Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

symbIoTe Domain-Specific Enablers and Tools

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
NextworksSrl, NXW, Italy
Consorzio Nazionale Interuniversitario per le Telecomunicazioni, CNIT, Italy
ATOS Spain SA, ATOS, Spain
University of Vienna, Faculty of Computer Science, UNIVIE, Austria
UnidataS.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: Report on symbIoTe Domain-Specific Enablers and Tools

Type: Public

Editor(s): Tomasz Rajtar

E-mail: tomasz.rajtar@man.poznan.pl

Author(s): Mario Kušek (UniZG-FER), João Garcia (UW), Gerhard Duennebeil (AIT), Matteo Pardi (NXW), Petar Krivić (UniZG-FER Luca De Santis (NAVIGO), Raquel Ventura Miravet (S&C), Szymon Mueller (PSNC), Tomasz Rajtar (PSNC), Vasilis Glykantzis (ICOM), Zvonimir Zelenika (VIP), Marcin Plociennik (PSNC), Ivana Podnar Žarko (UniZG-FER)

Doc ID: D2.6-v1.0

Amendment History

Version	Date	Author	Description/Comments
v0.1	7/10/2017	Mario Kušek (UniZG-FER)	ToC
v0.2	7/11/2017	Mario Kušek (UniZG-FER)	Added section responsibilities
v0.2.1	7/18/17	Petar Krivić, Mario Kušek (UniZG-FER)	Updated sections 3 and 4.1
v0.3	7/20/2017	Matteo Pardi (NXW), Luca De Santis (NAVIGO), Raquel Ventura Miravet (S&C),	Integration of effort from NXW, S&C, NAVIGO and UNZIG-FER
v0.3.1	7/21/2017	Mario Kušek (UniZG-FER)	Comments on contributions
v0.4	8/30/2017	Petar Krivić, Mario Kušek (UniZG-FER), Matteo Pardi (NXW), Luca De Santis (NAVIGO),	Integration of effort from UniZG-FER, NAVIGO, NXW and ICOM comments (Vasilis) added after v0.3 by Tomasz Rajtar.
v0.4.1	8/31/2017	Mario Kušek (UniZG-FER)	Cleaning up comments
v0.5	9/01/2017	Tomasz Rajtar (PSNC)	Interfaces tables and general info tables for components added.
v0.6	9/11/2017	Matteo Pardi (NXW), Petar Krivić, Mario Kušek (UniZG-FER), Luca De Santis (NAVIGO), Zvonimir Zelenika (VIP)	Updated diagrams and procedures in section 5.3.1 Defining interfaces and component data Changes in 5.2 5.4 Section (enabler for indoor location) Executive summary, introduction and conclusion
v0.6.1	9/11/2017	Mario Kušek (UniZG-FER), Tomasz Rajtar (PSNC)	Minor updates of v0.6
v0.7	9/13/2017	Gerhard Duennebeil (AIT), Raquel Ventura Miravet (S&C), Luca De Santis (NAVIGO), Szymon Mueller (PSNC), Vasilis Glykantzis (ICOM), João Garcia (UW)	Integrated version with missing parts in sections 4 and 5 filled.
v0.8	9/26/2017	Marcin Plociennik (PSNC), Tomasz Rajtar (PSNC)	Internal review, editorial integration
v0.8.1	10/10/2017	Zvonimir Zelenika (VIP), Mario Kušek (UniZG-FER)	Changes according to internal review
v0.8.2	11/03/2017	Ivana Podnar Žarko	Final quality review with fixes and comments in all sections
v0.8.3	11/14/2017	Mario Kušek	Fixing comments from review.
V1.0	11/14/2017	Sergios Soursos	Final version

Table of Contents

(This page is left blank intentionally.)

1 Executive Summary

The aim of Deliverable 2.6, entitled “symbIoTe Domain-Specific Enablers and Tools”, is to document the final set of components and provided features implementing the symbIoTe Enablers. It also documents the architecture and design of Enablers for specific use cases.

Enablers are envisioned as domain-specific back-end services which are placed within the symbIoTe Application Domain (APP) to offer services to end-user applications while utilizing IoT resources provided by platforms. Since symbIoTe applications access resources directly on symbIoTe-enabled L1 platform instances, applications need to interact with multiple platforms and perform value-added services, e.g., data processing and aggregation on the application side. This may not be favorable for some domains and use cases where performed operations are compute-intensive, or dynamic and complex so that they require interactions with many resources managed by different platforms. Thus, Enablers are designed as value-added services providing domain-specific functionality to facilitate the development of domain-specific applications as well as cross-platform and cross-domain applications. They conceal the complexity of the symbIoTe ecosystem from applications and can be used as a single point of access to this ecosystem.

An Enabler takes on two major roles:

- Platform role, since it behaves as an IoT platform offering value-added IoT-based resources (services), and
- Application role, since it acts as any other application to access resources offered by symbIoTe-enabled L1 platforms.

In the platform role, an Enabler registers its value-added resources (services) with the symbIoTe Core Services, integrates all symbIoTe Cloud components for L1 compliant platforms, and serves application requests over its Interworking Interface and domain-specific interface. In the application role, the Enabler acts as any symbIoTe application which searches for resources using the Core Services and accesses resources offered by other L1 platforms. Thus, each Enabler instance integrates generic symbIoTe-defined components which are independent of its domain-specific features. We report the design of such generic components in this deliverable: these are four L1 Cloud components (Registration Handler, Resource Access Proxy, Authentication and Authorization Manager, and Monitoring) and three additional Enabler-specific components (Enabler Logic, Resource Manager, and Platform Proxy)

This document reports the final design of generic software components for building Enablers and includes component descriptions, a complete list of component features, as well as a description of their interactions and communication interfaces. This document is an accompanying report documenting the software produced in task T2.3 which is published as open source in the symbIoTe GitHub repository [5].

Within the symbIoTe project, Enablers are designed for domains covered by symbIoTe use cases to facilitate the development of use case specific applications. In particular, four such enablers are defined: Smart Mobility and Ecological Urban Routing Enabler, Smart Yachting Enabler, Smart Residence Enabler and Smart Stadium Enabler. We report the design of all listed Enablers in this document with focus on their domain-specific features. Finally, to showcase a complete Enabler solution, we present the implementation details of the Smart Mobility and Ecological Urban Routing Enabler. It integrates an Interpolator for air quality monitoring which uses air quality readings acquired from different symbIoTe-

enabled platforms within the same city, performs data interpolation, and provides output results in an as-a-Service manner to applications. It also includes two additional services: Green Route Calculator to offer less polluted routes in the city to cyclists and pedestrians, and Point of Interest search facility.

The symbIoTe GitHub repository [5] contains a complete source code of the aforementioned Enabler components, configuration files and guidelines on how to implement a new symbIoTe Enabler (https://github.com/symbiote-h2020/SymbioteEnabler). The source code of the Smart Mobility and Ecological Urban Routing Enabler is available at https://github.com/symbiote-h2020/EnablerSMEUR).

2 Introduction

2.1 Purpose of this document

The purpose of this deliverable is to explain the design of symbIoTe domain-specific Enablers and to provide guidelines on how to implement a new Enabler for a different domain. This deliverable includes the specification of communication between components, basic information related to developed components: further information is provided on GitHub with instructions on how to implement a Domain-Specific Enabler for other purposes. The design and interaction between components in four specific use cases is provided to be used as an example for designing and implementing future Enablers in different domains.

2.2 Relation to other deliverables

The system Requirements relevant to Enablers are defined in D1.2 “Initial Report on System Requirements and Architecture” [3] and D1.4 “Final Report on System Requirements and Architecture” [4]. The generic architecture for symbIoTe Enablers is built upon the symbIoTe Core Services and L1 Cloud components presented in D1.4 since symbIoTe Enablers reuse components from the symbIoTe L1 solution presented in D2.2 “symbIoTe Virtual IoT Environment Implementation” [9] and D2.5 “Final symbIoTe Virtual IoT Environment Implementation” [7]. The proposed design and implementation of Domain-Specific Enablers is within the domains framed by the symbIoTe use cases which are reported in D1.3 “Final Specification of Use Cases and Initial Report on Business Models” [10]. The technologies used for the implementation of symbIoTe system are specified in D5.1: “Implementation Framework” [11]. This deliverable presents the final specification related to symbIoTe Domain-Specific Enablers which is a continuation of work reported in D2.3 “Report on symbIoTe Domain-Specific Enablers and Tools” [6].

2.3 Document structure

Section 3 describes the final generic Enabler architecture with all components and relevant features as well as interactions between components. Section 4 provides a detailed description of each generic Enabler component including documentation and implemented features. The design of Domain-Specific Enablers specific for symbIoTe use cases is presented in Section 5 while the implementation details of Smart Mobility and Ecological Urban Routing Enabler are provided in Section 6.

3 Generic Architecture for symbIoTe Enablers

The main purpose of Enablers is to provide value-added features based on IoT resources needed in specific domains. Examples of such functionalities are data aggregation based on sensor readings stemming from various IoT platforms, or continuous processing of such data, etc. However, each Domain-Specific Enabler has generic functionalities that enable its interaction with other symbIoTe-enabled platforms and components within the symbIoTe ecosystem. This generic architecture with required components is shown in Figure 1.

