoT platform interoperability at the SSP level (platform transparency), a set of baseline operations and the support for nomadic devices. It is important for the SSP to be able to provide its functionality even when the cloud components are not reachable, in order to allow the user to interact with the SSP in every kind of connectivity conditions. For example, in the Smart Residence Use Case it is essential that devices keep working (i.e. they can be read and they can perform operations) even if it is temporarily (or even permanently) not possible to communicate with the Cloud.

The focus of the SSP and SDEV analysis, and the focus of this document accordingly, is on identifying general principles, interfaces and mechanisms, and defining a reference architecture for symbIoTe compliant systems that can be applied to any IoT platform. The specific components design, on the other hand, will be targeted at IoT platforms that are included in the project or well-known anyway, and despite being as generalized as possible it does not purport to be valid for any IoT platform.

2 Introduction

This deliverable presents the Smart Space “vision” developed within the project. In the context of this document, Smart Spaces and Smart Devices are the environment where the symbIoTe Middleware lives, with all its features and functionalities. The main purpose of an SSP is to accommodate more interoperable IoT platforms, so from a functional point of view it is a sort of container that act as a liaison from the cloud to the local components.

In the following, we use the acronym SSP to denote a Smart Space compliant with the symbIoTe architecture and requirements. Other non-capitalized uses of "smart space" are to be intended as generic.

2.1 Purpose and scope of the document

The aim of Deliverable 4.1 is to describe the architecture of the Smart Space and Smart Device domains, showing the different perspectives in designing the mechanisms already considered for the Application Domain and Cloud Domain, and analyzing how we can guarantee the same features also at a lower level. This will be true for interoperability of platforms, authentication and authorization, access to resources, as well as for additional features such as local discovery and objects roaming.

The document reports on the technical work performed in two tasks, T4.1 (Local Registration, Discovery and Interoperability of Smart Objects, M7–M15) and T4.2 (Wireless network virtualization and management, M9–M26), which are currently still in progress.

2.2 Relation to other tasks and deliverables

The content of this deliverable is related to the outcome of tasks T1.3 “System requirements” and T1.4 “System architecture” which has been published in D1.2 “Initial Report on System Requirements and Architecture”. Certain components of the symbIoTe SSP and SDEV domains are reused from symbIoTe Core Services, presented in deliverable D2.2 “symbIoTe Virtual IoT Environment Implementation”.

On the other hand, this document will serve as a basis for the revised version of requirements and architecture, which will be presented in deliverable D1.4 “Final Report on System Requirements and Architecture”.

2.3 Deliverable Outline

This document is structured as follows: Section 1 is the executive summary and Section 2 the introduction; an initial description of objective and scopes of SSP and SDEV Domains is provided in Section 3, where challenges and solutions are highlighted. Section 4 describes a complete overview of the architecture, both in general and with a one-by-one component decomposition. Subsequent paragraphs explain relationships with Core and Cloud Domains, as well as Security issues related to this low level design.

Section 5 presents the interactions of the different components designed for SSP and SDEV; Section 6 finalizes the document, reporting its conclusions.

3 Objectives and scope

3.1 Smart Spaces

SSPs are environments (residence, campus, vessel, …) where one or more IoT platforms provide coordinated services. Such environments are typically related to a physical space (e.g. a smart residence space is bound to a house or building), but in the more general case we can extend the concept of SSP to a broader physical space (e.g. a smart campus or smart city). We explicitly exclude from our notion of SSP the concept of a geographically dispersed IoT architecture, e.g. we do not consider an SSP the "space" of a fleet of trucks controlled by an IoT platform. There is a somehow arbitrary blurred line in this distinction: for example, we can consider weather sensors in a city controlled by a single platform as a "space" in the SSP sense or they can form multiple SSPs.

The main technical objectives that should be pursued in the design and implementation of the SSP are:

- Dynamic discovery and automatic configuration of resources
- IoT platform interoperability at the SSP level (platform transparency)
- Baseline operation even when the cloud components are not reachable
- Support for nomadic devices

An SSP comprises both virtual and physical objects, such as gateways and smart objects.

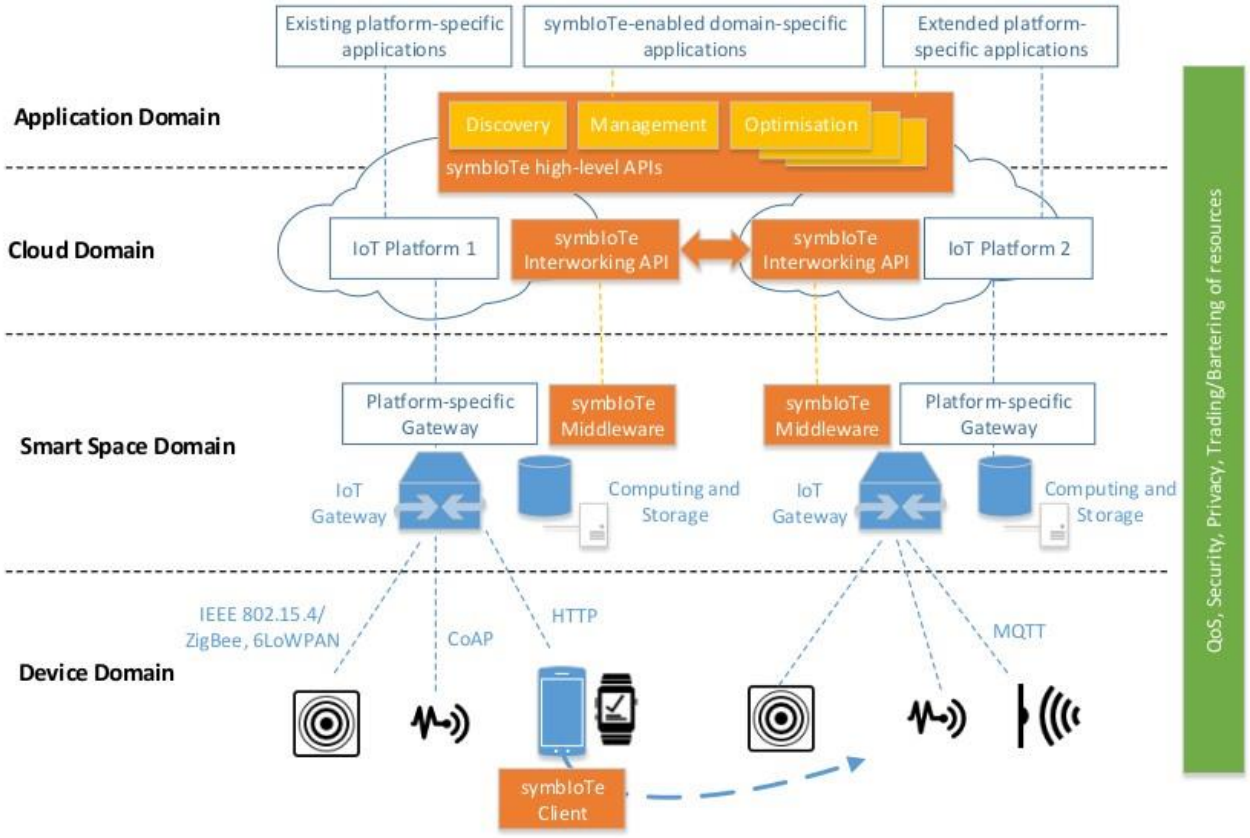

Figure 1 – symbIoTe domains

To cooperate in a system like symbIoTe, the SSP needs to include a series of modules to enable operations related to authentication, authorization, discovery and registration of smart objects. Because of the tight correlation between some specific low-level functions and the physical hardware, features like dynamic discovery and automatic configuration (see Section 3.3, req. 54) depend on the particular technology being used. For example, some wireless technologies natively support roaming devices at the protocol level, whereas others do not support it because, for example, they require human interaction (such as pressing a button) for device association.

Moreover, some technologies enable the possibility to use alternative gateway (GW) as an alternative fallback (see Section 3.3, req. 58): network topologies (typically star topology) that include multiple “border router” are an example of networks that don’t suffer from a single point of failure. In this situation, even if there’s a problem with one border router, the system remains online due to the use of an alternative fallback point. Examples of these technologies are:

- LoRaWAN, where a single device is heard by multiple gateways
- Thread, where the system can reconfigure itself by changing the role of the device in the network and reconfigure a device to act as a border router

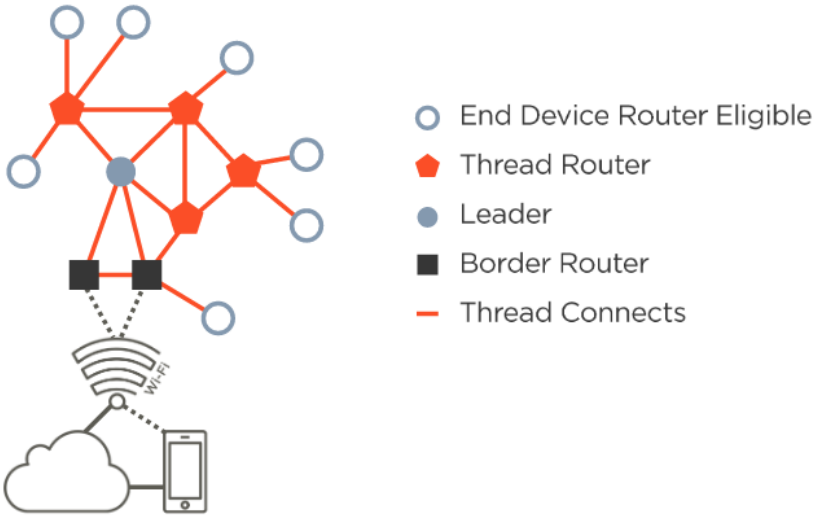

Figure 2 – Thread [22] network topology

The above image (Figure 2) is a schematically representation of a Thread network topology: devices act with different roles and can dynamically change their behavior if the network allows it.

We expect a wide variety of commercial sensors to be “compatible” with symbIoTe’s SSPs (see Section 3.3, req. 57). The architecture of an SSP shall thus need to include different physical level gateways and software components to connect with L2 APIs through a custom symbIoTe gateway: the interaction between the platform and the SSP could pass through CLD level (L2) using the cloud platform APIs.

These modules have a well-known digital interface to interact with each other and should support a pluggable structure to allow an easier deployment on IoT platform. The idea of the smart space is to enable access to the symbIoTe ecosystem at a low level. Such an approach gives registered users access to local resources, even if no internet connection is available, if this resources have already been registered in the symbIoTe ecosystem (see Section 3.3, req. 55, 59).

Because of the need to interact with different software and data architectures, symbIoTe middleware components must be able to manage data model adaptation, exchange of information with the local IoT platforms regarding current associated devices, as well as regarding devices leaving or requesting to join the local space. This kind of behaviour should be asynchronous.

We envision the possibility to configure some kind of device as nomadic: this means that they can move across different SSP but keeping their “identity” in the SymbIoTe ecosystem. Once a nomadic device is registered in the SymbIoTe ecosystem, it can be recognized as the same device also if it moves in different SSP.

For the dynamic discovery, the middleware uses a specific module (Resource Scanner, described in the architecture section of this document) but it needs to use specific technology to work properly: using, for example, a well-known Wi-Fi-SymbIoTe SSID, to which SDEV connect and perform registration operations or maybe a well-known LoRa network, or a well-known symbIoTe-BLE beacon. This kind of automatic discovery isn’t

possible, for example, using technologies like Z-Wave, because, in order to associate, it needs to press the association button on the device.

Similar reasoning for nomadic devices: this behavior is possible if the technology also supports it. For example, in the case of Wi-Fi networks, an ESSID is useful to support this kind of behavior; also for the LoRaWAN technology, because this is actually automatic supported in the star topology of the network.

3.1.1 Dynamic adaption to changing topology

Smart spaces as introduced in the previous chapters need to provide a high degree of flexibility when it comes to: introductions of new sensors to an existing smart space; changes the topology of the smart space when the properties of the smart space change. We can identify three different changes to the smart space topology which make it necessary for the middleware to act accordingly: first, the topology may vary on a physical and/or network-layer with devices finding different options for creating or participating in networks or taking advantage of network features available in a given smart space as described in chapter 3.2. Second, devices in smart spaces may require (but not necessarily need) access to cloud-based services outside of the smart space, especially when it comes to communicate with the symbIoTe core. In this scenario, when outside services are not available due to a lack of internet connectivity, the SSP must react accordingly to this change in topology by re-routing service requests and offering locally available services instead. Thus, a smart space is able to temporarily act as a surrogate for the symbIoTe core layer in the cloud when necessary. Finally, the topology of a smart space might not change itself, but symbIoTe-enabled (smart) devices might find themselves travelling from one smart space to another with a different topology expecting similar functionality there. The following (simplified) scenario illustrates these changes to the topology:

An elderly person is using a smart phone to track weight readings from a scale at home, storing them in a personal health. The scale and the smart phone app are exchanging readings automatically (based on proximity). Both devices are part of a smart space, S1, and can be considered as smart devices as described in chapter 4.3 as they do not rely on an IoT-platform in the background. The elderly person is now ordered by a physician to spend two weeks in a health resort. This health resort also provides a smart space, S2, with various smart devices, including a smart scale. This (symbIoTe-enabled) smart scale uses a different underlying physical means of data transport. After entering the smart space, the person's smart phone is now automatically registered with smart space S2. The smart space middleware now enables the smart phone application to access the weight readings in the same manner as in S1, though the underlying topology might be different.

3.1.2 IoT platform interoperability at the SSP level

One of symbIoTe's value offerings consists of providing a common information model with which to access and use the resources registered in the ecosystem. This capability makes the lives of developers much easier, not having to worry about adapting to new, different, formats whenever some new device or platform is introduced into the ecosystem.

This situation is expected to continue to exist in the Smart Space domain. Services and applications within a given Smart Space shall be able to access its resources and

platforms transparently, meaning that when an application searches, accesses and uses certain symbIoTe-compliant resources within the Smart Space, these will be available and usable under the same information model, regardless of their origin. Certainly, the service must be authorized to access this resources.

As such, it is necessary that all resources use the symbIoTe information model, so they can be readily usable by applications, without any need of knowing about their origins and specifications. Additionally, this interoperability will allow applications to understand the sensors they are dealing with, namely about which information they are measuring or the units of the measurement, enabling the seamless use of cross-domain resources that share the same features. This consistent usage of the resources should exist whether they are being accessed within the Smart Space or when external applications manage them.

In the following scenario, based on the Smart Residence Use Case, it is possible to understand the concept of interoperability at the SSP level.

Scenario:

A residence contains a Home Automation Service. This service can control the brightness and temperature of the interior of the house. For such tasks, it needs to access information from brightness and temperature sensors and, additionally, be able to control actuators such as fan coils, lights or automated curtains.