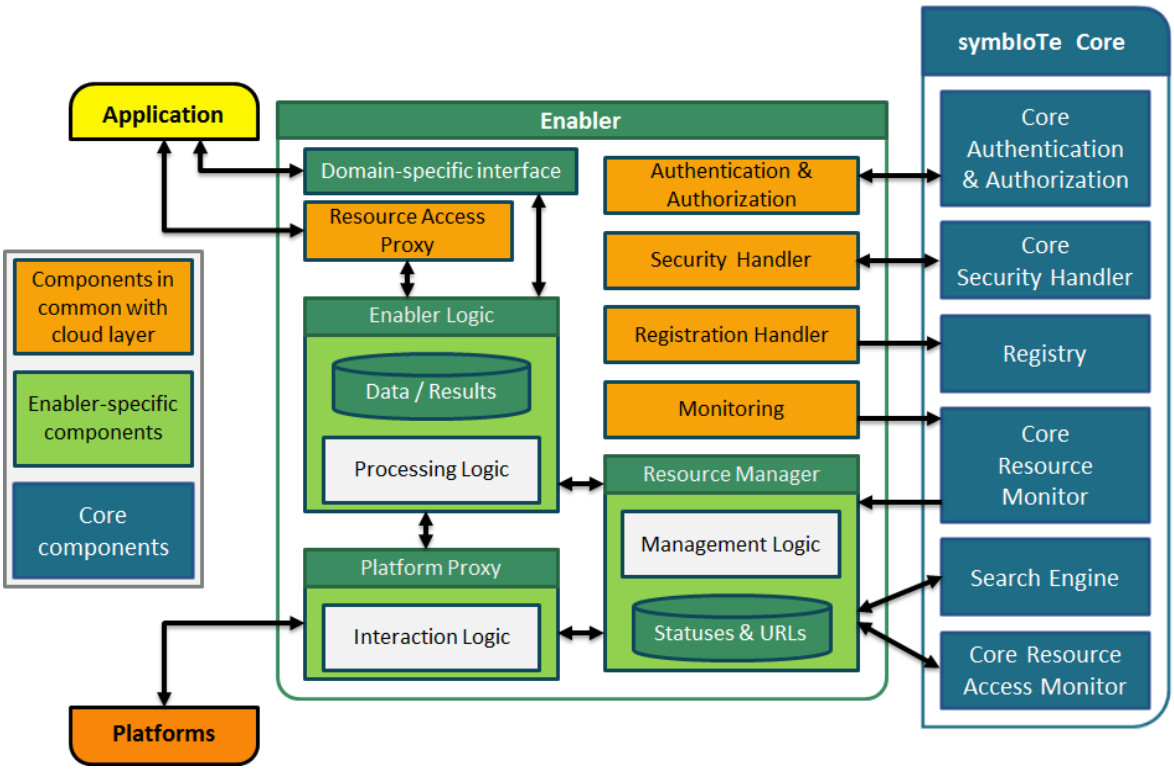

Figure 1. Enablers' Generic Architecture

Enablers' Generic Architecture is composed of two types of components: components in common with the Cloud Domain (marked in orange), and Enabler-specific components (marked in green). Each Enabler handles two types of resources since it acts both as a symbIoTe application and an IoT platform. Enabler Resources are resources that the Enabler offers to applications (in its role of a "virtual" IoT platform), and Underlying Resources are defined as resources that the Enabler uses from the underlying symbIoTe-complaint IoT platforms. Enabler Resources are created based on the Underlying Resources.

Components in common with the symbIoTe Cloud Domain are generic components for each Enabler. These are: Resource Access Proxy (RAP), Authentication & Authorization (AAM), Registration Handler (RH), and Monitoring. RAP serves as an access point for applications to use resources exposed by the Enabler. AAM is responsible for authenticating and authorizing users, applications and Enabler components. RH registers resources exposed by the Enabler to symbIoTe Core, while Monitoring tracks the availability and usage of the exposed resources.

Enabler-specific components implement domain-specific functionalities. These components are Resource Manager, Platform Proxy and Enabler Logic. Developers wanting to facilitate the usage of symbIoTe system for their domain-specific applications should be able to create those components according to their requirements.

Resource Manager is responsible to find the IoT resources using the symbIoTe Core Services which are required by an Enabler to provide its value-added services. Domain-specific functionality of this component relates to the process of finding adequate Underlying resources and replacing them with others when some resources become unavailable. This is done using the symbIoTe Core Services.

Platform proxy is used to access Underlying resources exposed by IoT platforms and found by the Resource Manager. A domain-specific functionality of this component relates to the interaction logic with those resources, for the proxy can retrieve data from underlying IoT every 10 seconds).

Enabler Logic is an Enabler-specific component responsible for specifying the type of resources that will be found by the Resource Manager, accessed by the Platform Proxy and offered to applications. It also integrates domain-specific logic, e.g., functions for processing the retrieved data (e.g. data aggregation), statistical operations and similar. These functionalities should be customized for each specific Enabler.

We describe Enabler architecture in more detail in the following subsections. Section 3.1 presents Enabler-specific Interface exposed to applications. The Enabler components are described in Section 3.2. Sequence diagrams showing the interaction between these components, as well as interaction with components from other domains, are presented in Section 3.3.

3.1 Enabler-specific Interface

For communication with Applications, Enabler-specific Interface is specified, composed of the RAP Interface and a Domain-Specific Interface. RAP offers the same functionality as the component with the same name in the Cloud domain; it allows symbIoTe applications to access the wanted Enabler resources (services) through the interface defined in D2.5 [7]. When accessing Enabler resources through RAP, the communication goes through symbIoTe Core (as in L1 Compliance) and Interworking Interface. In that way, symbIoTe Core is informed of the usage of Enabler resources. Domain-specific Interface provides additional domain-specific functionalities, mainly for accessing the Enabler Logic and data store. When using the Domain-specific Interface, communication between an application and the Enabler is direct (it does not use the Interworking Interface). Enabler-specific Interface is exposed to application developers so that they can use it within their applications. Further details regarding access to Enabler features are presented in Sequence diagrams in Sections 3.3.13 and **Error! Reference source not found.**.

3.2 Enablers' Components

In this subsection, we provide an overview of all developed Enabler components and definition of component interfaces. Components in common with the symbIoTe Cloud Domain are presented first. These are:

- Registration Handler (RH),
- Authentication and Authorization Manager (AAM),

- Resource Access Proxy (RAP),
- Monitoring, and

Afterwards, Enabler-specific components are described:

- Resource Manager,
- Enabler Logic, and
- Platform Proxy.

Security Handler is a library for accessing IoT platform resources by Platform Proxy and its usage is shown in the diagrams in section 3.3.

3.2.1 Registration Handler

Table 1. Registration Handler overview

Component	**Registration Handler**
Description	This component provides similar functionalities to the platform-side Registration Handler (CLD), apart from the support for IoT federations. Federations between enablers or enablers and platforms are not considered. The component allows sharing of IoT resources published by Enablers with other application developers by registering them at symbIoTe Core. Furthermore, the component enables registration updates.
Provided functionalities	• Registers resources to the symbIoTe Core using the symbIoTe Core Information Model and Best-Practice Information Model • Updates resource status and unregisters resources • Registers security info • Handles configuration of the exposed resource • Synchronizes the information with symbIoTe Core Services
Relation to other components	Registry (at symbIoTe Core level): To register the resources in the Registry Security Handler (Enabler): To retrieve the necessary core tokens and communicate securely with the symbIoTe Core Resource Access Proxy (Enabler): To register the resource offered by the Enabler including the access policy to access the offered resources Enabler Logic: Initiates registration of resources (sensors, actuators or services).
Related use cases	ALL
Related requirements	22, 34, 39, 45

Table 2. Registration handler interfaces

#	Interface	Message Type	From	Msg Consumers	Address/Queue/Exchange type	Payload[1]	Description
1	Resource registration	RabbitMQ	RH	INTERWOKING INTERFACE	*Exchange*: symbIoTe.InterworkingInterface *routing key*: symbIoTe.InterworkingInterface.registrationHandler.register_resources Routing type: RPC	Resource information	Event requesting registration of new resource
2	Resource unregistration	RabbitMQ	RH	INTERWOKING INTERFACE	Exchange: symbIoTe.InterworkingInterface routing key:symbIoTe.InterworkingInterface.registrationHandler.unregister_resources Routing type: RPC	Resource id	Event requesting unregistration of existing resource
3	Resource update	RabbitMQ	RH	INTERWOKING INTERFACE	Exchange: symbIoTe.InterworkingInterface routing key: symbIoTe.InterworkingInterface.registrationHandler.update_resources Routing type: RPC	Resource information	Event requesting update of existing resource
4	Resource registration	RabbitMQ	RH	RAP	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.rap.register_resources Routing type: routing-direct	Resource information	Event requesting registration of new resource
5	Resource unregistration	RabbitMQ	RH	RAP	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.rap.unregister_resources Routing type: routing-direct	Resource id	Event requesting unregistration of existing resource
6	Resource update	RabbitMQ	RH	RAP	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.rap.update_resources Routing type: routing-direct	Resource information	Event requesting update of existing resource
7	Resource registration	RabbitMQ	RH	MONITORING	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.monitor.register_resources Routing type: routing-direct	Resource information	Event requesting registration of new resource
8	Resource unregistration	RabbitMQ	RH	MONITORING	Exchange: symbIoTe.platformExchange Routing key:	Resource id	Event requesting unregistration of existing

[1] Payloads descriptions are available in Appendix, chapter **Error! Reference source not found.**

					symbIoTe.platformexchange.rh.monitor.unregister_resources Routing type: routing-direct		resource
9	Resource update	RabbitMQ	RH	MONITORING	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.monitor.update_resources Routing key: routing-direct	Resource information	Event requesting update of existing resource
10	Resources Information	REST	Platform Owner	RH	GET /resources	None	Get the list of resources which have been registered to the Core
11	Resources Information	REST	Platform Owner	RH	GET /resource/{internalResourceId}	None	Get information of a particular resource given its platform's internal ID
12	Resource registration	REST	Platform Owner	RH	POST /resource	Resource information	Registers a resource in the core
13	Resource registration	REST	Platform Owner	RH	POST /resources	List of resources information	Register a list of resources in the core
14	Resource update	REST	Platform Owner	RH	PUT /resource	Resource information	Updates the information about a resource in the core
15	Resource update	REST	Platform Owner	RH	PUT /resources	List of resources sinformation	Updates the information of several resources in the core
16	Resource unregistration	REST	Platform Owner	RH	DELETE /resource/{internalResourceId}	None	Unregister a resource on the core
17	Resource unregistration	REST	Platform Owner	RH	DELETE /resources	None	Unregister a list of resources on the core

3.2.2 Authentication and Authorization Manager

Table 3. Authentication and Authorization Manager overview

Component	**Authentication and Authorization Manager**
Description	This component provides similar functionalities as the platform-side Authentication and Authorization Manager component located in CLD.
Provided functionalities	• Authenticates native applications registered in the Enabler's space, and provides home tokens containing attributes in the Enabler's space • Enables sign out functionality for applications registered in the Enabler's space • Checks any asynchronous revocation of home tokens when polled by external AAMs, by managing a "Token Revocation List" • Checks the validity of foreign tokens or core tokens provided by applications that are not natively registered in the Enabler's space • Performs the "Attributes Mapping Function" for applications that are not natively registered in the Enabler's space and would like to access resources in the Enabler's space • Generates foreign tokens for applications that are not natively registered in the Enabler's space
Relation to other components	Security Handler (Enabler): To check home token revocation in Enabler's components (e.g., Registration Handler, Resource Access Proxy) and to authenticate applications registered in the Enabler's space Core Authentication & Authorization Manager (APP): To check home token revocation in Enabler's components (e.g., Registration Handler, Resource Access Proxy) Platform Authentication & Authorization Manager (CLD): To check home token revocation in Enabler's components (e.g., Registration Handler, Resource Access Proxy)
Related use cases	ALL
Related requirements	35, 37

Table 4. Authentication and Authorization Manager interfaces

Operation	From	Type	Request	Response
Get available AAMs	REST client	GET	empty	ResponseEntity<Set<AAM>> with code HTTP:200 on ok and HTTP:500 on error
Get Component certificate	REST client	GET	empty	CA certificate in PEM String format
Manage platform	Administration	AMQP	Platform Management Request	Platform Management Response
get owned platforms instances' details	Administration	AMQP	Platform Owner's Token String	OwnedPlatformDetails
Manage user (application / platform owner)	Administration	AMQP	User Management Request	User Management Response
Get client certificate	REST client	POST	ClientCertificateRequest: - username - password - client identifier - Base64 encoded String with certificate signing request	Success: Base64 encoded String with PEM certificate Error: error status
Get Home Token	Administration	AMQP	Signed LoginRequest	token String issued for that client (user/platform owner client)
Get Home Token	REST client	POST	Signed LoginRequest	Headers with X-Auth-token containing token String for that client
Get Roamed / federated / foreign Token	REST client	POST	Headers with: X-Auth-token containing HOME token String for that client; (opt) X-Auth-Cert containing Base64 encoded PEM Certificate String matching SPK from token	Headers with X-Auth-token containing FOREIGN/ROAMED/FEDERATED token String for that client
Get Guest Token	REST client	GET	empty	Headers with X-Auth-token containing GUEST token String
validation	symbIoTe components	AMQP	JWS token String (opt) Base64 encoded PEM Certificate String matching SPK from token	ValidationStatus
validation	REST client	POST	Headers with: X-Auth-token containing HOME token String for that client; (opt) X-Auth-Cert containing Base64 encoded PEM Certificate String matching SPK from token	ValidationStatus
Revoke token	REST client	POST	X-Auth-token containing HOME token String for that client, Username, Password	{"status": <success/failure>}
Revoke certificate	REST client	POST	Username, Password, client_id	{"status": <success/failure>}
Revoke certificate	Administration	AMQP	AAM admin credentials, Base64 encoded PEM Certificate String issued by this AAM	{"status": <success/failure>}
Released attributes provisioning	Administration	AMQP	WIP	{"status": <success/failure>}
Attributes mapping provisioning	Administration	AMQP	WIP	{"status": <success/failure>}

ew

s functionalities similar to those offered by the platform-side Resource Access Proxy component (CLD) developed for L1. The ediator between Enabler Logic and the application. The presence of this component is necessary to make the Enabler allowing the applications to access the resources offered by Enablers in the same way as those offered by L1 IoT Platforms.
as a service exposed by the Enabler pplication starts or stops using Enabler resources
cation: It accesses Resource Access Proxy to access/use Enabler resources abler): Informs Resource Access Proxy of the registered resources that it should provide access to r): Verifies tokens source Access Proxy notifies Monitoring when applications start or stop using resources Contacts Resource Access Proxy to check the availability/status of Enabler resources Monitor: Resource Access Proxy emits resource usage and notifies the Core Access Resource Monitor when a resource is ld be more the one Enabler Logic in one Enabler. If one Enabler Logic needs data from another one, then it sends application te Enabler Logic which implements a RAP plugin.

Table 6. Resource Access Proxy interfaces

#	Interface	Name	Message Type	From	Msg Consumers	Address/Queue	Payload[2]	Description
1	Resource registration	symbIoTe.rap.registrationHandler.register_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.rap Routing key: routing-direct	Resource information	Event requesting registration of new resource
2	Resource registration	symbIoTe.rap.registrationHandler.unregister_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.rap Routing key: routing-direct	Resource id	Event requesting unregistration of existing resource
3	Resource registration	symbIoTe.rap.registrationHandler.update_resources	RabbitMQ	RH	RAP	Exchange: symbIoTe.rap Routing key: routing-direct	Resource information	Event requesting update of existing resource
4a	Resource access read	/rap/Sensor/{resourceId}	REST	Interworking Interface	RAP	GET	None - replies with the value of the resource	Event reading the value of a resource
4b	Resource access read	/rap/Sensors({resourceId})/Observations?$top=1	OData	Interworking Interface	RAP	GET	None - replies with the value of the resource	Event reading the value of a resource
5a	Resource access read history	/rap/Sensor/{resourceId}/history	REST	Interworking Interface	RAP	GET	None - replies with the history values of the resource	Event reading the value of a resource
5b	Resource access read history	/rap/Sensors({resourceId})/Observations	OData	Interworking Interface	RAP	GET	None - replies with the history values of the resource	Event reading the value of a resource
6a	Resource access write	/rap/Service/{resourceId}	REST	Interworking Interface	RAP	POST	{ “inputParameters”: [{ “name”: “param_name”, “value”: “param_value” }, ..] }	Event writing the value of a resource
6b	Resource access write	/rap/ActuatingService{resourceId}	OData	Interworking Interface	RAP	PUT	{ “inputParameters”: [{ “name”: “param_name”, “value”: “param_value”	Event writing the value of a resource

[2] Payloads descriptions are available in Appendix, chapter **Error! Reference source not found.**

							}, ..] }	
7	Resource notifications	/notification	WebSocket	Interworking Interface	RAP	client / server	the value of the resource	Event reading the value of a resource

3.2.4 Monitoring

Table 7. Monitoring overview

Component	**Monitoring**
Description	This component provides features similar to those offered by the platform-side Monitoring component (CLD). It monitors the load of IoT resources (services) offered by an Enabler and the usage of registered services by applications. Monitoring information can be used to estimate service popularity (useful for ranking). The component must monitor the quality of the offered services so as to make sure that the advertised quality of service is met.
Provided functionalities	• Checks load/availability of resources registered by an Enabler • Records the start and end of access to a resource
Relation to other components	Security Handler (Enabler): To request core token Resource Manager (Enabler): To notify Resource Manager about the status/availability/performance of Enabler resources Core Resource Access Monitor: To send usage report for the purpose of ranking Enabler resources by symbIoTe Core Services
Related use cases	ALL
Related requirements	6, 81

Table 8. Monitoring interfaces

#	Interface	Message Type	From	Msg Consumers	Address/Queue/	Payload[3]	Description
1	Resource registration	RabbitMQ	RH	MONITORING	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.monitor.register_resources Routing type: routing-direct	Resource information	Event requesting registration of new resource
2	Resource unregistration	RabbitMQ	RH	MONITORING	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.monitor.unregister_resources Routing type: routing-direct	Resource id	Event requesting unregistration of existing resource
3	Resource update	RabbitMQ	RH	MONITORING	Exchange: symbIoTe.platformExchange Routing key: symbIoTe.platformexchange.rh.monitor.update_resources Routing key: routing-direct	Resource information	Event requesting update of existing resource

[3] Payloads descriptions are available in Appendix, chapter **Error! Reference source not found.**

3.2.5 Resource Manager

Table 9. Resource Manager overview

Component	**Resource Manager**
Description	This component manages the Underlying IoT resources used by Enabler. It receives input from the Enabler Monitoring component about the status/availability/performance of the resources offered to the Enabler (Underlying IoT Resources). If the advertised quality of service is not met by the Underlying IoT resources (e.g., some sensors are offline on the platform side), this component is responsible to automatically discover alternative platform resources and starts using them instead. The component queries the symbIoTe Core Search Engine to discover new resources which match certain criteria required by Enabler services. Furthermore, it filters the results returned by the Search Engine according to the Enabler domain-specific requirements. Additionally, the component keeps track of IoT resources assigned to the Enabler during the mediation process, e.g., when an enabler has identified, requested and has been granted access to IoT resources for the intended value-added service. The usage of new resources may also be facilitated by an automated payment system to support access to private resources that are not exposed to everyone. The Resource Manager should also periodically search for resources matching its needs and if it finds more suitable ones (e.g., cheaper or more accurate), it should replace the old ones with the new ones.
Provided functionalities	• Supports resource discovery • Ranks resources relevant to the Enabler's needs. The Enabler might have different criteria compared to the symbIoTe ranking engine • Supports automated filtering of resources • Periodically searches for more suitable resources in the symbIoTe Core Registry
Relation to other components	Enabler's Logic: The Enabler's Logic provides information about the characteristics of resources required by the Enabler to the Resource Manager Monitoring: The monitoring notifies the Resource Manager about the status/availability/performance of used resources Search Engine: Finds resources in the symbIoTe Core Registry. Platform Proxy: Resource Manager sends requests for data acquisition to Platform Proxy.
Related use cases	ALL
Related requirements	6, 20, 21, 82

Example 1:
An Enabler claims to provide temperature sensors in all the European capitals. However, the temperature sensors used to provide temperature information in Zagreb suddenly become unavailable. The Resource Manager should be notified about this incident in order to automatically search and subscribe to temperature sensors offered by other platforms in Zagreb.