These sensors and actuators can be made available either by IoT Platforms inserted in the SSP or by Smart Devices within the SSP. This is transparent for the Home Automation Service, which can access the resources through the SSP without needing to know about their origins. As such, the service can handle the resources in the same manner, either if they all belong to the same IoT Platform, or to different IoT Platforms in the SSP or even if they are symbIoTe-compliant resources obtained independently by the resident.

Since the devices use the symbIoTe information model, the platform can easily understand what type of sensors it is dealing with, what kind of units they operate with and their location. As such, it can quickly and dynamically use devices in the SSP, easily adapting to new resources that enter it.

For this scenario to work, the SSP needs a service to keep track of which resources are currently within the SSP and a Resource Access Proxy (RAP) to access the resources. Additionally, the resources should follow the symbIoTe information model.

3.1.3 Cloud dependency

The Cloud Domain in the symbIoTe architecture provides to third parties unified and secure access to platform resources which have been registered within the symbIoTe core services. It also allows the formation of platform federations, where collaborating platforms can easily exchange information and perform bartering and trading actions (see Section 3.3, req. 59).

These functionalities are very relevant to the many symbIoTe Use Cases, becoming fundamental when dealing with cross-domain applications. However, there are certain situations where it would be very beneficial, or even required, to work without access to the cloud. While, for example, applications in the Smart Cities domain are very dependent on the cloud, other situations, such as Smart Residence applications, operating locally

(within a residence), could run without reaching the cloud. Access to the brightness readings of sensors installed in a residence and the ability to handle its lamps should not be dependent on Internet access.

There are several advantages not to be reliant on the symbIoTe's cloud. First of all, and the most prominent one, is not being dependent on internet connectivity. In some areas, it can be unreliable and, even if it isn't, given the local context, it doesn't make sense to depend on it. Secondly, security will be improved. By running the system locally, security breaches such as man-in-the-middle attacks can be easily avoided. Finally, network performance can be improved, by bringing the middleware closer to the devices. This results in fewer network hops, improving efficiency (see Section 3.3, req. 59-63, 66).

Software to implement these functionalities developed for the Cloud Domain can be reused or adapted to handle these kinds of requests in the Smart Space Domain with no need for outside communications. This would improve the speed of the development of these components.

As such, it is envisioned for the Smart Space Domain to function normally when no internet connectivity is available (assuming the resources can communicate with the middleware internally).

The following scenario, reusing the Smart Residence Use Case as in the previous section, will help envision the use of symbIoTe-enabled platforms and devices without being connected to the symbIoTe Cloud.

Scenario:

A residence uses a Home Automation Service to keep the home of the users in their preferred conditions. It does so by controlling and adjusting the levels of brightness and temperature in the rooms where the residents are present.

Due to some problems of the Internet Service Provider, the residence won't be connected to the internet for a while. With the help of the Smart Space the service operates in, it is capable of continuing to function and to access sensor data and actuators.

Local components, based on the components developed for the cloud, manage tasks such as looking for and registering new devices in the SS, keeping track of what devices are currently on the SS, access local resources and authentication and authorization of applications so that they can access the SS.

With components to handle these tasks locally, the service can continue to function normally and securely with no connection to the cloud.

3.1.4 Architectural mapping to existing IoT platforms

As clarified in the previous paragraph, from a general point of view for a "closed" SSP, such as a residence or a yacht, being independent of cloud-based components makes perfect sense and in certain cases it can even be seen as a requirement. Resource access must be guaranteed and control must be feasible with a very low latency (e.g. for switching on a light or opening a door lock). On the other hand, if we consider SSPs in a wider sense, for example a stadium or campus, this requirement becomes less essential.

A more general understanding of the dependency of a given IoT platform from one or more cloud-based functions is also very important to design its integration with symbIoTe

and, thus, to design an SSP architecture flexible enough to accommodate for different types of platforms.

In principle, IoT platforms can be split in three different categories (Figure 3):

A. Fully local platforms
B. Hybrid local / cloud based platforms
C. Fully cloud-based platforms

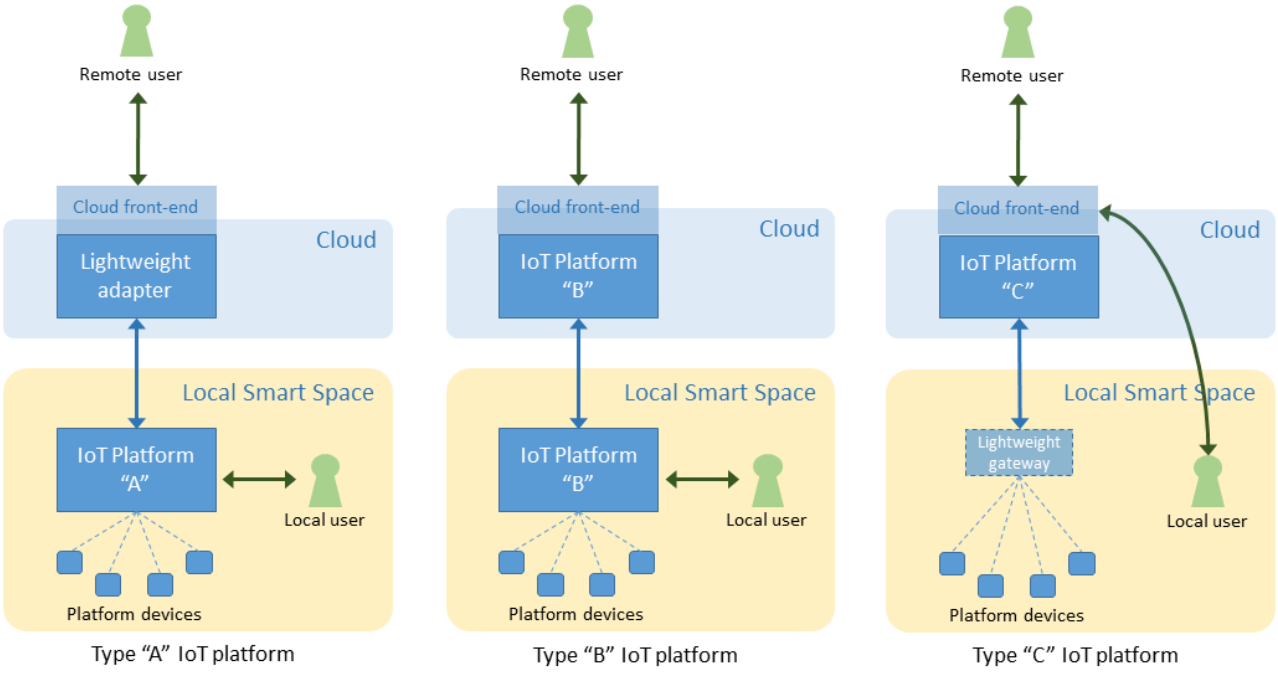

Figure 3 – Different categories of IoT platforms

In this categorization, with "local" we identify a network space which is under reach by a controller (e.g. a smart phone app or PC-based desktop application) in the same LAN or routing domain where the platform components and resources are located. It could be for example the LAN of a home, or even multiple LANs in adjacent buildings connected via routers in the buildings themselves. The distinction is blurred, of course, since it may be possible to consider some resources to be "more local" than others because they are in the same building, on the same floor, or maybe attached to the same Wi-Fi access point. Hence the categorization must be always casted to the specific IoT platform being analyzed.

Fully local platforms (type "A") are completely isolated environments, where devices and resources are managed by local hardware and software components. Users located in the same environment can access the resources by interacting directly with those components, with minimum latency and very limited risk of unavailability. Remote users do not necessarily have the possibility to access the resources within the environment: in order for this to be possible, either the platform must be publicly addressable from the Internet, or a minimum platform counterpart ("lightweight adapter" in the picture) must be deployed in the cloud to provide such an access point. This adapter contains no platform specific functions, and is barely a sort of tunnel end-point to allow entering the smart space.

Hybrid platforms (type “B”), instead, have a more substantial cloud counterpart, where part of the platform’s functions is deployed. Functions that are provided by local modules and functions that are delegated to cloud-based ones will be different, according to the specific IoT platform architecture. For example, management of the platform’s resources might be provided by cloud modules, whereas control might be done by local ones; data storage might be done locally and synchronized to the cloud, whereas aggregated analytics might be done in the cloud; and so on. Local users connect either to on premise or to cloud-based modules, depending on where the functions they need to use are located; remote users always connect to cloud-based modules.

Finally, type “C” platforms have all of their functions provided by cloud-based software modules. Physical resources are controlled directly (in case they are IP-reachable) or by means of an on premise gateway (“lightweight gateway” in the picture). The role of the on premise gateway in this case is dual with respect to the “lightweight adapter” for type A platforms: this component can be reduced down to a simple tunnel end-point to allow for the remote modules to communicate with the local devices.

The three types of platforms can be mapped to the symbIoTe architecture by properly positioning the symbIoTe interworking interface:

- for type A platforms, all the information related to the platform is known only by modules located in the same smart space where the IoT platform resides. So, that’s where the interworking interface will need to be placed;
- for type B platforms, where information is split across local and cloud modules, the interworking interface will need to be split as well. For example, the RAP might be in the cloud, while the Registration Handler might be local. This depends completely on the architecture of the IoT platform;
- for type C platforms, of course, the interworking interface will be located in the cloud.

Some examples of commercial IoT platforms belonging to the three categories include:

- Type A: Samsung SmartThings [7], AllSeen / Open Connectivity Foundation [8], Philips Hue [9] ecosystem (as long as we want to consider it an IoT platform), Zipato Home Automation [10], Symphony [11] by Nextworks (when in standalone mode)
- Type B: Symphony [11] by Nextworks (when deployed with cloud services), Vera Smart Home [12]
- Type C: Netatmo Connect [13] platform, products built with Microsoft Azure IoT Suite, most vehicular fleet tracking systems, Koubachi [14] plant monitoring system (currently discontinued)

Of course, it is important to consider that the distinction is not always unambiguous.

3.2 Smart Devices

Most IoT devices are wirelessly connected to the Internet and possibly nomadic from one service platform to another. In the symbIoTe vision, a Smart IoT Device is a device in which a symbIoTe software component is installed in order to interact with a Smart Space. The interaction with the Smart Space is characterized by the physical space in which the

Smart Device is located. Indeed, in such a physical space, the wireless medium and the technologies providing local connectivity represent some resources that need to be shared, in a coordinated manner, by the IoT platforms and the Smart Devices that are in radio visibility. Moreover, the device contains additional resources, including sensing and actuating capabilities, applications, storage, etc., which may generally depend on the device type and can be accessed by exposing a symbIoTe-compliant API.

In order to define the architecture of this symbIoTe software component, we considered two main features for Smart Devices:

- *attaching to a smart space*, by configuring an authorized physical link to a selected gateway, in order to make available the device resources and participate to the service creation in the smart space.
- *roaming across heterogeneous platforms,* which are usually managed by heterogeneous providers.

To support these functions, several technical challenges related to wireless connectivity have to be considered in the design of this software component. Moreover, other functionality to be designed could involve the possibility to make it able to discover environmental resources, in terms of platforms and relevant network infrastructures, as well as other Smart Devices present in the Smart Space.

3.2.1 Objects roaming

The roaming concept for a Smart Device includes different planes:

i) a more traditional connectivity plane, for roaming from a network infrastructure to another, based on the same technology, but managed by different operators;

ii) a more advanced multi-technology connectivity plane, for roaming from a network infrastructure based on a given technology to a new one exploiting a different technology and/or for seamless aggregating each available wireless link;

iii) a service plane, for roaming across independent IoT platforms, that have been designed for completely different services; for example, a smart plug could roam from a home automation platform to an electric load monitor platform, while keeping the wireless connectivity to the same physical gateway.

As far as concerns the first aspect, it is easy to adapt existing solutions to our architecture. For traditional wireless devices, roaming means enabling the attachment to a network infrastructure different from home network and managed by a different network provider. For example, for a cellular phone visiting a foreigner country, roaming allows to receive/start phone calls even in geographic areas in which the home network operator does not offer connectivity. This is achieved by delegating to the home network the device authentication and by means of agreements (SLA) among network operators. Similar concepts have been implemented for enterprise networks that are not managed by a network operator. For example, Eduroam [21] has been designed to enable roaming across Wi-Fi University campuses and is a de-facto standard for forwarding authentication requests, in a scalable manner, by means of a hierarchy of RADIUS servers.

Multi-radio access technologies (multi-RAT) have also received large attention in recent literature for advanced devices equipped with multiple interfaces. However, solutions for

managing multiple wireless interfaces are often limited to priority-based switching among interfaces. For example, most smartphones exploit Wi-Fi networks when available and switch to 4G networks in absence of Wi-Fi coverage. In some other cases, the interface selection is triggered by the application type (e.g. switch to Wi-Fi to synchronize a file) or explicit user or network intervention, but it never happens that multiple interfaces are used simultaneously. Finally, roaming across IoT service platforms involve a logical registration of the smart device, which can be completely decoupled by the attachment to the physical gateway.

3.2.2 Level3 vs. Level4 compliance

In the context of symbIoTe, a “Smart Device” (SDEV) has no relation to a specific IoT Platform. It only relies on the functions provided by the symbIoTe middleware in any SSP in order to be accessible from the symbIoTe Core and, ultimately, by any symbIoTe app. As a consequence, the SDEV is required to expose an API to access the resources it contains: basically, the SDEV should include its own equivalent of a RAP.

As regards L3 vs. L4 compliance, the distinguishing features of the two levels are dynamic configuration (L3) and roaming (L4). Hence, an SDEV 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 SDEV 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 (see Section 3.3, req. 64). In other words:

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

An L3 SDEV (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 SDEV will always keep the same URL.

In this sense, we can distinguish between *roaming* of L4 compliant SDEVs and *migration* of L3 compliant SDEVs which receive a new identity from the hosting SSP.

An interesting capability of L4 SDEVs could be the exploitation of multiple network interfaces providing Internet access, integrating heterogeneous wireless technologies such as 4G and Wi-Fi. Indeed, coordination among heterogeneous networks is explicitly considered in the emerging 5G systems and even in current 4G systems, where traffic offloading from cellular networks to local Wi-Fi networks appears a promising opportunity

for increasing system capacity. However, at the device level, solutions for managing multiple wireless interfaces are often limited to priority-based switching among interfaces, such as smartphones using Wi-Fi networks when available and switching to cellular otherwise. In some other cases, the interface selection is triggered by the application type (e.g. switch to Wi-Fi to synchronize Dropbox) or explicit user or network intervention, but it never happens that multiple interfaces are used simultaneously in order to achieve an inter-technology and/or inter-operator bandwidth aggregation.

In symbIoTe, multiple homogeneous and heterogeneous interfaces of SDEVs could be used to aggregate the bandwidth available through each interface, by addressing the following main issues:

i. identifying the most promising technical solutions to utilise simultaneously the multiple interfaces;

ii. designing a scheduler to allocate data and/or control packets to each interface by taking into account the heterogeneity of network performance, in terms of reliability, throughput and latencies.

Apart from bandwidth aggregation, we can envision other strategies to exploit the availability of independent links with heterogeneous behaviours. For example, the best technology can be used to transport data, while the additional links can be used to carry control information, enabling some forms of optimizations. We plan to experimentally benchmark solutions for optimizing the Wi-Fi network performance by exploiting cellular links. We expect that such an approach can be relevant in the emerging scenarios of high-density deployments of Wi-Fi networks, for which reducing the contention level on Wi-Fi networks (e.g. by shifting TCP ACKs on the cellular links), or optimizing the planning of Wi-Fi channels on the basis of mobile terminals' feedback can bring significant benefits.

3.2.3 LoRaWAN Technology

LoRa™ [16] is a radio physical technology designed to be ultra low power and robust to interferences. It uses spread spectrum modulation to minimize power consumption and to add robustness: this means that it uses a wider band to transmit the signal. This way, the modulation is able to associate more energy to the transmitted signal.

Based on top of the physical layer, the LoRa™-Alliance [16] defines a transport protocol named LoRaWAN™ [17] (LoRa™ for Wide Area Network) and, thanks to this specification, the system can handle wide IoT scenarios. A typical LoRaWAN™ architecture is shown in Figure 4.

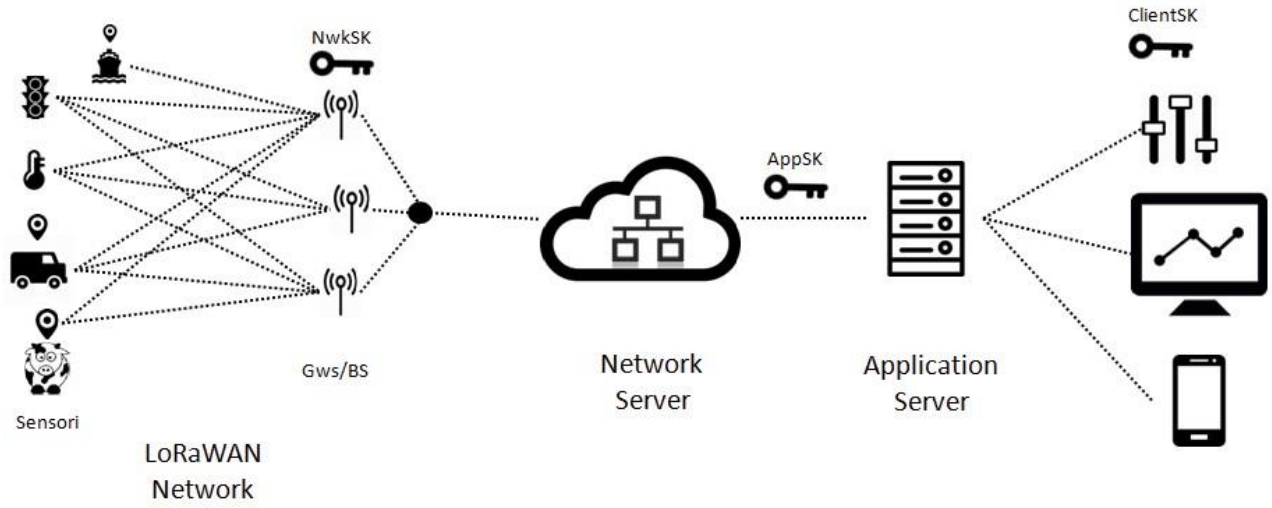

Figure 4 – LoRaWAN™ architecture

The network topology is a double star, where the gateway can receive messages from multiple devices and devices can be heard from multiple gateways. Every gateway covers a wide area, which depends on the density of the area, but can be approximately of 2 km in urban areas and 10-15 km in rural ones. The gateway network is managed by a specific server, called Network Server (NS), that takes decision about which is the elected gateway, from the whole that heard the device packet, that actually manages the device. The NS and thus the gateways have the possibility to additionally manage the data rate of the devices: LoRa™ is a technology that takes benefit from a well-coverage network area. The more the device is close to the gateway, the less power it uses for transmission and it can use a higher data rate to transmit the packet.

Data coming from devices is sent to the proper application server by the NS and then exposed by services to the user. Both data paths are encrypted with keys. The first one, from device to network, is encrypted with the network session key (NwkSKey), the second one, from NS to application server, with the application session key (AppSKey). Application server is where the client's service is running.

LoRaWAN defines 3 major classes of operation for devices:

- **Class A**: every LoRaWAN compliant gateway should implement at least class A compatibility. This is the simplest and more power efficient class from the device point of view. Class A simply means that the behavior of the device is to sleep for almost all of its time. When it decides to send a packet, it wakes up and it sends the packet. After that, it waits for a well-known delay and then open a first receiving window. If it senses no radio preamble, meaning an upcoming downlink packet from NS, it waits another well-known delay and it opens a second receive window. If no preamble is detected, the device conclude that no downlink frame is available from NS and it goes back to sleep. In class A, if a downlink message should be sent to the device, the application has to wait for the awakening of the device.
- **Class B**: in this class, the device grants some extra downlink slots at prefixed times to give applications better device reliability. The device synchronizes its timing with a periodic beacon signal sent from gateway. Beacon signal is sent every 128

seconds. In this time slot, the devices choose at least one of the 4096 ping slot available to be reachable from NS if a downlink message needs to be performed. Obviously, the device should inform NS of which ping slot uses for reachability.

- **Class C**: the device simply is always listening. This is the most reliable class but also the most power consuming ones.

3.3 System requirements

The specification of system requirements for SSP and SDEV is directly derived from the objectives and the scope of these domains, basing on an accurate assessment of the symbIoTe architecture, in the context of the identified goals (see Section 3.1 and 3.2).

Index	Domain	Type	Category	Importance	Description
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
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
56	Smart Space	Functional	Management	SHOULD	The system SHOULD be able to prioritize the information to send in the platform (i.e. important information first)
57	Smart Space	Non-Functional	Interface	SHOULD	The system SHOULD support the dynamic configuration of a subset of commercial sensors
58	Smart Space	Functional	Interface	MAY	Inside Smart Space multiple gateways MAY be used as an alternative fall-back router for a given device

59	Smart Space	Functional	Management	SHOULD	SymbIoTe smart spaces SHOULD be able to operate without a permanent Internet connection
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)
61	Smart Space	Functional	Management / Interface	SHOULD	Different collocated IoT platforms SHOULD be able to interact locally with mediation from symbIoTe cloud components
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
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
64	Smart Space	Functional	Management	MUST	An L4-compliant Smart Device MUST have a globally unique identifier
65	Smart Space, Smart Device	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
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
67	Smart Space, Smart Device	Functional	Management/ Interface	MUST	SymbIoTe MUST accept visiting devices to be merged in the visited Smart Space

Table 1 – System requirements

In Table 1 are listed the requirements related to the Smart Space and Smart Device domains. These requirements, that can be either Functional or Non-Functional, concern to the interface and the management areas and their importance is indicated as a MUST (shall), SHOULD (recommended) or MAY (optional), same as indicated for other domains.

The purpose of the list is to specify a set of properties that a Smart Space needs to support, in order to permit a correct mode of operation inside of this domain. The Smart Device domain, which is related to roaming devices, specifies a set of requirements concerning the device identification and their tracking when they pass from a Smart Space to another.

4 Smart Space and Smart Device

4.1 Smart Space Architecture

In order for these environments to be turned into SSPs and integrated in symbIoTe, we need to deploy proper software adapters (that we generically refer to as symbIoTe Smart Space Middleware, or S3M).

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 S3M 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 SSP 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, SDEVs associated to the SSP will also be exposed directly, that is without being "mediated" by any of the local platforms. symbIoTe-compliant local IoT platforms within an SSP will be able to access all the resources associated to that SSP, provided that the required authentication and authorization policies are in place. This includes both resources provided by any co-located IoT platforms, and those provided by the locally associated SDEVs. This way, the interoperability role played by symbIoTe will be fully functional also at the SSP level. When more than one IoT platform is active in an SSP, the S3M shall thus be acting as a local resource interchange.

Since each IoT Platform manages its own devices according to its internal protocols (which shall remain opaque for symbIoTe), discovery and auto-configuration of those devices will be in charge of the Platform itself. It will be the Platform's duty to register and de-register its devices with the S3M whenever a change happens. The S3M will then publish information about the Platform's devices in the same way it does for symbIoTe's SDEVs. Hence, SDEVs will not be distinguishable from the devices of a native platform.

This provides a second procedure for resources to get registered with the symbIoTe Core, with the advantage of providing a transparent extension for SDEVs.

A proposed approach for resource registration is described in Section 0.

4.1.1.1 Smart Spaces 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.

4.2 SymbIoTe middleware

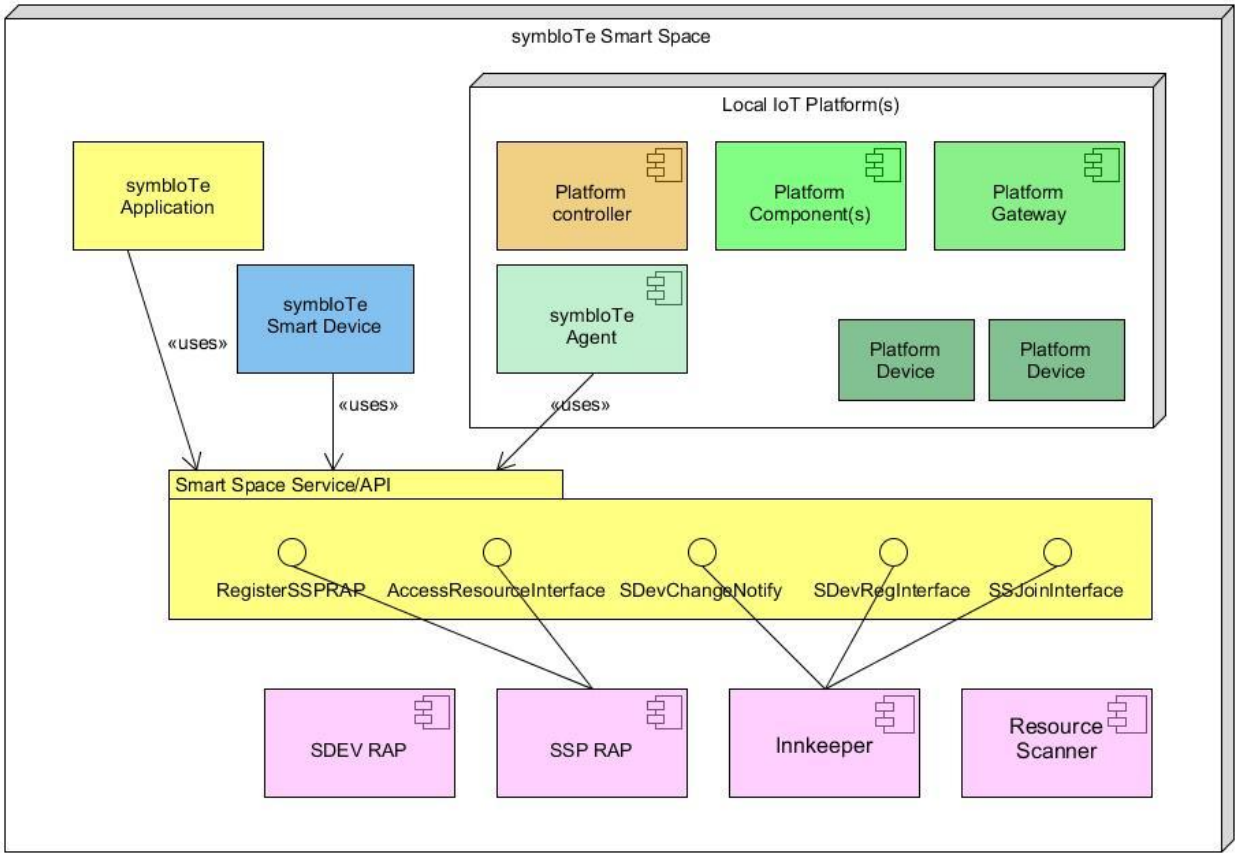

Figure 5 – Smart Space architecture

The S3M is a set of dedicated components required for the rest of the symbIoTe world to interface with the Smart Space. It provides system features and functionalities to the local IoT platforms in the SSP, and allows bidirectional access between the symbIoTe Core and the resources.

The S3M can run either on a dedicated hardware device, or as a software component in one of the local IoT platforms' servers. In the former case, the symbIoTe gateway will be based on UNIDATA's Spine board [18]; otherwise it will be provided as a software component or container to be deployed on a third-party server (e.g. the S3M could be an optional plug-in for the IoT platform's server).

Due to the nature of the interface of a Smart Space for the outside world, it is assumed that the S3M will be deployed in each SSP as a singleton, in order to be a single access point to the IoT platforms residing into that Smart Space.

Since the S3M is unique in an SSP, consequentially, it can handle an arbitrary number of IoT platforms, comprised from zero to N. The N-case is the normal S3M situation, where symbIoTe handles some IoT platforms registered into the space, making their resources addressable from the symbIoTe environment. The zero-case instead, arises as an interesting borderline case, where the SSP is allowed to have no local IoT platforms at all and to have just a symbIoTe gateway running the S3M. In this situation, such a "degraded" SSP would allow roaming and nomadic SDEVs to attach to an SSP (thus becoming available to the symbIoTe Core), even if there aren't any IoT platform registered in the same SSP.

In the Smart Space architecture described in Figure 5, it is also indicated a platform internal component, mentioned as Platform controller, which is in charge of forwarding the resource access requests coming from the SSP RAP to the platform devices.