Table 10. Resource Manager interfaces

#	Interface	Name	Msg type	From	Msg Consumers	Address/Queue	Payload	Description
1	Start Task Acquisition		RabbitMQ RPC	Enabler Logic	Resource Manager	Exchange: symbIoTe.resourceManager Routing key: symbIoTe.resourceManager.startDataAcquisition	Array of Task Information requests containing: • taskId • minimum number of resources • Parameterized Query (i.e. CoreQueryRequest) • queryInterval • boolean for allowing caching of resource ids • caching interval • boolean for passing the data to the Platform Proxy • the name of the enabler logic which made the request • the SPARQL query request Returns: Same info along with: • the list of resource ids • the status	Request for acquisition of sensor data for a new task
2	Update Task		RabbitMQ RPC	Enabler Logic	Resource Manager	Exchange: symbIoTe.resourceManager Routing key: symbIoTe.resourceManager.updateTask	Array of Task Information requests containing: • taskId • minimum number of resources • Parameterized Query (i.e. CoreQueryRequest) • queryInterval • boolean for allowing caching of resource ids • caching interval	Request for updating the task information

							• boolean for passing the data to the Platform Proxy • the name of the enabler logic which made the request • the SPARQL query request Returns: Same info along with: • the list of resource ids the status	
3	Cancel Task		RabbitMQ RPC	Enabler Logic	Resource Manager	Exchange: symbIoTe.resourceManager Routing key: symbIoTe.resourceManager.cancelTask	List of task ids to be deleted Returns: • status • message	Request for canceling a task
4	Unavailable Resources		RabbitMQ (async)	Platform Proxy	Resource Manager	Exchange: symbIoTe.resourceManager Routing key: symbIoTe.resourceManager.unavailableResources	ProblematicResourcesInfo containing: • TaskId of the problematic resources • Resource ids	Platform Proxy informs Resource Manager for unavailable resources
5	Wrong Data		RabbitMQ (async)	Enabler Logic	Resource Manager	Exchange: symbIoTe.resourceManager Routing key: symbIoTe.resourceManager.wrongData	ProblematicResourcesInfo containing: • TaskId of the problematic resources • Resource ids	Enabler Logic informs Resource Manager for resources with wrong data

3.2.6 Enabler Logic

Table 11. Enabler Logic overview

Component	**Enabler Logic**
Description	This component implements the minimal application domain logic. In the simplest case this corresponds to the mere aggregation of IoT resources from multiple IoT platforms. More advanced processing can be applied to support value-added services corresponding to the requirements of a specific domain. The component is domain-specific and its additional functionalities should be implemented by the developers of the Enabler in a class implementing the Processing Logic interface. It is responsible for notifying the Resource Manager about the characteristics of required resources as well as for accessing and storing resource data (in case of sensors). Furthermore, it groups platform resources in virtual resources (if necessary), processes acquired sensor data and offers more specialized services (e.g., weather forecast, access to historical data, statistical analysis, etc.). Finally, it notifies the Enabler's Registration Handler about resources which should be registered in symbIoTe Core as services offered by an Enabler to third-party applications. It also initiates the update of registered resources. There could be more than a single Enabler Logic component integrated within a single Enabler.
Provided functionalities	• Defines the type and amount of Underlying IoT resources required • Groups platform resources in virtual resources • Provides data storage for acquired sensor data • Processes the data and offers the possibility to develop specialized value-added services
Relation to other components	Resource Manager (Enabler): The Resource Manager receives the description of the resource type and number from Enabler Logic Registration Handler (Enabler): The Enabler Logic specifies the type of resources which are registered in symbIoTe Core as services offered by the Enabler to third parties Resource Access Proxy (IoT Platforms): Requests from applications are forwarded to Enabler Logic. Enabler Logic returns response through Resource Access Proxy Domain-specific Interface: Requests from applications are forwarded to Enabler Logic. Enabler Logic returns responses through the Domain-specific Interface
Related use cases	ALL
Related requirements	20, 21, 39, 45, 83

Example 2:
An Enabler offers temperature readings for all European capitals. Enabler logic should continuously receive data from selected temperature sensors in European capitals (found by Resource Manager) calculate the average and store it in its internal database. In an application, a user wants to access average temperature in Zagreb for the last three days. Enabler Logic will receive request for temperature in Zagreb (indirectly over Domain-Specific Interface or Resource Access Proxy), read calculated average form internal database and send the response to the application indirectly over Domain-Specific Interface or Resource Access Proxy.

Table 12. Enabler Logic interfaces

#	Interface	Name	Msg type	From	Msg Consumers	Address/Queue	Payload	Description
1	Acquire measurements		RabbitMQ RPC	RAP/ Domain-Specific Interface	Enabler Logic	Exchange: symbIoTe.enablerLogic Routing key: symbIoTe .enablerLogic .acquireMeasurements		Request for acquisition of sensor data
2	Receive acquired data		RabbitMQ (async)	Platform Proxy	Enabler Logic	Exchange: symbIoTe.enablerLogic Routing key: symbIoTe .enablerLogic .dataAppeared	taskID, timestamp, observations list	Platform Proxy forwards received resource data to Enabler Logic
3	RAP plugin – access read resource		RabbitMQ RPC	RAP/ Domain-Specific Interface	Enabler Logic	Exchange: plugin-exchange Routing key: symbIoTe {EnablerName}.get	Resource info with simbioteId, interlanlId and observed properties Returns: observation	RAP/DSI asks plugin for value of resource
4	RAP plugin – access read resource history		RabbitMQ RPC	RAP/ Domain-Specific Interface	Enabler Logic	Exchange: plugin-exchange Routing key: symbIoTe {EnablerName}.history	Resource info with simbioteId, interlanlId, observed properties and constraints Returns: observations	RAP/DSI asks plugin for historical values of resource
5	RAP plugin – access to set resource or call service		RabbitMQ RPC	RAP/ Domain-Specific Interface	Enabler Logic	Exchange: plugin-exchange Routing key: symbIoTe {EnablerName}.set	Resource info with simbioteId, interlanlId, observed properties and parameters Returns: result	RAP/DSI asks plugin to write values of resource or RAP/DSI asks plugin to call service with parameters and enabler logic returns service result
6	Receiving synchronous request form another enabler logic		RabbitMQ RPC	Another enabler logic	Enabler Logic	Exchange: symbIoTe.enablerLogic Routing key: symbIoTe .enablerLogic .syncMessageToEnablerLogic	Parameter message Returns: result	Another enabler logic send synchronous message to this enabler logic

7	Receiving asynchronous request form another enabler logic		RabbitMQ (async)	Another enabler logic	Enabler Logic	Exchange: symbIoTe.enablerLogic Routing key: symbIoTe .enablerLogic .asyncMessageToEnablerLogic	Parameter message	Another enabler logic send asynchronous message to this enabler logic
	Resources Updated		RabbitMQ (async)	Resource Manager	Specific Enabler Logic	Exchange: symbIoTe.enablerLogic Routing key: symbIoTe.enablerLogic.resourcesUpdated.{specific enabler logic name which owns the task}	ResourcesUpdated containing: • task id new Resource List	Resource Manager informs a specific enabler logic which owns the task about new resources
	Not enough available resources		RabbitMQ (async)	Resource Manager	Specific Enabler Logic	Exchange: symbIoTe.enablerLogic Routing key: symbIoTe.enablerLogic.notEnoughResources.{specific enabler logic name which owns the task}	NotEnoughResourcesAvailable containing: • task id number of resources acquired	Resource Manager informs a specific enabler logic which owns the task about tasks with not enough resources

3.2.7 Platform Proxy

Table 13. Platform Proxy overview

Component	**Platform Proxy**
Description	This component is responsible for accessing IoT Platforms to gather data specified by Enabler Logic, and found by Resource Manager. Access to platform resources is initiated by the Enabler Logic. Sensor data can be acquired either by a push or pull mechanism supported by symbIoTe. The received data is forwarded to Enabler Logic where it is stored and processed further. This component is also responsible for access to actuators. When Enabler Logic needs to actuate some resources, it will send a message to Platform Proxy that will contact appropriate IoT Platforms.
Provided functionalities	• Sends requests to IoT Platforms to access and use resources • Forwards acquired data from sensors to Enabler Logic • Offers the possibility to develop mechanisms for accessing IoT Platforms based on domain-specific needs
Relation to other components	Resource Manager (Enabler): The Resource Manager sends the description of the required resources (resourceIds, acquisition frequency, …) to Platform Proxy. Enabler Logic: Platform Proxy acquires data from resources and forwards it to Enabler Logic. Enabler Logic can send request for particular resource data or resource actuation then Platform Proxy make request to specific resource and returns it to Enabler Logic.
Related use cases	ALL
Related requirements	20, 21

Example 3:
An Enabler offers temperature readings for all European capitals. In an application, a user wants to receive updated values every five minutes from two nearest sensors. Platform Proxy should acquire these measurements from sensors whose characteristics are specified by Enabler Logic, and found by Resource Manager.

Table 14. Platform Proxy interfaces

#	Interface	Name	Msg type	From	Msg Consumers	Address/Queue	Payload	Description
1	Start Resource Acquisition	symbIoTe.enablerPlatformProxy.acquisitionStartRequested	RPC	Resource Manager	Platform Proxy	Exchange: symbIoTe.enablerPlatformProxy Routing key: same as name	PlatformProxyAcquisitionStartRequest Containing task id, list of resources, name of enabler logic and query interval	Starts a periodic acquisition of data for a task
2	Start Resource Acquisition response	-	RPC response	Platform Proxy	Resource Manager	Answer to the temporary RPC response queue	Schedule status and taskId	Response containing status of starting data acquisition
3	Stop resource acquisition	symbIoTe.enablerPlatformProxy.acquisitionStopRequested	RPC	Resource Manager	Platform Proxy	Exchange: symbIoTe.enablerPlatformProxy Routing key: same as name	List of taskIds	Stops the acquisition task for specified task Ids
4	Stop resource acquisition response	-	RPC response	Platform Proxy	Resource Manager	Answer to the temporary RPC response queue	Cancellation message	Simple cancelation message

3.3 Sequence diagrams

The workflows defined for symbIoTe Enablers are the following:

- Enabler Registration
- Enabler Resource Registration
- Enabler Resource Unregistration
- Enabler Resource Update
- Enabler Resource Availability Reporting
- Scheduled Monitoring of Enabler Resources
- Start Data Acquisition
- Search
- Stop Data Acquisition
- Data Acquisition
- Unresponding Resource During Data Acquisition
- Replacement of a Malfunctioning Resource
- Access to Enabler Resource Using Enabler RAP
- Access to Enabler Resource Using its Domain-specific Interface
- Reporting of Enabler Resource Usage

Hereafter all functional capabilities are presented in the form of UML sequence diagrams, with detailed description of the exchanged messages. Figure 2 shows the legend for messages used in the diagrams: mandatory messages, optional messages and mandatory interactions. A “mandatory interaction” is referred to as a “Procedure” because it presents a sequence of messages being exchanged between components. These “Procedures” are mainly related to security mechanisms.