4.2.1 Innkeeper

Component	Innkeeper
symbIoTe Domain	SSP
Description	It is in charge of receiving local requests from L1-compliant apps and L3/L4-compliant devices. Note: Innkeeper is required also for L1 compliance, nevertheless it is specified in WP4 because it resides in the Smart Space.
Provided functionalities	• it allows an app to understand that it has entered a symbIoTe compliant SSP. • it allows a new device to register itself to the SSP. • it keeps a register of locally registered apps and SDEVs. • it maps global URIs to local URIs
Relation to other components	Local RAP Registration Handler
Related use cases	Smart Residence, Smart Yachting

4.2.2 Resource Scanner

Component	Resource scanner
symbIoTe Domain	SSP
Description	It is the network scanner for well-known devices and device gateways. (E.g. Philips Hue, Z-Wave devices/gateways, Smart TVs, storage systems, UPnP devices)
Provided functionalities	• it scans the local network for device/gateway, attaching the device to the symbIoTe middleware
Relation to other components	• Registration Handler
Related use cases	• Smart Residence

4.2.3 SSP RAP

Component	SSP RAP
symbIoTe Domain	SSP
Description	It is the local access point for symbIoTe applications running in the Smart Space. It is composed of a generic software part, common for every platform, which is the interface with the rest of the symbIoTe software, and of a platform plugin part, which is the layer that interacts with the platform controller.
Provided functionalities	• It allows direct access to the Smart Space's resources without going through the cloud components • Optionally, it can also act as a network controller, coordinating access to the wireless medium. Indeed, wireless access can be seen as a common resource to be shared among platforms. Thus, the RAP can act as a mediator for the "virtualization" of channel resources and the optimization of coexisting wireless technologies.
Relation to other components	• Innkeeper

Related use cases	• Smart Residence

4.2.4 SDEV RAP

Component	SDEV RAP
symbIoTe Domain	SSP
Description	Gateway to allow access to local SDEVs for clients outside the SSP
Provided functionalities	• it provides a resource access point for SDEVs to be addressable from outside the SSP • it provides a local RAP to access SDs from within the SS
Relation to other components	• Innkeeper • symbIoTe Client
Related use cases	• Smart Residence • Smart Yachting

4.2.5 Local AAM

Component	Local AAM
symbIoTe Domain	SSP
Description	Local authentication and authorization component to allow symbIoTe applications and SDEVs access the SSP when no Internet connectivity is available
Provided functionalities	• it allows AAM mechanisms without cloud component connectivity • it implements configurable fallback policies to allow each Smart Space to balance security risks with available functions
Relation to other components	• Innkeeper • symbIoTe Client

Related use cases	• Smart Residence • Smart Yachting

4.2.6 IoT platform's symbIoTe agent

Component	symbIoTe Agent
symbIoTe Domain	SSP / SDEV
Description	It is the access point for a symbIoTe-aware IoT platform to SDEVs registered in the same SSP
Provided functionalities	• it allows a symbIoTe-aware IoT platform to access SDEVs registered in the same SSP. • it allows a platform to expose its own devices to the symbIoTe ecosystem. • it allows a platform to become a "client" (like an app or an enabler) and access foreign resources. Note: we should decide whether it makes sense to limit access only to locally registered SDEVs (direct access through the Local RAP), or to generalize to any symbIoTe resource (full access, but requiring Internet connectivity).
Relation to other components	• Local RAP • Local AAM
Related use cases	• Smart Residence • Smart Yachting

4.3 Smart Device Architecture

In the context of symbIoTe, a "Smart Device" (SDEV) is a device that can directly interact with an SSP, without external components. To this extent, any mobile device (smartphone, tablet) running a proper symbIoTe app can be considered an SDEV. The containing SSP shall allow the SDEV to be accessed from components (apps, enablers) outside the SSP itself, and from components inside it (an app associated to the same SSP). As a consequence, the SDEV is required to expose an API to access the resources it contains: basically, the SDEV should include its own equivalent of a RAP. For simplicity, we might as well assume that the interface is exactly the same of the RAP as defined in L1.

A basic SDEV architecture is shown in the figure below: a symbIoTe agent module is responsible for granting access to the SDEV local resources, which include sensors and/or actuators, storage, network interfaces and services or apps running locally. This module communicates with the symbIoTe SSP RAP to interface with the other symbIoTe modules. The presence of the symbIoTe agent is the minimum requirement for SDEVs to be accessed by symbIoTe compliant applications or platforms.

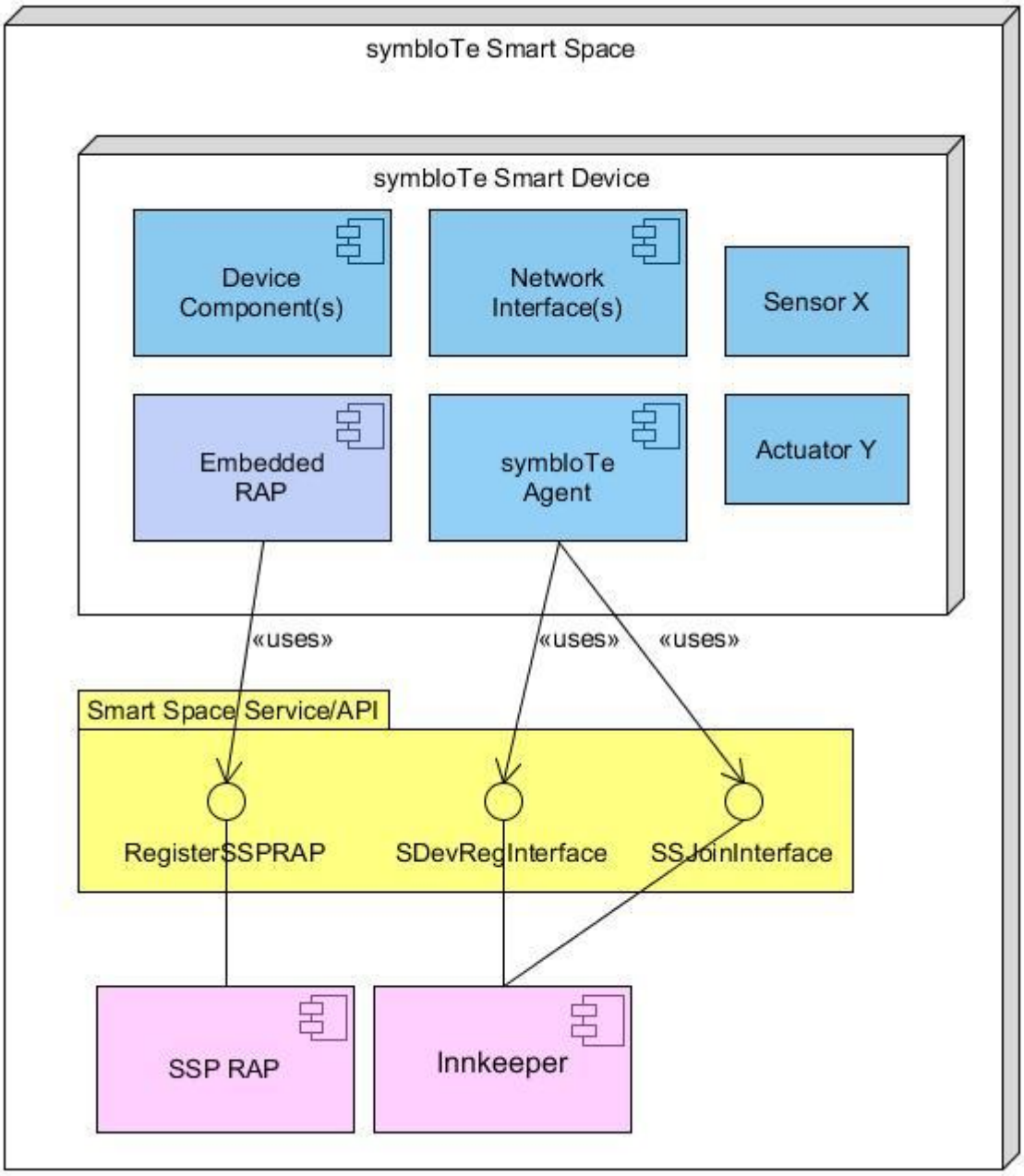

Figure 6 – Smart Device architecture (v1)

This basic architecture (Figure 6) is best suitable for sensors or actuators with limited capabilities in terms of battery, storage, memory or computational capacity. For more powerful SDEVs, such as mobile phones or laptops/computers, it could be desirable to

have a more sophisticated architecture. In this case, the symbIoTe agent could be extended with some of the Smart Space's modules and could even run locally some of the symbIoTe enablers or applications (eventually the local IoT platform itself).

We also envision some additional modules in the symbIoTe agent to specifically deal with the wireless connectivity in both the Smart Devices and the IoT platforms.

4.4 Relationship with Core and Cloud components

Platforms with the intention of enabling symbIoTe Smart Space (L3 Compliance), and with the intention of hosting Smart Devices (L4 Compliance) should also adhere to L1 and L2 Compliance. As a result, resources within a SSP should also be available at symbIoTe Core and symbIoTe Cloud domains. Different configurations of SSP also influence the relationships between SSP components with CLD and APP components. To illustrate these relationships, we need to distinguish between two different cases:

1. symbIoTe application is running within the SSP, and
2. symbIoTe application is running outside the SSP.

IoT platforms wanting to form Smart Spaces do not need to have Cloud components. For such platforms, SSP components should communicate directly with the symbIoTe Core. On the other hand, platforms containing Cloud components could use these components to communicate with symbIoTe Core. These two options will be considered when describing interactions for the two aforementioned cases.

In the first case, an application running within the SSP can have the same local address as the SSP, and all the interactions within the SSP can function even when the connection to the Internet is lost. The interactions between components for this case are shown in Figure 7. The application in that case can access the resources by using the SSP RAP. Furthermore, application can also search for resources by using symbIoTe Core Search Engine. For these resources to be available in symbIoTe Core, the Innkeeper should register them to the symbIoTe Registry. Regarding access control, Local Authentication and Authorization Manager (Local AAM) contacts the Core AAM every time an application joins the Smart Space, and checks whether this L3 application is authorized for such access.

For the case when symbIoTe application is running outside the SSP, shown in Figure 8, the access to the SDEV goes through RAP at CLD and SDEV RAP. SDEV RAP then redirects the request for accessing certain resources to the SSP RAP. If the SSP does not have CLD at its disposal, the outer application can access this SDEV by contacting SDEV RAP directly.

In both cases, irrelevant to the fact whether an application is registered to the Smart Space or not, there exist some common interactions between components in different domains. Innkeeper should update the Core Registry each time after a new device registers to or deregisters from the SSP. If SSP is implemented on a platform containing symbIoTe Cloud components, Registration Handler can also be used for that. On the other hand, Innkeeper should communicate directly with symbIoTe Core. If the Cloud Monitoring component is available, a Monitoring Agent should be deployed within the SSP to collect data for Cloud Monitoring Component. Core Resource Access Monitor (Core RAM) is connected with SSP RAP, and SSP RAP can inform it how symbIoTe-registered

applications are using symbIoTe-registered resources. In the case when RAP in CLD is used for accessing SSP RAP, communication from SSP RAP to Core RAM can go through RAP in CLD. Furthermore, each Application should inform the Core RAM when it starts accessing a certain resource.

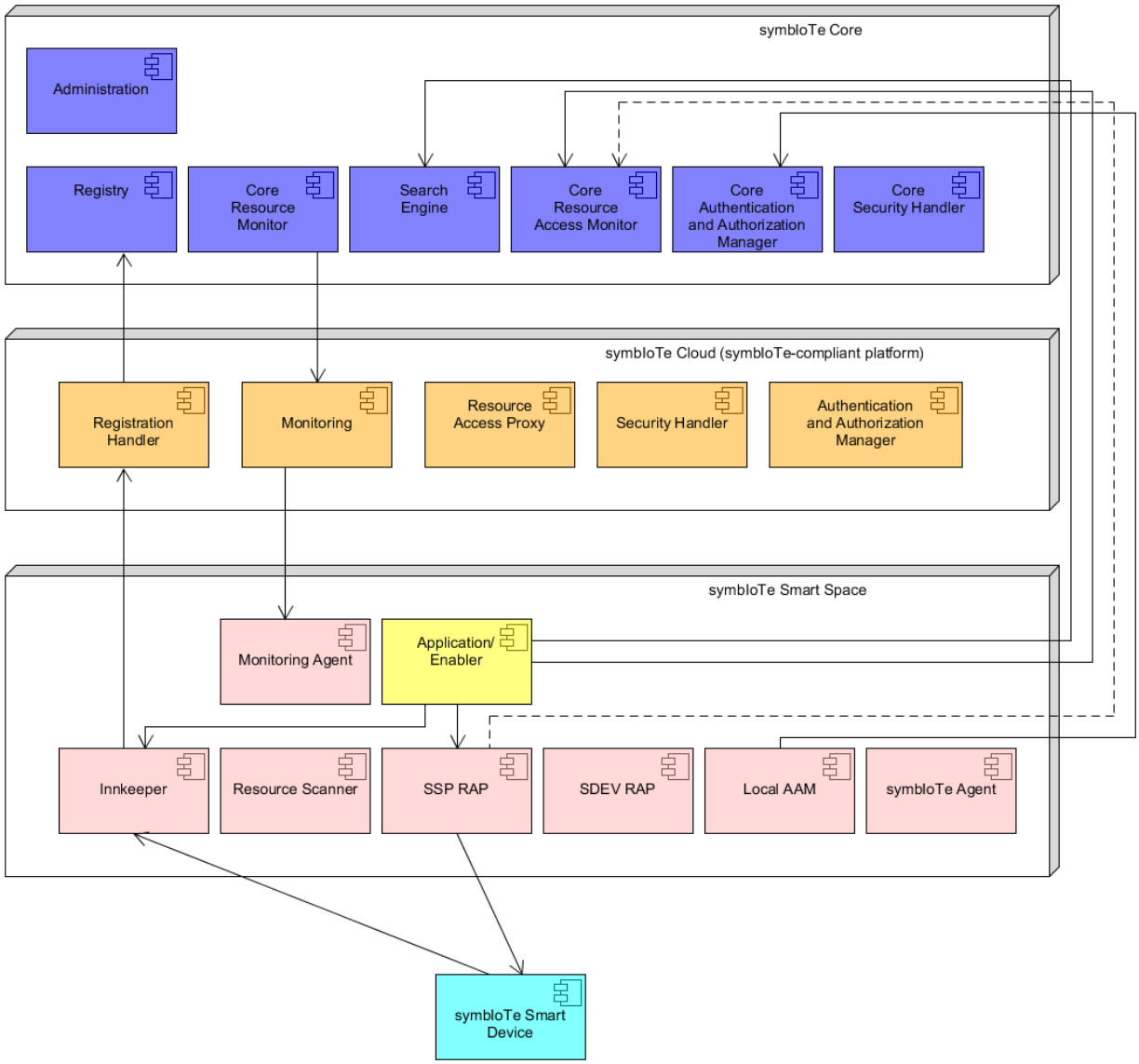

Figure 7 – symbIoTe components from APP, CLD and SSP – Application in the Smart Space

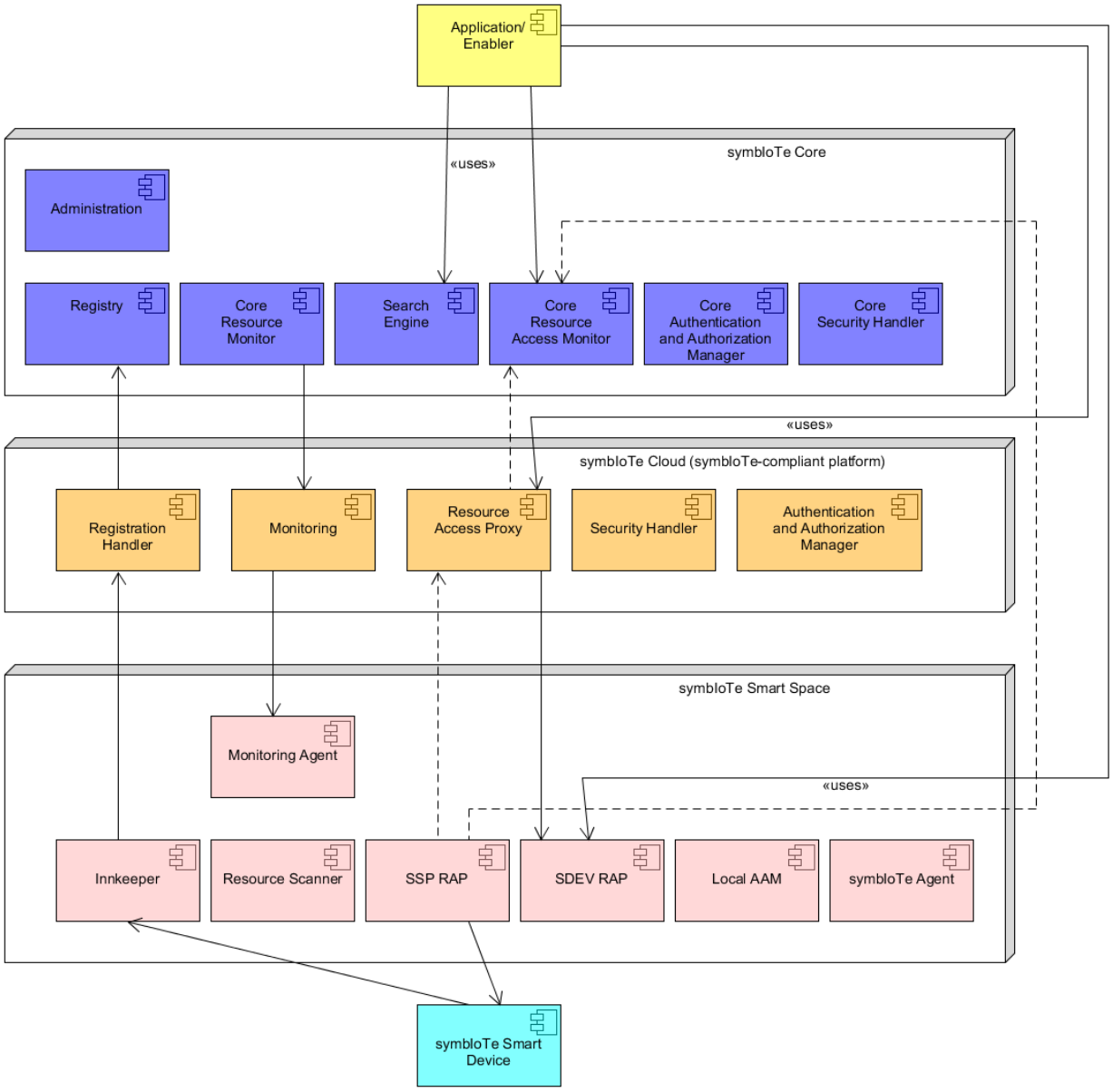

Figure 8 – symbIoTe components from APP, CLD and SSP – Application outside the Smart Space

Note: interactions between components from the same domain, as well as interactions between APP and CLD are not within the scope of this subsection, and therefore only selected interactions are shown in the figures.

4.5 Security considerations

As stated in the previous section, symbIoTe will provide access to the Smart Devices for the symbIoTe applications residing in the Smart Space, as well as for the symbIoTe applications outside of the SSP, over the public internet. Therefore, the enforcement of

the security mechanisms is essential for providing reliable information transfer from applications to the devices in the Smart Space. This requires data anonymity, integrity, and confidentiality, which will be ensured with encrypted data transfer and mutual authentication. Goal of the security mechanisms in the Smart Space is to provide authorized access, preserve privacy of the user information and prevent malicious attacks.

Since symbIoTe components have been designed to be reusable across domains, Smart Space security components and policies are tightly coupled with those in the Application and Cloud domain. That means that in the Smart Space, just as in the Application and Cloud domains, access control and access policy definitions are based on ABAC (Attribute-Based Access Control). An 'attribute' refers to a generic property/role/permission that the application provides during the authentication and authorization phases. In order to be granted access to the smart device, an application has to provide a set of attributes that will be validated against predefined access policies. Main benefit of using ABAC over other access control systems, such as RBAC (Role-Based Access Control) or GBAC (Group-Based Access Control), is the possibility for less troublesome and less error prone definitions of the access policies based on complex logic with multiple attributes.

However, since confidential user information will be sent as attributes, it is vital to ensure privacy protection and not publicly expose any of those attributes. Furthermore, for successful authorization it is required to send additional information, such as who issued the access right, when the right expires, etc. For those reasons, all the information required for authorization will be sent in a token, which has to be properly and uniquely formatted across the whole symbIoTe ecosystem. Partners in the consortium chose JWT (JSON Web Tokens) to be used as the token format in symbIoTe. JWT includes all the required information for access rights validation and enables symmetric and asymmetric token encryption to preserve the privacy of the user and Smart Device related information. Furthermore, JWT enables authentication and authorization even if a smart space is disconnected from the internet, which is defined as one of the requirements of the Smart Space.

Furthermore, in order to prevent man-in-the-middle attacks and identity spoofing, mutual authentication is required in communication between applications, cloud and local platforms, but also in communication between smart devices.

Access control functionalities are decoupled into several components in the Smart Space. Functionality for issuing access token, including required attributes, is provided in Local Authentication and Authorization Manager (Local AAM). Task of the Local AAM is to conduct access rights validation once the required data is provided as input. In case of successful validation, it issues an access token to the symbIoTe application, which will be further used for accessing Smart Devices or other services in the Smart Space. Once the application obtains an access token, it forwards the token while requesting resource access. For resource access, we can distinguish two different cases:

- symbIoTe application within the SSP requires resource access
- symbIoTe application outside the SSP requires resource access

In the first case the symbIoTe application resides in the same local network as the SSP middleware. Additionally, no connection over the Internet is required. In this scenario, the symbIoTe application will try to access the resource over the SSP RAP, which will perform an ABAC check using the information provided in the access token, e.g. attributes, token

expiration time, token issuer, etc. If the information provided in the token is valid and sufficient for successful authorization, access to the Smart Device is granted. In the second case, the symbIoTe application cannot access the resource directly via the Smart Space, but has to use Cloud domain components. That means that an access control check will be performed in the Cloud domain, particularly by the Cloud RAP component, which will redirect the request to the SDEV RAP component, which resides in the Smart Space, if access control to the particular resource is granted. Both scenarios are depicted in the message sequence diagrams in the Sections 5.3 and 5.4, respectively.

4.6 Registration, discovery and interoperability of smart objects

Registration, discovery and interoperability of smart objects is an important aspect that needs to be considered within SSP. When a new IoT device enters a SSP, it should have the possibility to automatically register to the SSP, while SSP Gateway should be able to automatically discover new devices, and enable their registration.

This section presents a registration and discovery mechanism of smart objects planned to be implemented within an existing system developed at UNIZG-FER. In the future steps of symbIoTe project, this mechanism will be considered as a possible solution for registration and discovery in symbIoTe SSP, and adapted to become in line with symbIoTe SSP requirements. The system developed at UNIZG-FER takes advantage of a few opportunities in the area of IoT, brought by the possibilities to access sensor nodes via Internet, and by advancement of hardware components used by IoT devices. First opportunity is performing more complex algorithms, e.g. for data analysis, on IoT devices. Second opportunity is that one IoT device can have more than one sensor connected to it, and the fact that it can provide data to more than one service. The third opportunity arises from the fact that services do not need to be active all the time, but only through certain periods of time. Therefore, obtaining data from sensors should be executed only at times when there is expressed demand for that data by users. The second and third opportunity are one of the main challenges which arise from the integration of WSNs with IoT - offering services based on user-centric approach. Only those services for which exists an expressed interest by users at a certain moment will be executed on IoT devices. Not all services able to be run on those devices, and not all the time. The designed solution takes advantage of the aforementioned opportunities and brings the following novelties to IoT systems with battery-powered IoT devices:

- enables a user-centric deployment of services, based on needs or rules defined by users;
- data processing tasks can be executed on IoT devices or on IoT gateway, the decision is made according to energy efficiency of possible solutions; and
- each service has a priority level, where the goal is to extend the lifetime of the services with higher priority.

The goal of the proposed system developed at UNIZG-FER is to enable aforementioned novelties by proposing a service provisioning mechanism, more specifically an algorithm for allocating tasks of which IoT services are composed to different nodes within IoT architecture, and a protocol that enables exchange of control information. Task allocation is a well-studied problem in parallel and distributed computing research. Our mechanism

enables deciding on which nodes should certain tasks be executed, starting or stopping the services based on rules expressed by users, and choosing which services should have their lifetime extended. The decision is made according to energy efficiency, i.e. battery powered IoT devices are chosen in a way that maximizes the availability of services with higher priority. The architecture of the system is shown in Figure 9.

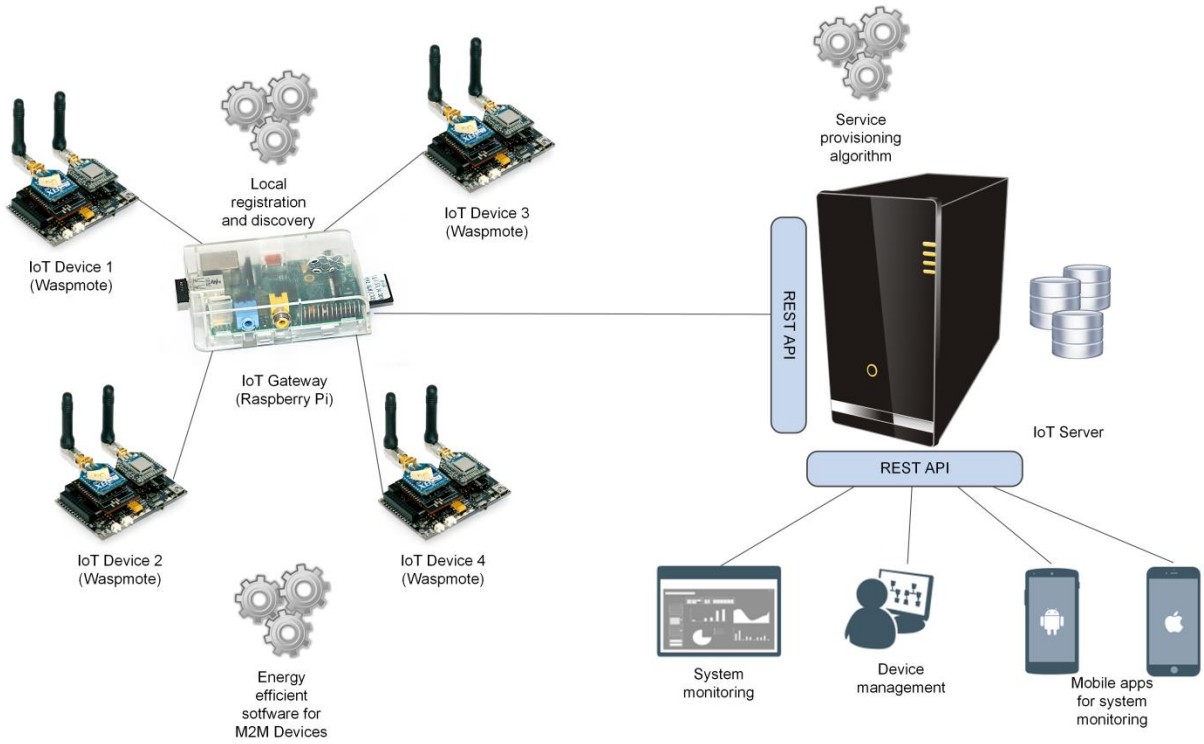

Figure 9 – Architecture of the IoT system for energy-efficient Service Provisioning

As can be seen in Figure 9, apart from task allocation mechanism (referred to as service provisioning mechanism), the system contains components for system monitoring (web and mobile applications for monitoring the values of obtained measurements, and device management application connected to service provisioning algorithm). On the side of IoT devices, energy efficient software for IoT devices is developed, enabling operation in a way that consumes smallest amounts of energy.

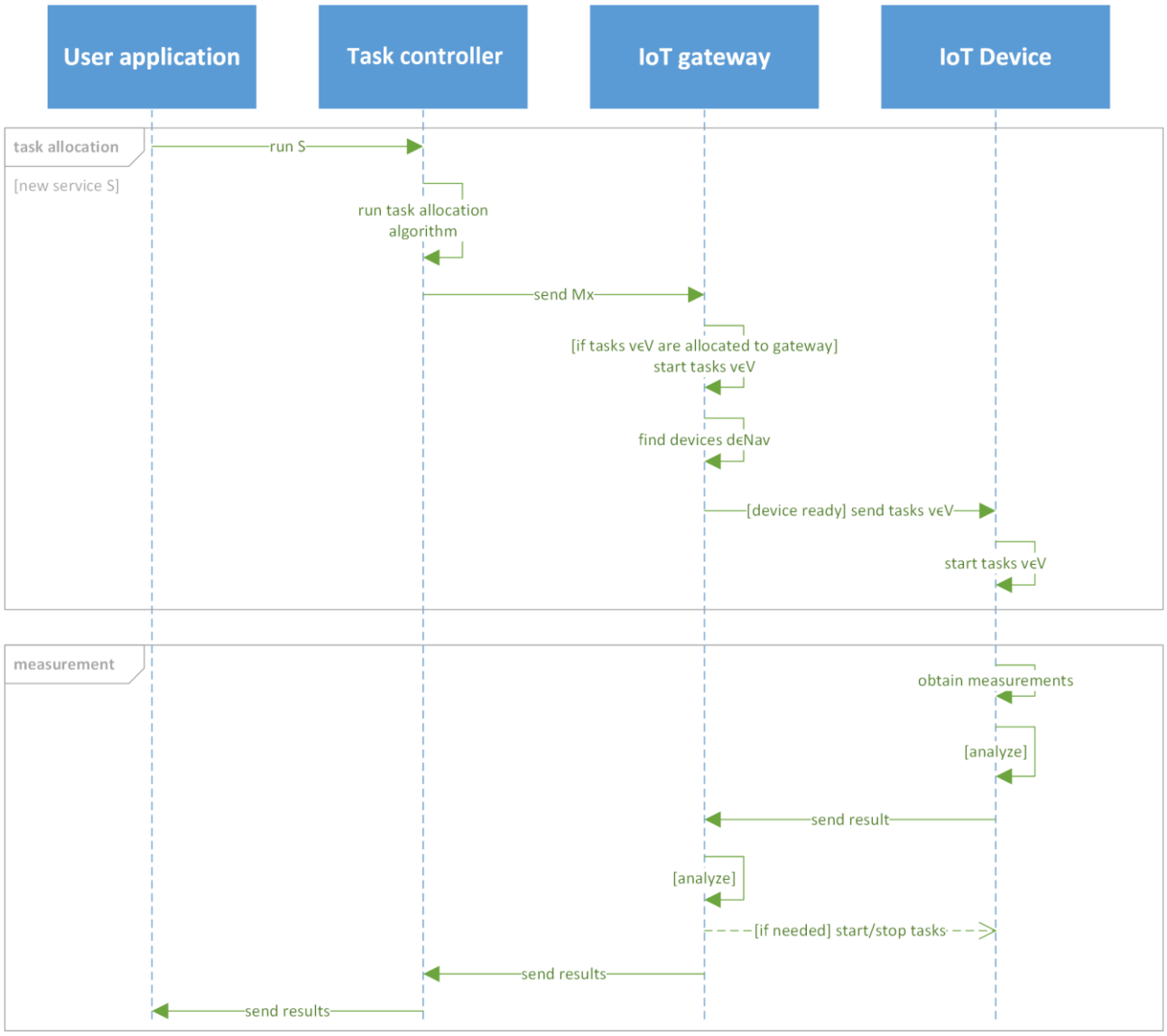

Figure 10 – Communication protocol for task allocation

The protocol for allocating tasks to IoT devices is shown in Figure 10 [3] Device management application from Figure 9 or a certain rule within the system initiates a service. Task allocation algorithm is executed on a Task controller, a component in the network which runs a rule-based engine or receives commands directly from users to start or stop a service. All task-device mappings (*Mx*) are forwarded towards the IoT gateway which then starts tasks *v* allocated to itself, or forwards tasks to certain IoT devices d that were chosen by the algorithm. Since IoT devices spend majority of their time in low-energy mode (since they are battery-powered, in such setting it is substantial to conserve energy), the IoT Gateway needs to wait until they are awake to send them commands. After IoT device receives new requests, it starts to execute the allocated tasks and forward the results to gateway. Additionally, the devices need to listen to new requests from IoT gateway.