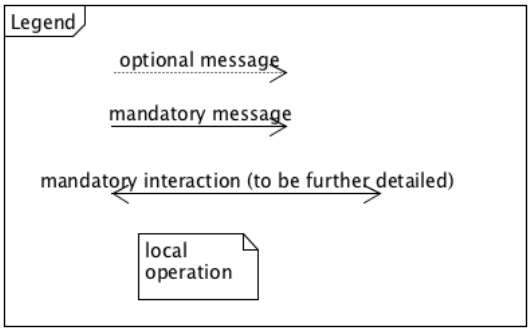

Figure 2. Legend of messages used in sequence diagrams

Components in sequence diagrams are color-coded so that each color is assigned to a specific domain in the symbIoTe architecture:

- Yellow: Enablers

- Green: Application / other Enabler
- Blue: symbIoTe Core Services
- Orange: IoT Platform Cloud

3.3.1 Enabler registration

Before entering the symbIoTe ecosystem, each Enabler is obliged to register to symbIoTe Core. The process is handled by Enabler owner using the symbIoTe Core Administration application.

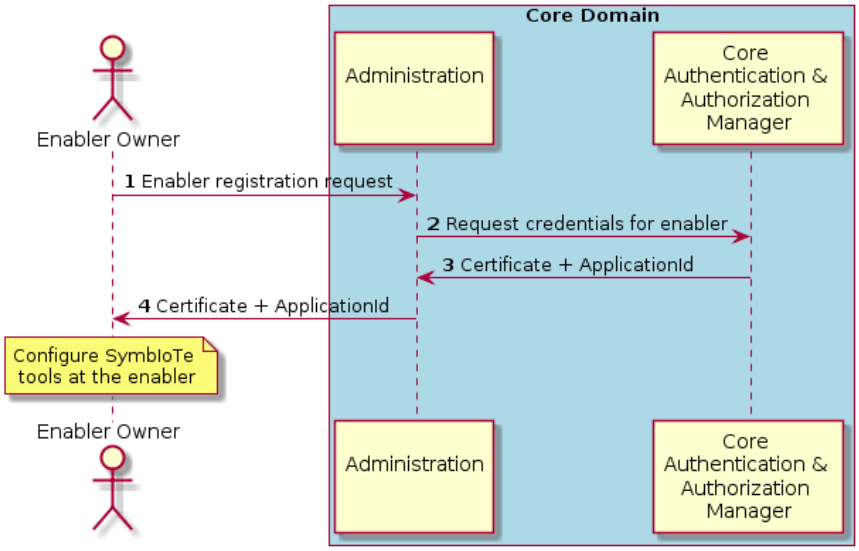

Figure 3. Enabler registration

Description:

- Message 1: Enabler owner sends a registration request to symbIoTe by using the Administration web application. The request is either for a trail or normal registration.
- Message 2: Administration sends request to the Core Authentication and Authorization Manager, which requests credentials for the Enabler.
- Message 3: Core Authentication and Authorization Manager returns the generated certificate and applicationId to Administration.
- Message 4: Administration web application returns certificate and applicationId to Enabler owner. The Enabler owner can subsequently configure the Enabler to become L1 Compliant.

3.3.2 Enabler resource registration

Enabler registers its resources in symbIoTe Core so that they can be found by other applications.

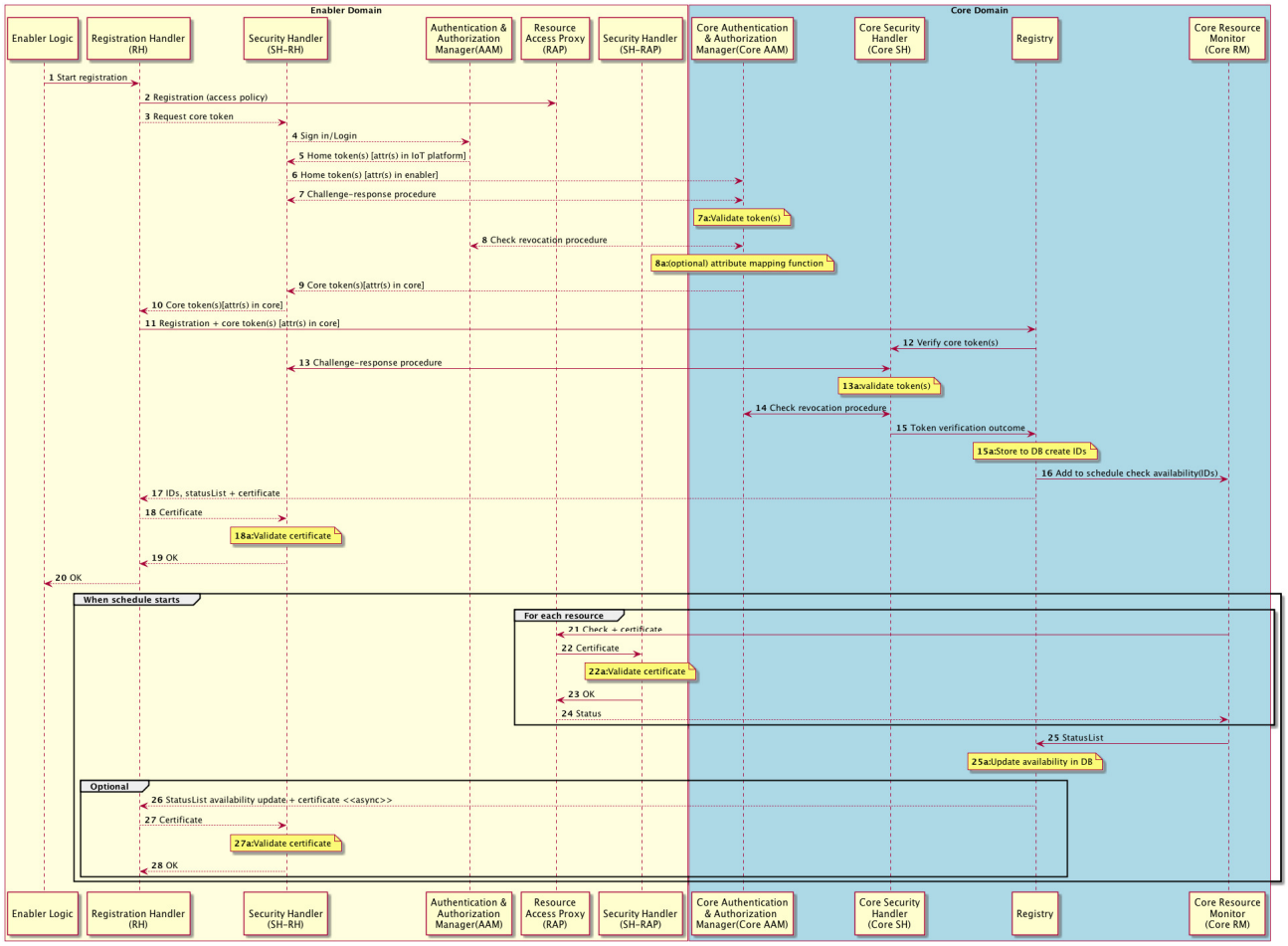

Figure 4. Enabler resource registration

Description:

- Message 1: Registration is initiated by Enabler Logic.
- Message 2: generated by the Registration Handler and sent to the Resource Access Proxy in the same Enabler. It is used to register a resource on the Resource Access Proxy, along with an access policy specifying rules for resource access;
- Message 3 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, this step is not needed.
- Message 4 (optional): generated by the Security Handler and sent to the home (enabler) AAM in which the Registration Handler is registered. It is used to authenticate the Registration Handler. If the Registration Handler is already logged in, this step is not needed.
- Message 5 (optional): generated by the home (Enabler) AAM and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 6 (optional) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the Core Domain. It is used to trigger the operations for

obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.

- Procedure 7 (optional) (SecurityInterface): procedure that allows the Security Handler acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 7a (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8a (optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the home AAM (Enabler) in a new set of attributes that it has in the Core Domain. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 9 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 10 (optional): generated by the Security Handler and sent to the Registration Handler. It is used to forward the core token generated at the previous step.
- Message 11 (RegPlatformInterface): generated by the Registration Handler and sent to the Registry. Its main purpose is to provide the metadata describing a resource or a set of resources which the Enabler exposes to the Registry. In addition to the registration message, it also provides the core token(s) containing the attributes assigned to the Registration Handler and resource access policies.
- Message 12: generated by the Registry and sent to the Core Security Handler. It is used to ask the Security Handler to verify token validity.
- Procedure 13 (SecurityInterface): procedure that allows the Security Handler acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).
- Procedure 13a: verification of time validity, authenticity and integrity of the provided token(s).
- Procedure 14: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 15: generated by the Security Handler in the Core Domain and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 15a: stores registrations to database and generates ID for the resource

- Message 16: Registry sends a message to Core Resource Monitor to add a scheduled task to check availability of registered resources. Core Resource Monitor will in the future check resource availability (messages 21-24) and asynchronously inform Registry about availability with updated status list (message 25).
- Message 17: Registry returns: IDs of registered resources, status list and a certificate (used to demonstrate the identity of the entity generating the message, for authentication purposes, the certificate must be validated by the Security Handler of the component)
- Message 18: Registration Handler forwards the certificate to Security Handler for validation
- Procedure 18a: Security Handler validates the certificate
- Message 19: Security Handler returns status of validation
- Message 20: Registration Handler forwards status of validation to Enabler Logic.
- Messages 21-24 are used for checking the availability of registered resources
- Message 21 (AccessResourceInterface): Core Resource Monitor sends message to Resource Access Proxy in order to check availability. It includes the certificate as well (used to demonstrate the identity of the entity generating the message, for authentication purposes, the certificate must be validated by the Security Handler of the component).
- Message 22: Resource Access Proxy sends the certificate to Security Handler for validation
- Procedure 22a: Security Handler validates the certificate
- Message 23: Security Handler returns status of validation
- Message 24: Resource Access Proxy returns availability status
- Message 25: Core Resource Monitor collects all availability status, makes a status list and sends it to Registry
- Procedure 25a: Updates availability in database
- Message 26 (optional): Registry sends asynchronous message with availability list and the certificate to Registration Handler
- Message 27 (triggered by 25) (RegistrationHandlerInterface): Registration Handler forwards the certificate to Security Handler for validation
- Procedure 27a: Security Handler validates certificate
- Message 28: Security Handler returns status of validation

3.3.3 Enabler resource unregistration

Enabler unregisters its resource in the symbIoTe Core Domain that will no longer be offered to applications.