The mechanism for discovery of new smart objects will be developed and integrated with the presented system shown in Figure 9. The discovery process is planned to work with

XBee [19] and LoRa wireless technologies, with the possibility to be expanded to other technologies. The discovery, along with the registration process, is planned to be executed periodically or just before the process of starting/stopping a service, shown in Figure 10. This particular mechanism is planned to be considered within symbIoTe for discovery and registration within SSP. In the future steps of symbIoTe project, it is planned to examine how this mechanism fits with SSP requirements and other concepts within SSP. If the mechanism proves to be applicable in symbIoTe, it will be adjusted to become in line with symbIoTe SSP.

The mechanism functions as follows: each IoT device entering the system should be initially configured (in XBee network, this is done by setting the Channel, PAN ID and encryption method) [3]53. Furthermore, the software for each device should be deployed, enabling the device to read the measurements from the connected sensors when request for its data is received.

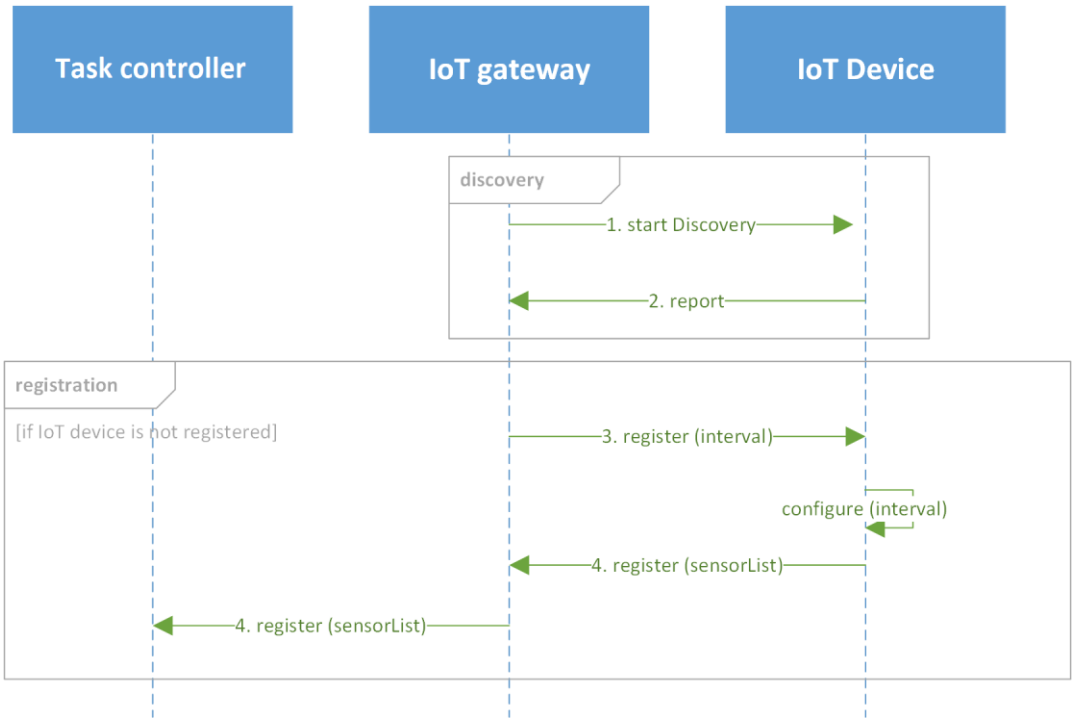

Figure 11 – Discovery and registration

Before joining the system, the device should register by providing data about sensors it has connected, and it should configure intervals in which the device will be able to be contacted by IoT gateway. The initial idea is shown in Figure 11, and inspired by XBee discovery process [5]. The IoT gateway should at certain times (e.g. before starting a new service) initiate discovery process. If the process recovers unregistered devices, in the next stage it should suggest the intervals in which IoT devices should be available. During the rest of the time, the IoT Device should perform the assigned operations or stay in low-energy mode in order to save energy. Furthermore, the device should send data about the sensors it has available, that should be stored in the database located at Task controller.

After the planned discovery and registration process, the tasks on the newly registered IoT device should be managed by using the process presented in Figure 10. At the moment,

the focus is on enabling IoT devices to be discovered and registered by using XBee communication modules. In the future, we plan to expand this process also by enabling discovery and registration of LoRa devices. The process of discovery and registration should work regardless of the communication technology. As mentioned earlier, in next steps of symbIoTe project, the applicability of this mechanism within SSP will be considered.

4.7 Smart Device event notification mechanism

The symbIoTe environment should enable functionalities for remote management and monitoring of the end devices (e.g. sensor or actuators). Thus, providing efficient notification mechanisms for events to the Smart Devices plays a significant role. By asynchronously sending notifications about changes in system behavior symbIoTe will reduce the network traffic required for obtaining current readings of the end-devices and improve overall system monitoring, including SSP gateway, middleware and SDEVs.

In order to define event notification mechanisms, we analyzed existing solutions present in various network and device management protocols. For example, SNMP (*Simple Network Management Protocol*) enables asynchronous notifications with *traps*, which represent predefined observed properties. When the value of this property changes, an event notification for all network devices is triggered. Moreover, TR-069 (*Technical Report 069)* offers advanced event notification mechanisms, which are two-layered and enable a higher level of granularity. The first layer consists of events that occur on the gateway and describe significant state changes in the system. On the second layer, there are events that primarily occur on end devices, their services and properties and can be efficiently used for monitoring in SSP scenarios. This is accomplished by extending system parameter values with attributes, that define which value or state changes in system should be reported to the other devices in the system.

From the above-mentioned solutions, we conclude that the most important characteristics of the event notification mechanism are:

- extensibility of event types
- granularity of the event subscription mechanisms
- parameterized notification deliveries

Event types should be structured to provide a clean overview of the supported events in symbIoTe. It is essential to provide a sufficient set of events and split them into event groups, so that entities in the SSP can obtain events of interest in a simple and efficient way. Relying on the architecture of the symbIoTe ecosystem, we propose solution for dividing SDEV event types into the following groups, including event type examples:

- SDEV Events impacting whole Smart Space
    - Smart Space Registry Change Event
        - Events that describe changes in the Smart Devices registry in SSP
        - *Examples*: SDEV joined SSP, SDEV left SSP (Device roaming scenario) …
- Smart Device events
    - Smart Device State Change Event
        - Events that describe changes in the overall functioning of the SDEV

        - *Examples*: SDEV started functioning, SDEV stopped functioning
    - Smart Device Property Change Event
        - Event that describe state or value changes of properties or services in the SDEV
            - *Examples*: current temperature measured in temperature sensor changed, dim level in smart bulb changed

Please note that this organization of events is a preliminary solution and could be extended or changed in the future. However, it is essential to provide an initial skeleton for grouping and categorizing events, since it becomes significant once the number of supported events rapidly increases.

The **granularity of the subscription mechanism** relies on the above declared grouping of supported events. Main benefit of the granularity is the possibility for other SDEVs in the SSP to subscribe to and only receive a minimum sufficient set of events from the particular SDEV. By using a tree structure to group event types, subscriptions to events are provided in a way that resources in the SSP can subscribe to either all or only particular events in a group. Furthermore, other SDEVs can also unsubscribe from any event type declared in the event grouping. For example, in the Smart Residence scenario, by subscribing to the *Smart Device Events, SSP* entities will be notified when any of the events in that subtree occur, e.g. *SDEV started functioning* or *dim level in smart bulb changed.* In case the notification of certain event isn't required, it can be excluded by unsubscribing from a specific event, e.g. all events from the *Smart Device Property Change* group, only events from the *Smart Device State Change* group will be delivered.

Parameterized notification delivery provides the SDEV with the possibility to notify system about the previously subscribed events and depending of the event's impact on the other SDEVs and overall system. This partially corresponds to requirement number 67 ("*An app/enabler SHOULD be able to receive a notification whenever an L4-compliant resource it is using changes Smart Space association*"), which states that the notifications should be triggered when L4-compliant resource changes the SSP. For example, some events could be used for tracking minor system properties over time, which means Smart Devices shouldn't be informed about the event as soon as it occurs, but with a certain delay, e.g. periodically, in order to collect data for further analytics. Based on those findings, we defined notification delivery policies. One of those policies has to be chosen during the event subscription. Those event delivery policies are:

- Periodical notification
    - Events are buffered until a certain time period expires and the event delivery is triggered
- Immediate notification
    - Events aren't buffered, but rather propagated immediately to all of the subscribed entities

It is also important to notice that buffered events are bound to the identifiers of the subscribed entities, which means that if entity changes environment, e.g. Smart Device roams, all undelivered events will be removed in order to minimize memory footprint.

4.7.1 Events structure and integration

Since the main purpose of the symbIoTe ecosystem is to achieve interoperability between different platforms and vendors, event notification functionalities should build on common data structures and interfaces, independent from the integrated platforms. Data structures required for implementation and integration of events correspond to the information required for collecting, buffering and delivering events and the information sent on message delivery.

Since specific events are tightly connected to changes of particular resource, each event subscription should be linked to the specific instance of the SDEV in SSP.

This means that each SDEV instance should be extended with the information about current event subscriptions. Each event subscription should include subscribed entities identifiers, which have to be unique across the symbIoTe, type of events the Smart Device is subscribed to and delivery policy (whether events are delivered periodically or immediately). This information should be used for filtering and further propagation of events. By having a list of subscribed entities for particular events in the SDEV instance, the distribution of events across the network will be using a *multicast* type of message sending.

Once an event occurs in any SDEV in the SSP, a sufficient set of information about this event should be delivered, in order to provide a complete description of the system change. This information set should contain:

- Event type, optionally event code
- Human-readable description of the event
- Timestamp, when the event occurred
- Additional information (optionally)
    - Smart Device identifier
    - Optionally SDEV property name, identifier and new value or state

Please note that the above described information set is prone to be changed or extended, since it is highly dependable on the event types defined for the particular SDEV.

4.8 Wireless technologies

Smart Devices can be equipped with one or more widespread wireless technologies, such as ZigBee [15] and Wi-Fi, operating in the unlicensed ISM bands, or 3G/4G cellular technologies. These technologies are often complementary in terms of coverage, reliability and costs. Therefore, for a given device with multiple available technologies, it is important to select the most suitable technology and access point to support a specific service in a given environment. The symbIoTe environment may face this problem, identify the coexisting networks and allow fast reconfigurations and integration of multiple network interfaces.

Moreover, apart from the single-device decisions, the proliferation of platforms based on Wi-Fi and ZigBee technologies working on unlicensed bands, as well as the recent introduction of LTE in Unlicensed spectrum (LTE-U/LAA-LTE) is creating coordination problems among independent platforms, which may be faced by the symbIoTe middleware. Finally, roaming across platforms require authentication and authorization

problems to be solved at the connectivity levels, i.e. not only to access the services available in the application domain, but also to attach to a particular wireless network, which represents a basic connectivity service available in all the platforms. Since multiple infrastructures can be deployed within the Smart Space, the symbIoTe middleware may enable bartering mechanisms among different IoT platforms to take care of the roaming smart devices.

Summarizing, although not tangible as hardware and software resources deployed in the Smart Space, the wireless medium represents a physical resource in the Smart Space, whose access needs to be orchestrated among coexisting IoT platforms and Smart Devices by means of the symbIoTe client and middleware modules. A dedicated local component may be considered for managing the access to this resource. In order to clarify this concept, we consider the following two reference scenarios.

Scenario 1: *a smart device/smart platform needs to set-up a ZigBee ad-hoc network in presence of interfering IoT platforms.* In this scenario, the smart device discovers the interfering IoT platforms available in the smart space and tries to infer about the effects of potential interferers for the support of its application requirements. It then informs the local component responsible for wireless medium allocations in the SSP about its needs, and receives indications about the less interfered channels that can be used. These indications are computed by means of global optimizations improving the coexistence of multiple networks. Indeed, the SSP can also change the configuration of the coexisting IoT platform as a consequence of the new resource allocation. In case the SSP does not support coordination among coexisting wireless networks, the SDEV can exploit one of its internal modules to configure its ad-hoc network by means of unilateral optimizations (e.g. to identify a ZigBee channel non-interfering with the coexisting Wi-Fi networks).

Scenario 2: *a smart device based on LoRaWAN™ technology needs to be connected to its network server in a region covered by two network providers.* In this scenario, the Smart Device can exploit multiple available gateways to improve the link reliability. In the LoRaWAN™ architecture (see Section 3.2.3), devices are associated to network servers, while gateways simply act as relays, forwarding data frames received from the wireless link to the network server on the Internet. In case SDEV transmit broadcast frames, each gateway in the coverage area of the devices can forward the frames to the network servers, thus improving the probability of correct delivery. To this purpose, gateways need to deploy additional functionalities to address the correct network server and/or to filter the frames to be forwarded, according to some link quality and load balancing metrics or SLA among operators. For example, gateways could forward frames from a maximum number of devices belonging to a different provider or up to a maximum total number. Dynamic SLAs, which allow to bartering the number of end devices served in different geographic areas, can also be considered. Figure 12 depicts the envisioned scenario, in which we imagine to overcome current protocol limitations which prevent to reach more than one operator from the same gateway.