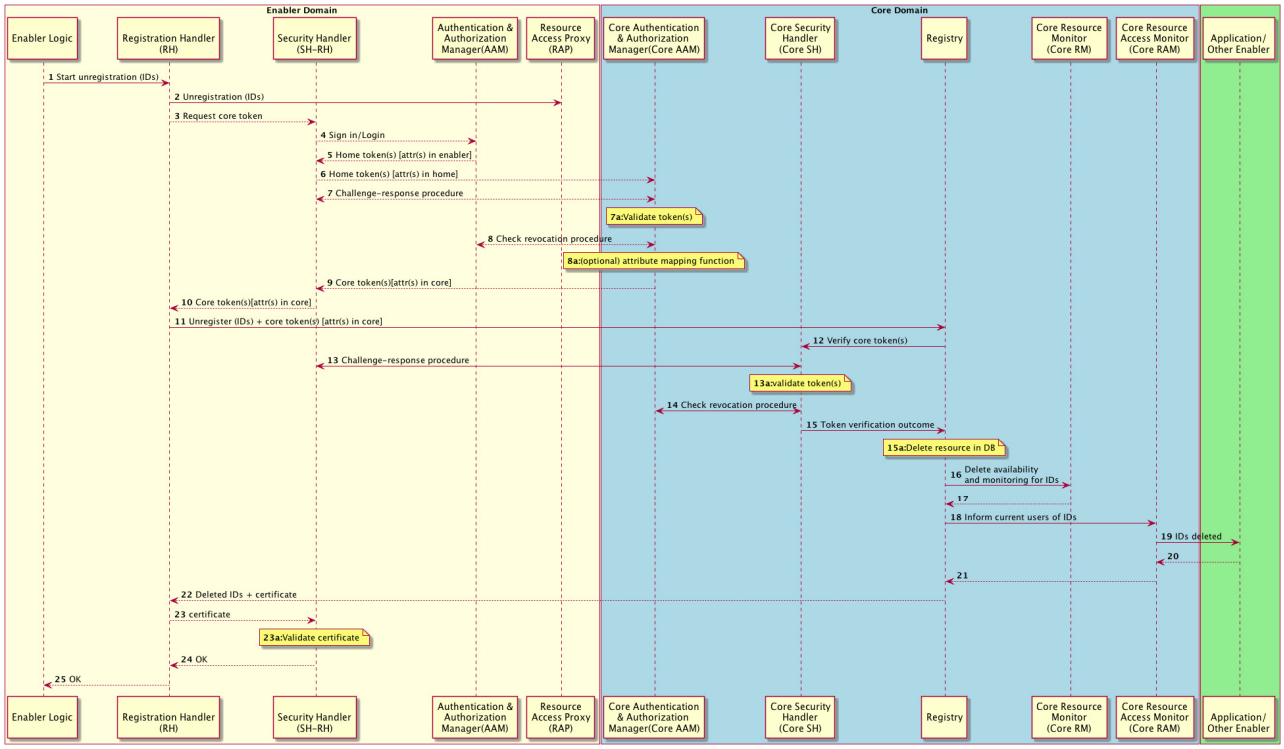

Figure 5. Enabler resource unregistration

Description:

- Message 1: Unregistration is initiated by Enabler Logic.
- Message 2: generated by the Registration Handler and sent to the Resource Access Proxy in the same Enabler. It is used to unregister the resource on the Resource Access Proxy;
- Message 3 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.
- Message 4 (optional): generated by the Security Handler and sent to the home AAM in which the Registration Handler is registered. It is used to authenticate the Registration Handler. If the Registration Handler is already logged in, it is not necessary.
- Message 5 (optional): generated by the home AAM in the Enabler and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 6 (optional) (Enabler AAInterface): generated by the Security Handler and sent to the Core AAM in the Core Domain. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 7 (optional) (Security Interface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.

- Procedure 7a (optional): verification of time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8a (optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the Enabler in a new set of attributes that it has in the Core Domain. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 9(optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 10 (optional): generated by the Security Handler and sent to the Registration Handler. It is used to forward the core token generated at the previous step.
- Message 11 (RegEnablerInterface): generated by the Registration Handler and sent to the Registry. It is used to provide, along with the unregistration message, the core token(s) containing the attributes assigned to the Registration Handler.
- Message 12: generated by the Registry and sent to the Core Security Handler. It is used to ask the Security Handler to verify the complete validity of the token.
- Procedure 13 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate it is the real owner of the token(s).
- Procedure 13a: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 14: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 15: generated by the Core Security Handler and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 15a: Registry deletes resource in database
- Message 16: Registry sends message to Core Resource Monitor to delete availability and cancel scheduled monitoring tasks
- Message 17: returns call
- Message 18: Registry informs Core Resource Access Monitor that specific source is unregistered and that the users of that resource need to be informed
- Message 19 (optional) (ApplicationInterface): Core Resource Access Monitor informs each reachable (open connection or registered endpoint) Application/Enabler that uses specific resource about deletion of resource

- Message 20 returns call
- Message 21 returns call
- Message 22: Registry returns deleted IDs and certificate to Registration Handler (certificate is used to demonstrate the identity of the entity generating the message, for authentication purposes, certificate must be validated by the Security Handler of the component)
- Message 23: Registration Handler forwards certificate to Security Handler for validation
- Procedure 23a: Security Handler validates certificate
- Message 24: Security Handler returns status of validation
- Message 25: Registration Handler forwards status of validation to Enabler Logic.

3.3.4 Enabler resource update

Enabler updates the resource exposed through symbIoTe Core. By doing that, it can be ensured that resource descriptions at the symbIoTe Core are up-to-date.

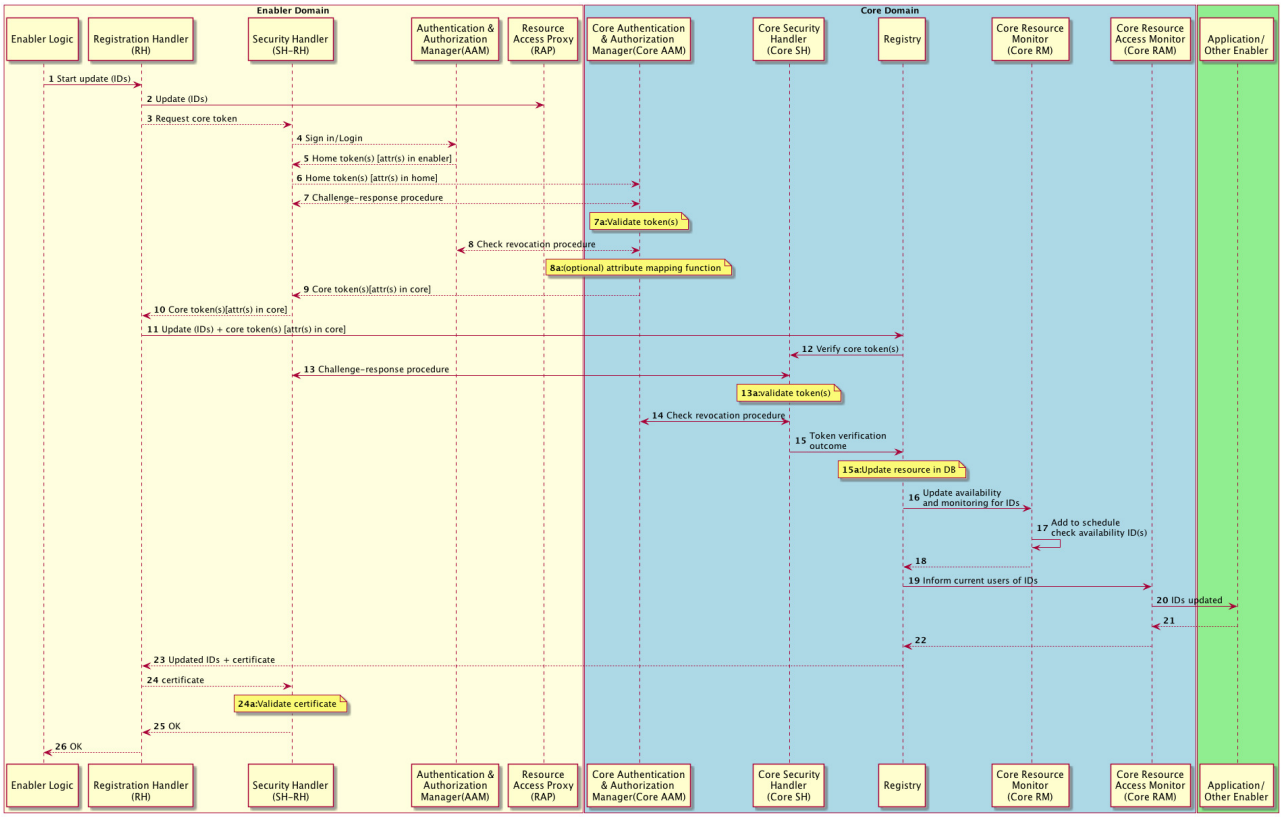

Figure 6. Enabler resource update

Description:

- Message 1: Update is initiated by Enabler Logic.
- Message 2: generated by the Registration Handler and sent to the Resource Access Proxy in the same Enabler. It is used to update the resource on the Resource Access Proxy;

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

- Message 13 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).
- Procedure 13a: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 14: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 15: generated by the Security Handler in the Core Domain and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 15a: Registry updates resource in database
- Message 16: Registry ends request to Core Resource Monitor to update availability and to schedule availability check
- Message 17: Core Resource Monitor schedules task for checking availability for specified resources
- Message 18: returns call
- Message 19: Registry sends message to Core Access Resource Access Monitor to inform current user of updated resources
- Message 20 (optional) (ApplicationInterface): Core Resource Access Monitor informs each reachable (open connection or registered endpoint) Application/Enabler that uses specific resource about resource update
- Message 21 returns call
- Message 22 returns call
- Message 23: Registry returns updated IDs including a certificate
- Message 24: Registration Handler forwards certificate to Security Handler for validation
- Procedure 24a: Security Handler validates certificate
- Message 25: Security Handler returns status of validation
- Message 26: Registration Handler forwards status of validation to Enabler Logic.