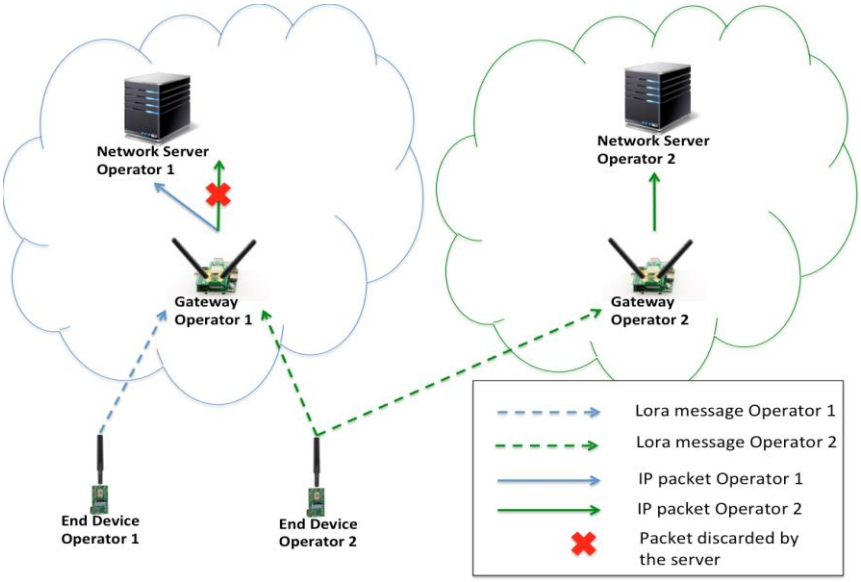

Figure 12 – Exemplary scenario based on the exploitation of multiple gateways

Another interesting scenario, based on the analysis of LoRa™ capacity limits, is the possibility of optimizing the reuse of several frequencies and orthogonal spread factors in presence of many gateways, possibly belonging to different operators. In this case, symbIoTe could act as a mediator between the different coexisting platforms.

4.8.1 Detect interfering technologies by means of Wi-Fi interfaces

Wi-Fi interfaces are undoubtedly very common in smart devices, especially for simple devices equipped with single wireless interfaces. This technology is not able to decode frames transmitted according to different modulation formats and therefore cannot directly detect the presence of coexisting networks. Despite of that, it has been shown that it can indirectly infer about the presence of coexisting networks by monitoring some low-level receiver events, such as synchronization errors, invalid header formats, too long frames and repeated synchronization trials. Indeed, in presence of non-Wi-Fi modulated signals, the occurrence of these types of errors follows statistics that can be easily recognized. Moreover, the duration of the error bursts depends on the transmission interval of the interference source, while the error spacing depends on the receiver implementation. On the basis of these considerations, in a previous work CNIT developed a tool, called ErrorSense [20], based on the adoption of hidden Markov chains for characterizing the behavior of Wi-Fi receivers in presence of controlled interference sources (training phase) and then run-time recognizing the most likely cause of error patterns. By means of experimental results, with commercial cards the effectiveness of the approach was proven for detecting ZigBee and microwave interference. CNIT wants to extend this tool to other emerging technologies (e.g. LTE in unlicensed spectrum) and by using simpler classifiers that can be run by any device.

4.8.2 Set-up of cross-technology communication channels

It may happen that a smart device equipped with a Wi-Fi (or ZigBee) single interface is located in a smart space in which only gateways based on ZigBee (or Wi-Fi) technology are available. Although in this scenario the association of the device could appear to not be feasible, CNIT has demonstrated that indirect forms of communication are possible for creating low-rate communication channels, whose bandwidth can be enough for several sensing applications. In particular, symbol constellations for the Wi-Fi-to-ZigBee and ZigBee-to-Wi-Fi links can be built by considering the physical parameters of the two standards. On one side, the interference duration can be measured with a different accuracy according to the carrier sense granularity of the receivers, which is much smaller for Wi-Fi (a few μs) than ZigBee (about 128 μs). On the other side, the variability range of the interference duration depends on the maximum payload size (2304 byte for Wi-Fi, 127 bytes for ZigBee), which can be mapped into a maximum number of potential communication symbols by opportunistically spacing the frame transmission times. Moreover, the valid set of symbols can be restricted to increase robustness of other interfering sources, which can cause insertions and deletions of symbols.

Channel insertions can occur when a receiver erroneously considers interference intervals due to other interfering sources and coexisting networks as a cross-technology symbol. This phenomenon can be relevant in the Wi-Fi-to-ZigBee direction, because of the density of coexisting Wi-Fi networks. Although symbols can be chosen with durations different from environment traffic, it may happen that channel busy times due to the overlapping of multiple interference sources are mapped by the ZigBee receiver (which moreover works with a carrier sense granularity of only 128 μs) into valid symbol durations. In the other direction, the probability of having an insertion due to other ZigBee networks is very low because they need to be in proximity of the Wi-Fi receiver. Moreover, Wi-Fi receivers can distinguish channel busy intervals with an accuracy of 1 μs.

Channel deletions can occur when the cross-technology symbol is missed by the intended receiver because the interference is not detected or its duration measurement is wrong. This phenomenon can be very critical in the ZigBee-to-Wi-Fi direction, because with probability 1/4 the Wi-Fi receivers assume that ZigBee frames are valid Wi-Fi frames and read the duration from the relevant frame field (rather than measuring it on the channel). This corresponds to a wrong measurement and, generally, to the loss of the symbol. Moreover, in this direction the ZigBee interference power is 20dB lower than other coexisting Wi-Fi networks.

CNIT plans to study specific coding and communication schemes, based on repetition codes and multiple dynamic constellations, to exploit such a channel and to characterize its capacity. Indeed, capacity of insertion and deletion channels is still unknown, although there are some bounds and coding solutions to deal with binary channels (which model synchronization errors between the transmitter and the receiver) or symbols carrying large number of bits (which model packet losses in Internet). Our cross-technology channel falls in the middle between these two cases, because symbols carry multiple bits in both the channel directions, but such a number is limited to a few units.

4.8.3 Support cross-network TDMA

In a smart space, it is likely that multiple wireless networks based on heterogeneous or homogeneous technologies coexist. In such a scenario, coordination mechanisms can

lead to significant performance benefits. Suppose, for example, that two IoT platforms work on Wi-Fi networks configured on the same operating channel or one of the two platforms exploit a proprietary wireless technology working on the Wi-Fi channel used by the coexisting platform. While the first condition is intrinsically tackled by the carrier sense mechanism of the Wi-Fi access protocol, the second condition could lead to coexistence problems. A possible solution for dealing with this condition or for improving coexistence of contention-based networks is enabling time-division among coexisting networks, i.e. introducing a coordination mechanism according to which some networks or some cells as a whole are alternatively prevented from accessing the channel and enabled to transmit at regular non-overlapping time intervals. Examples of solutions that introduce coordination among the data path of independent networks have been already proposed in research literature. CENTAUR [1] allows configuring of non-conflicting operation intervals among the APs of a Wi-Fi large-scale network by exploiting a central controller for scheduling the activity intervals in each cell. FLUID [2] allows allocating heterogeneous bandwidths among the cells according to the traffic load. The implementation of these solutions requires a global orchestrator in the middleware, as well as a basic synchronization and pausing mechanism for the wireless interfaces in the smart device. CNIT plans to design effective solutions, working on commercial (off-the-shelf) wireless interfaces, to support programmable time-division activities. In the allowed activity interval, medium access is performed according to the standard protocols supported by the commercial interface. Synchronization limits and latencies for switching to the pause mode have to be considered by the global orchestrator to make time allocations feasible. For more advanced wireless interfaces, supporting programmability in terms of medium access protocols (such as interfaces employing the Wireless MAC Processor architecture or other software-defined MAC), time-division can be supported by programming a dedicated protocol switching from an elementary MAC protocol and doze state.

4.8.4 Transmit to multiple gateways

For some technologies, such as LoRa-based devices, frame transmissions are natively received and forwarded by multiple gateways. This operation allows to increase the wireless connectivity robustness and simplify the mobility management. These solutions could be generalized for transmitting to multiple gateways managed by different operators and/or for transmitting to multiple gateways based on heterogeneous wireless technologies. In the first case, at the device side, it could be enough to perform broadcast or multi-cast transmissions; in the second case, it is necessary to add a software module to dispatch or replicate traffic flows among heterogeneous wireless interfaces connected to different gateways.

4.8.5 Support device configuration protocols

In order to enforce device configurations decided by the symbIoTe middleware, it is necessary to define control messages and protocols between the smart devices and the relevant middleware modules. Rather than defining new protocols, CNIT plans to investigate the generalization or adaptation of existing protocols dealing with device configuration. Examples of these protocols are the ones used for working on experimental testbed facilities, such as protocols defined in the OML/OMF framework, or for configuring

Wi-Fi Access Points in large scale networks, such as the CAPWAP protocol or various extensions of the OpenFlow protocol, such as OpenRoards.

4.8.6 Fast attach

One of the main issues of IoT environments is the mobility of SDEVs. For example, Wi-Fi connections suffer well-known problems regarding connectivity and authentication mechanism in terms of delay of association time. An interesting solution could be provided by the emerging IEEE 802.11p standard: 802.11p is designed for vehicular scenarios and aims to overcome connectivity lag issues, interference and mobility limitation of current Wi-Fi standard. However, this standard does not work on ISM bands and this could be a limitation in terms of availability and short-term penetration in market applications. A short-medium term solution could be provided by the analysis and exploitation of wide open-source project deployed for association and authentication in Wi-Fi networks. An interesting research path can be focused on an analysis of current association mechanisms in order to find an optimized solution able to give quicker and stronger Wi-Fi connection in public networks, which may speed-up network discovery, association and/or IP acquisition. Indeed, authentication can last for a long-time interval, because of the security architecture of the federated system, which involve the exchange of messages among multiple entities.

5 Components interaction

In the following paragraphs, sequence diagrams are presented to describe the expected interactions among the SSP, SDEV and CLD level components.

These scenarios show how the SSP and the SDEV interface with the upper domains (the APP and the CLD domains), when some particular action is triggered:

- an app joining a SSP
- a Smart Device joining a SSP
- the access to a SDEV is requested from within the SSP
- the access to a SDEV is requested from outside the SSP

Moreover, the message exchange steps between S3M components are described, in order to explain the internal mechanisms which handles the different SSP functionality.

5.1 App joining Smart Space

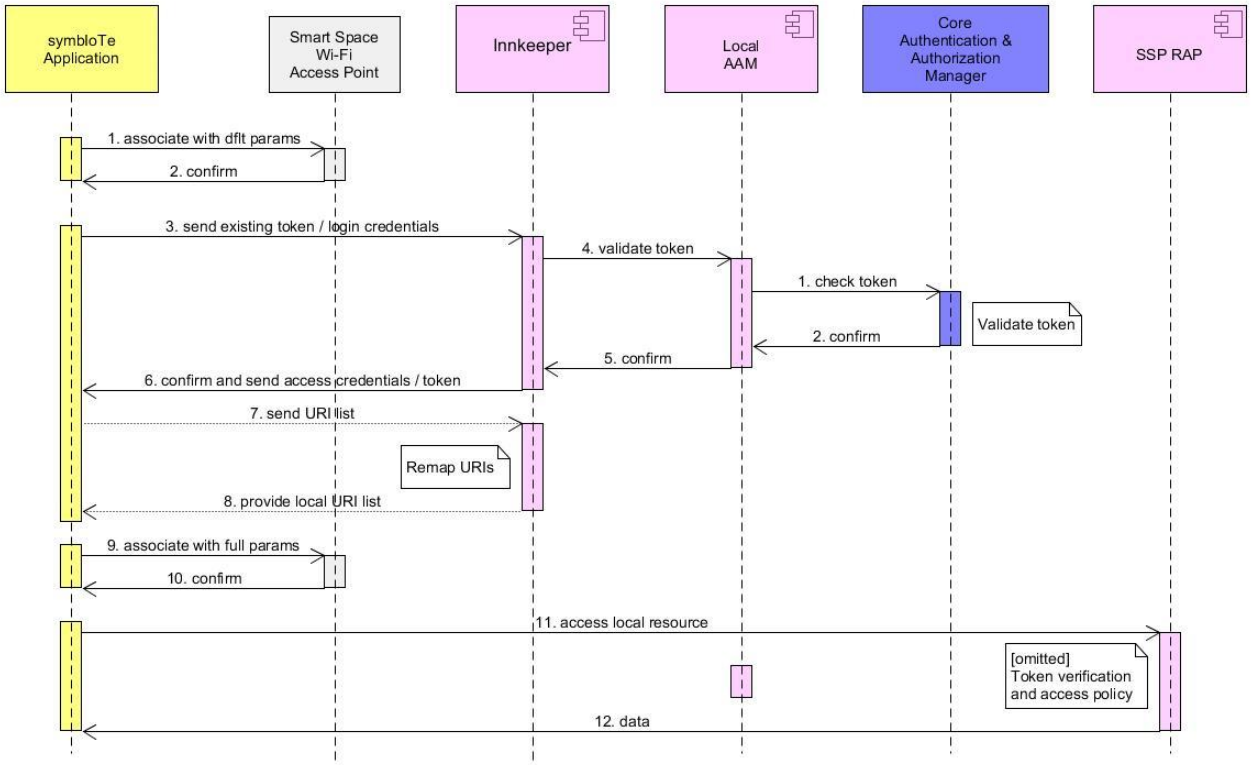

Figure 13 – App joining Smart Space

An application that wants to join a SSP needs to run on a device (e.g. a smartphone) which is connected to a Wi-Fi Access Point of that SSP. This can be achieved by an automatic association of the device to a well-known SSID (or SSID pattern) with pre-shared credentials (or pre-shared algorithms to infer credentials given the SSID name).

The trigger for this action can be either detection of the SSID name, or an explicit action by the user.

Then the device is given a “limited-access IP address” on the same network where the Innkeeper is located. So, the app will need to determine the Innkeeper address via a standard *autoconf* mechanism (IP multi/broadcast, mDNS, DNS-SD, …).

Then the application can send login credentials (or a valid token) to the Innkeeper, which will validate the credentials (or token) with the Local AAM component. Upon successful validation, the Innkeeper provides the app with the credentials to access the real Smart Space SSID, and optionally a local access token.

Optionally, the application can send also a list of URIs it is using, to the Innkeeper, which will remap any URI related, to locally accessible resources.

The first part of the association is now completed, and the application can fully associate the device to the Smart Space SSID. At this point, local resources can be accessed through the Local RAP automatically (remapped URIs); the Local RAP will validate access credentials with the Local AAM.

5.2 Smart Device joining Smart Space

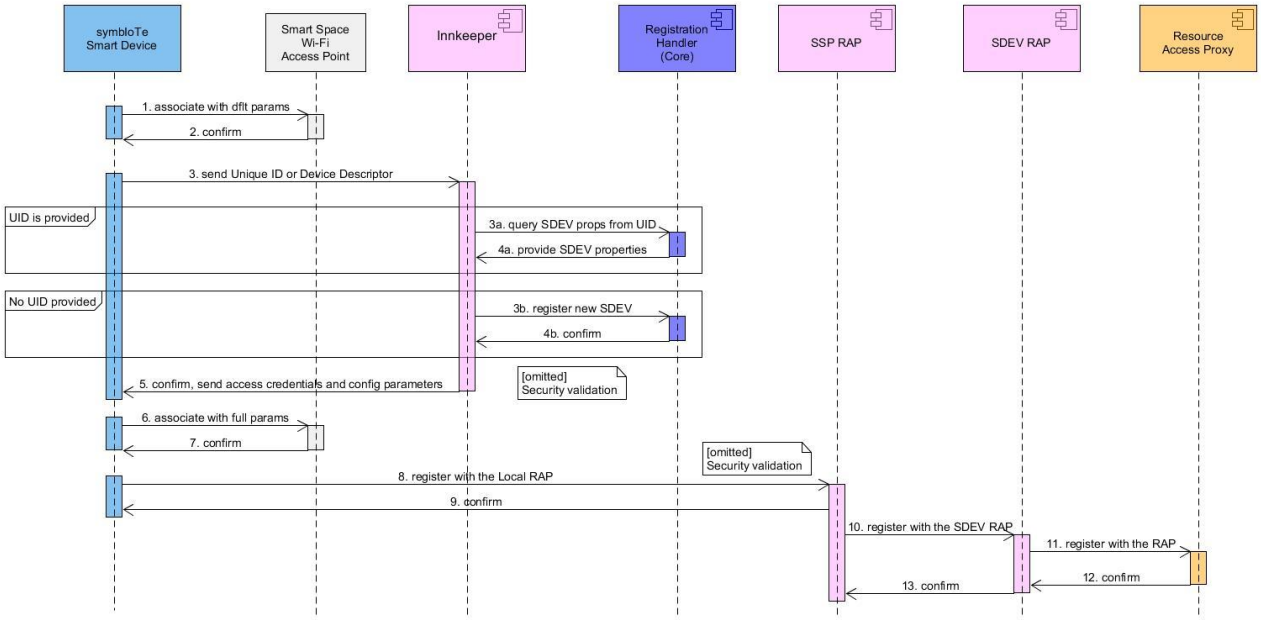

Figure 14 – Smart Device joining Smart Space

An SDEV that wants to join a SSP needs to associate itself to a Wi-Fi Access Point of that SSP. This can be achieved by an automatic association of the SDEV to a well-known SSID (or SSID pattern) with pre-shared credentials (or pre-shared algorithms to infer credentials given the SSID name). The trigger for this action can be either detection of the SSID name, or an explicit action by the user (e.g. press the “*associate*” button).

Upon successful join, the SDEV is given a limited-access IP address on the same network where the Innkeeper is located. The device will need to determine the Innkeeper address in the same way as an application which wants to join a SSP (Section 5.1).

The SDEV sends either its Unique ID or its Device Descriptor to the Innkeeper:

- if an UID is provided, the Innkeeper queries the symbIoTe Core to retrieve its data model and operating parameters. The Registration Handler component also specifies whether the SDEV shall be registered as a new device (L3 behavior) or as a roaming device (L4 behavior).
    - To simplify the flow at this stage, this query will also inform the Registration Handler that the given SDEV is currently associated with this Smart Space; this registration could also be done as a further step subsequently.
- If no UID is provided, the Innkeeper registers the SDEV as a new device. It is assumed that the SDEV itself provides its data model and parameters.

At this point a successful registration and validation have been reached, so the Innkeeper provides the SDEV with the credentials to access the real Smart Space SSID, and optionally further configuration parameters; the Innkeeper also provides the address of the Local RAP.

The first part of the association is now completed, and the SDEV can fully associate to the Smart Space SSID. The SDEV registers with the SSP RAP, in order to provide an access point to its exposed resources for any other device, component or app, within the same SSP. The SSP RAP, then, propagates the registration to the SDEV RAP; the SDEV RAP opens a transparent tunnel to the CLD RAP to allow requests coming from outside the SSP to reach the SSP RAP.

The above described steps are valid in the Wi-Fi connectivity scenario. An alternative scenario we are considering in symbIoTe is the LoRa™ connectivity scenario, where an SDEV (having both a LoRa™ and a Wi-Fi radio) exchanges information with the Innkeeper via the Application Server in the LoRa™ network.

We foresee the possibility of an SDEV roaming from a LoRa™ network to a Wi-Fi network, and vice versa. This could happen in the same SSP (in which case no interaction with an Innkeeper is involved) or between two different SSPs (in which case the above described steps apply, with the possible exception of providing Wi-Fi credentials to the SDEV while it is still in the source SSP).

5.3 Access to SDEV resources from within the SSP

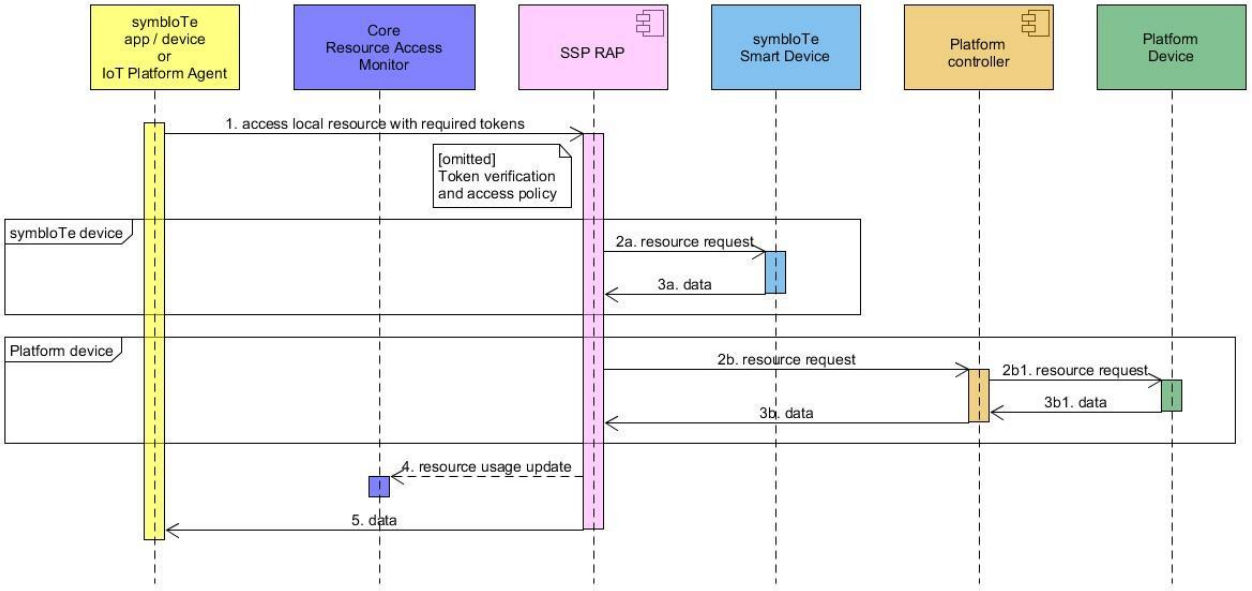

Figure 15 – Access to SDEV resources from within the SSP

This diagram shows the interaction that occur when a symbIoTe application, component, or an IoT Platform Agent registered in the same SSP of the required SDEV accesses the SDEV resources.

Assuming that the requester already has the required security tokens, it sends its request to the SSP RAP. Then the SSP RAP forwards the request to the SDEV (it is assumed that the SDEV natively provides a symbIoTe standard access interface as well).

Optionally, the SPP RAP updates the Core Resource Access Monitor; in case of connectivity problems with the CLD, this information could be stored and sent when connection is re-established.

5.4 Access to SDEV resources from outside the SSP

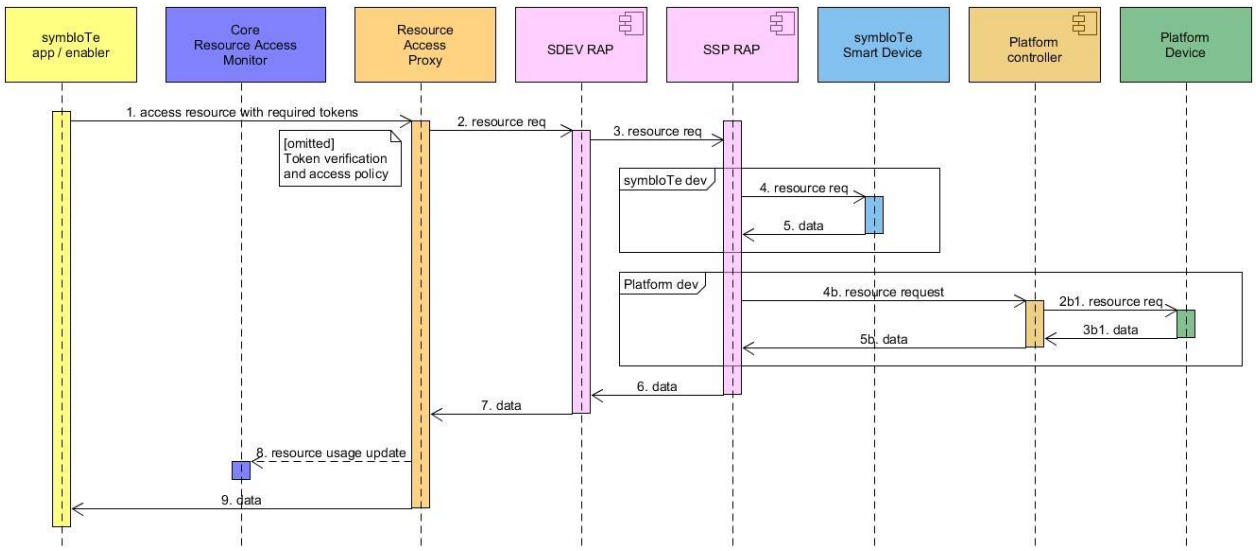

Figure 16 – Access to SDEV resources from outside the SSP

This diagram shows the interaction that occur when a symbIoTe app or component external to the SSP of the required SDEV need to access the SDEV resources.

The interaction is similar to that of the L1 case (see D1.2: Initial Report on System Requirements and Architecture), but in this case the CLD-RAP will forward the request to the SDEV RAP. The SDEV RAP then performs the request by going through the SSP RAP.

6 Conclusions and Next Steps

In this deliverable, the accommodation of more interoperable IoT platforms into the same virtual space has been presented and analysed, with the purpose of providing a container and gateway for resources to access and to be accessed from the outside.

Features and functionalities of the Smart Space domain have been highlighted, focusing on the platform interoperability and its dependency on the cloud domain, how to dynamically adapt to the changing topology and how to map existing IoT platforms. Smart Devices have been presented as well, describing how to handle object roaming and the different compliancy levels with the symbIoTe environment.

The architectures of the SSP and SDEV domain have been shown, together with a detailed analysis of the symbIoTe middleware and its components. Security issues have been considered with the aim of enforcing authorized access, preserving privacy of the user information and preventing malicious attacks. Through the design of the SSP and SDEV (and, consequently, of the S3M) it has been defined how symbIoTe can deal with the dynamic changes that take place inside a local environment and how to connect with IoT devices owned by other platforms and with symbIoTe compliant Smart Devices.

Multiple domains where future work is needed have been identified, mostly concerning the development of S3M in task T4.3. This includes the design of explicit mechanisms for registration, discovery and interoperability of smart objects inside the SSP (e.g. a new device that enters a symbIoTe SSP should have the possibility to automatically register). An automatic system that notifies the SSP about device status changes would be a desirable behaviour of symbIoTe, in order to reduce the network traffic required for obtaining information from resources, improving the overall system monitoring.

The design choices described in this deliverable have been the starting point for drafting the sequence diagrams describing the interaction of components, laying the grounds for the subsequent development phase of the SSP and SDEV domains components.

7 References

[1] Vivek Shrivastava, Nabeel Ahmed, Shravan Rayanchu, Suman Banerjee, Srinivasan Keshav, Konstantina Papagiannaki, and Arunesh Mishra. 2009. CENTAUR: realizing the full potential of centralized wlans through a hybrid data path. In Proceedings of the 15th annual international conference on Mobile computing and networking (MobiCom '09). ACM, New York, NY, USA, 297-308.

[2] Shravan Rayanchu, Vivek Shrivastava, Suman Banerjee, and Ranveer Chandra. 2011. FLUID: improving throughputs in enterprise wireless lans through flexible channelization. In Proceedings of the 17th annual international conference on Mobile computing and networking (MobiCom '11). ACM, New York, NY, USA, 1-12.

[3] Skocir P, Kusek M, Jezic G. Energy efficient task allocation for service provisioning in Machine-to-Machine systems. Concurrency and Computation: Practice and Experience, 2016, in review

[4] Libelium. Waspmote 802.15.4 Networking guide, 2016

[5] Digi, XBee Java Library, Documentation, http://docs.digi.com/display/DIGI/Home, access: January 2017

[6] Z-Wave, http://www.z-wave.com/

[7] Samsung SmartThings, https://www.smartthings.com/

[8] AllSeen / Open Connectivity Foundation, https://allseenalliance.org/

[9] Philips Hue, www.meethue.com

[10] Zipato Home Automation, https://www.zipato.com/

[11] Symphony, www.nextworks.it/en/products/brands/symphony

[12] Vera Smart Home, getvera.com/

[13] Netatmo, www.netatmo.com

[14] Koubachi, http://www.koubachi.com/

[15] ZigBee, www.zigbee.org

[16] LoRa™, https://www.lora-alliance.org/

[17] LoRaWAN™, https://www.lora-alliance.org/

[18] UNIDATA, https://www.unidata.it/

[19] XBee, https://www.digi.com/lp/xbee

[20] D. Croce, P. Gallo, D. Garlisi, F. Giuliano, S. Mangione and I. Tinnirello, "ErrorSense: Characterizing WiFi error patterns for detecting ZigBee interference," 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, 2014

[21] L. Florio and K. Wierenga, "Eduroam, providing mobility for roaming users," EUNIS 2005, June 2005

[22] Thread Group, Thread technical overview, https://www.threadgroup.org/Portals/0/documents/resources/Thread_Technical_Overview.pdf, 2015

8 Acronyms

- ABAC Attribute-Based Access Control
- ACK Acknowledgment
- APP Application
- AppSKey Application Session Key
- BLE Bluetooth Low Energy
- CLD Cloud
- ESSID Extended Service Set Identifier
- GBAC Group-Based Access Control
- GW Gateway
- IoT Internet of Things
- ISM Industrial, Scientific and Medical
- JWT JSON Web Tokens
- LoRaWAN Long Range for Wide Area Network
- MAC Medium Access Control
- NS Network Server
- NwkSKey Network Session Key
- RBAC Role-Based Access Control
- RAT Radio Access Technology
- RAP Resource Access Proxy
- S3M SymbIoTe Smart Space Middleware
- SLA Service Level Agreement
- SSID Service Set Identifier
- SSP Smart Space
- SDEV Smart Device
- TCP Transmission Control Protocol
- WSN Wireless Sensor Network