3.3.5 Enabler resource availability reporting

Enabler registers Enabler resources to Core Resource Monitor. Availability check will be initiated by Core Resource Monitor (described in Section 3.3.6).

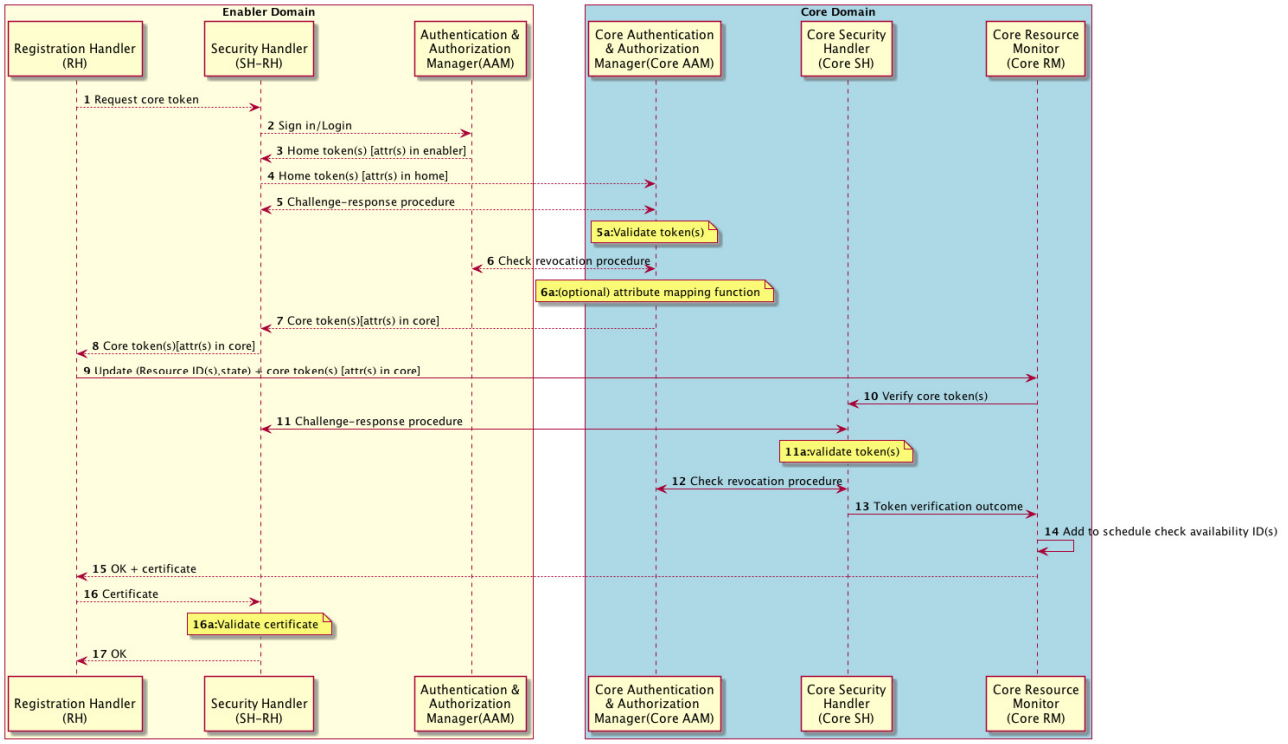

Figure 7. Enabler Resource availability reporting

Description:

- Message 1 (optional): generated by the Registration Handler and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Registration Handler is already logged in, it is not necessary.
- Message 2 (optional): generated by the Security Handler and sent to the home AAM in which the Registration Handler is registered. It is used to authenticate the Registration Handler. If the Registration Handler is already logged in, it is not necessary.
- Message 3 (optional): generated by the home AAM in the Enabler and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Registration Handler is already logged in, it is not necessary.
- Message 4 (optional) (EnablerAAInterface): generated by the Security Handler and sent to the Core AAM in the Core Domain. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 5 (optional) (SecurityInterface): procedure that allows the Security Handler acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 5a (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Registration Handler already has valid core token(s), it is not necessary.

- Message 6 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 6a (optional): procedure that, in case it is needed, translates the attributes that the Registration Handler has in the Enabler in a new set of attributes that it has in the Core Domain. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 7 (optional): generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 8 (MonitorResInterface): generated by the Registration Handler and sent to the Core Resource Monitor. It is used to provide, along with the update message, the core token(s) containing the attributes assigned to the Registration Handler.
- Message 9: generated by the Core Resource Monitor and sent to the Core Security Handler. It is used to ask the security handler to verify the complete validity of the token.
- Message 10 (SecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Registration Handler to demonstrate that it is the real owner of the token(s).
- Procedure 11: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 11a: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 12: generated by the Core Security Handler and sent to the Core Resource Monitor. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Messages 13: returns core token to Registration Handler
- Message 14: schedules task for checking availability of specified resources (IDs)
- Message 15: returns status of availability scheduling and certificate (used to demonstrate the identity of the entity generating the message, for authentication purposes, the certificate must be validated by the Security Handler of the component)
- Message 16: Registration Handler sends certificate to Security Handler for validation
- Procedure 16a: validate certificate
- Message 17: returns result of certificate validation

3.3.6 Scheduled monitoring of Enabler resources

Core Resource Monitor checks the availability of the resources exposed by Enabler and reports it to Registry.

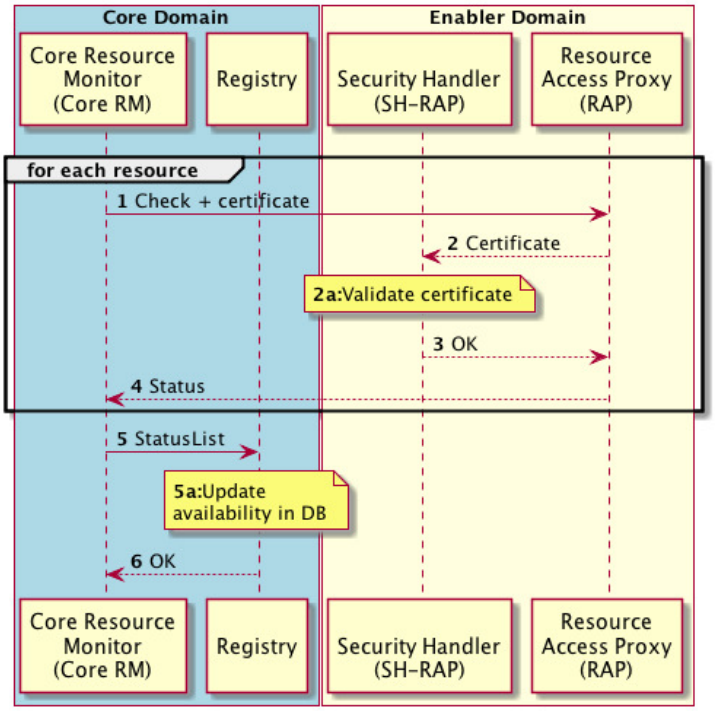

Figure 8. Scheduled monitoring of Enabler resources

Description:

- Message 1 (AccessResourceInterface): when the availability checking task is executed it starts with this message from Core Resource Monitor to Resource Access Proxy (includes certificate)
- Message 2: Resource Access Proxy sends certificate for validation to Security Handler
- Procedure 2a: validates certificate
- Message 3: returns result of certificate validation
- Message 4: Resource Access Proxy returns result of availability to Core Resource Monitor
- Message 5: Core Resource Monitor collects all availability results, creates status list and sends it to Registry
- Procedure 5a: updates availability in database
- Message 6: returns call

3.3.7 Start data acquisition

When an Enabler is started it should be initialized to start data acquisition in case Underlying resources are sensors.

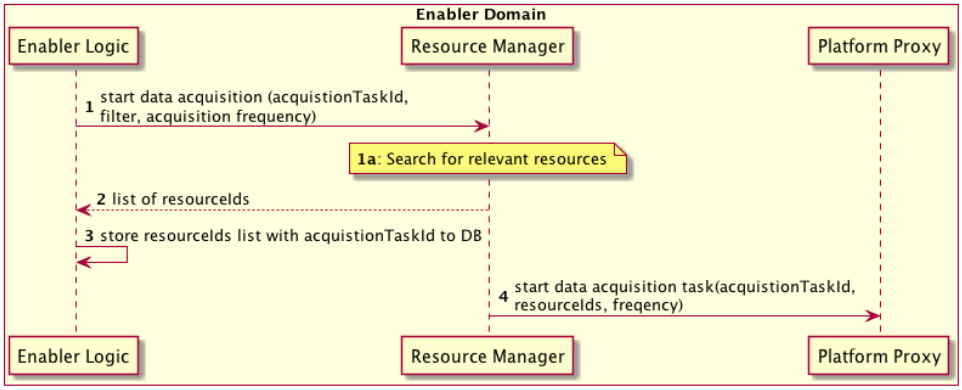

Figure 9. Start data acquisition

Description:

- Message 1: Enabler Logic sends a request to Resource Manager to start data acquisition. It sends generated acquisitionTaskId to Resource Manager.
- Procedure 1a: Resource Manager searches relevant resources and filters them (details are in next diagram).
- Message 2: Resource Manager returns chosen resourceIds to Enabler Logic.
- Message 3: Enabler Logic saves list of resources and acquisitionTaskId to internal database.
- Message 4: Resource Manager sends message for starting data acquisition.

3.3.8 Search

Enabler searches for resources needed for operations within Enabler logic.

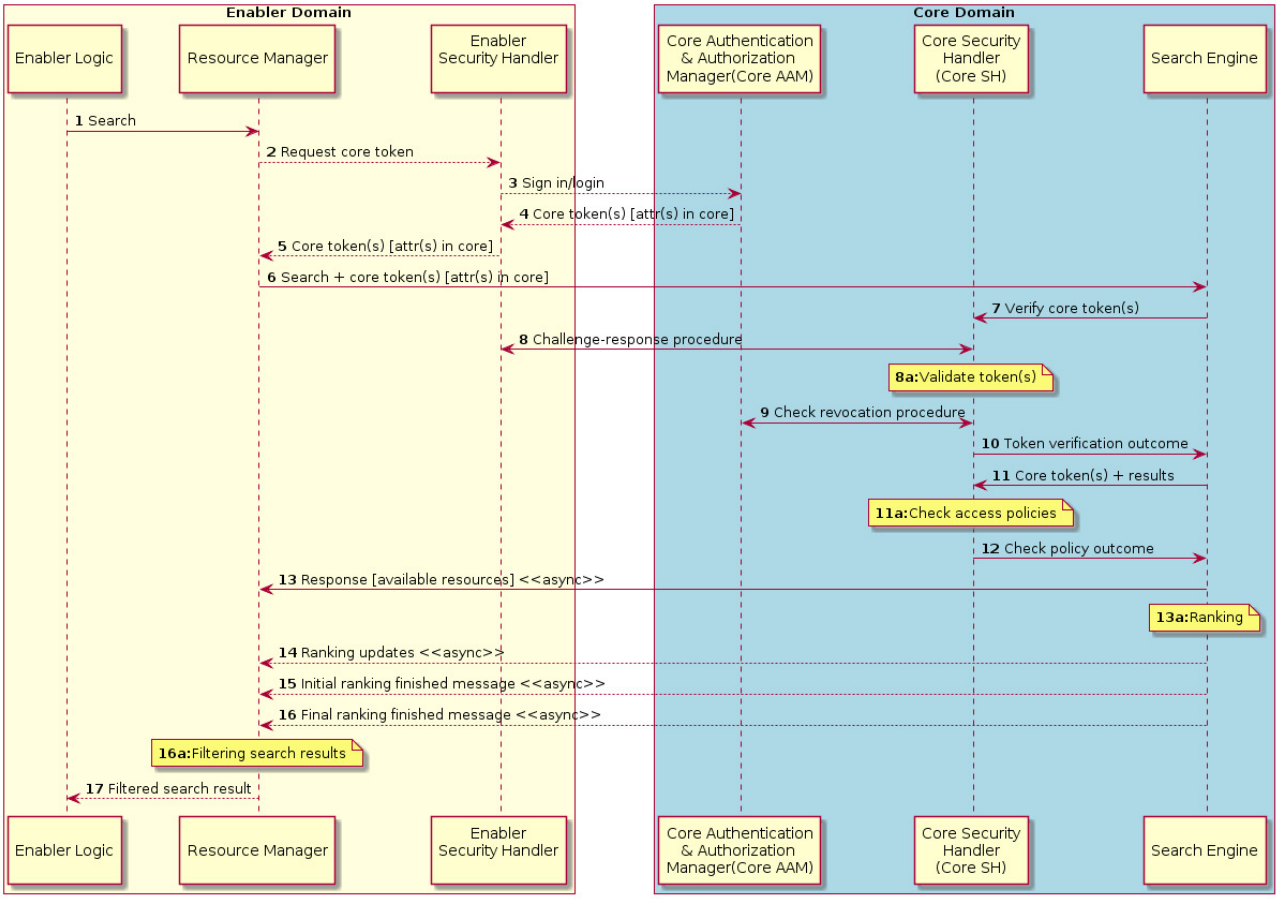

Figure 10. Search

Description:

- Message 1: Enabler Logic sends request to Resource Manager to find resources needed by Enabler Logic according to certain criteria.
- Message 2 (optional): generated by the Resource Manager and sent to the Enabler Security Handler. It is used to trigger the recovery of the core token(s). If the Enabler is already logged in, it is not necessary.
- Message 3 (optional): generated by the Enabler Security Handler and sent to the Core AAM in which the Enabler is registered. It is used to authenticate the Enabler. If the Enabler is already logged in, it is not necessary.
- Message 4 (optional): generated by the Core AAM and sent to the Enabler Security Handler. It is used to provide the home token(s) with attributes included. If the Enabler is already logged in, it is not necessary.
- Message 5 (optional): generated by the Enabler Security Handler and sent to the Resource Manager. It is used to deliver the core token(s).
- Message 6: generated by the Resource Manager and sent to the Search Engine. It sends a search query and the core token(s) to the Search Engine.
- Message 7: generated by the Search Engine and sent to the Core Security Handler. It is used to ask to the Security Handler to verify the complete validity of the token.

- Procedure 8: procedure that allows the Enabler Security Handler acting on behalf of the Resource Manager to demonstrate that it is the real owner of the token(s).
- Procedure 8a: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 9: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 10: generated by the Core Security Handler and sent to the Search Engine. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 11: generated by the Search Engine and sent to the Core Security Handler. It is used to deliver the core token(s) previously verified and the results of the search operation.
- Procedure 11a: procedure that checks, for each resource, if the attributes contained in the core token(s) satisfy the access policy associated to that resource.
- Message 12: generated by the Core Security Handler and sent to the Search Engine. It is used to deliver the result of the previous procedure.
- Message 13: generated by the Search Engine and sent to the Resource Manager asynchronously. It is used to deliver the result of the search operation (available resources).
- Procedure 13a: executes ranking of resources
- Message 14 (optional): asynchronously sends ranking update to Resource Manager
- Message 15 (optional): asynchronously sends a message about the availability of an initial ranked list of relevant results
- Message 16 (optional): synchronously sends a message about the availability of the final ranked list of relevant results
- Procedure 16a (optional): filtering of received search results by Resource Manager according to criteria defined in message 1
- Message 17: returns filtered search results to Enabler Logic

3.3.9 Stop data acquisition

Sometimes it is necessary that Enabler Logic stops an active process of data acquisition, which is initiated as defined in the Section 3.3.7.

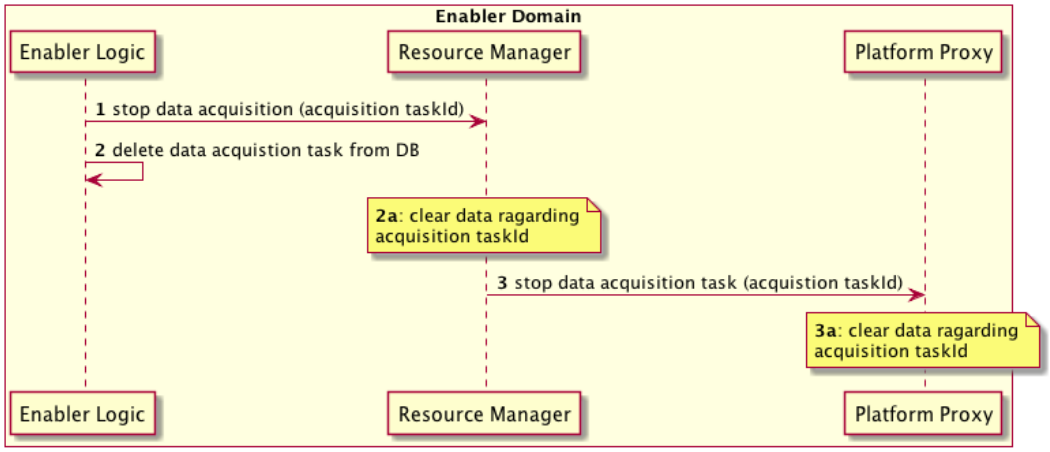

Figure 11. Stop data acquisition

Description:

- Message 1: Enabler Logic sends a request to Resource Manager to stop data acquisition.
- Message 2: Enabler Logic deletes an entry of the data acquisition in its internal DB
- Procedure 2a: Resource Manager clears its entry of the data acquisition task which is stopped.
- Message 3: Resource Manager sends a stopping request to Platform Proxy to stop.
- Procedure 3a: Platform Proxy clears all entries regarding the data acquisition task which is stopped.

3.3.10 Data acquisition

Platform Proxy acquires data from resources either, periodically or using subscriptions.

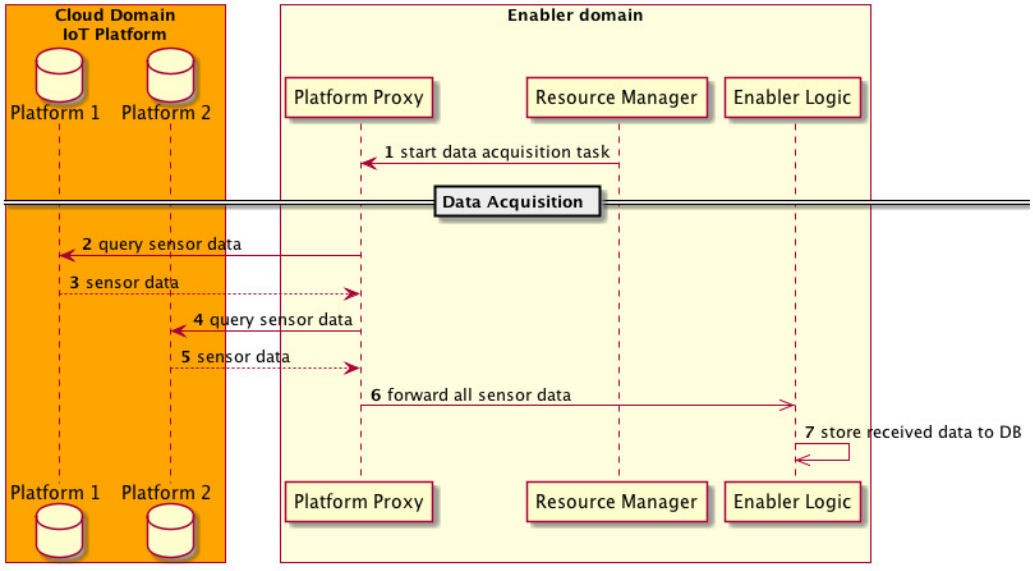

Figure 12. Data acquisition

Description:

- Message 1: When Resource Manager starts data acquisition then Platform Proxy schedules tasks for acquiring data
- Message 2: Platform Proxy sends request for resource access in platform 1
- Message 3: Platform Proxy receives data from platform 1
- Message 4: Platform Proxy sends request for resource in platform 2
- Message 5: Platform Proxy receives data from platform 2
- Message 6: After all data from platforms are received, Platform Proxy sends them to Enabler Logic in one message
- Message 7: Enabler Logic stores data to internal DB for further processing. Enabler Logic can start internal processing upon receiving this message or in scheduled tasks

3.3.11 Unresponding resource during data acquisition

This diagram shows what happens when Platform Proxy gets an error message or platform is not responding to a request for data from its resource.

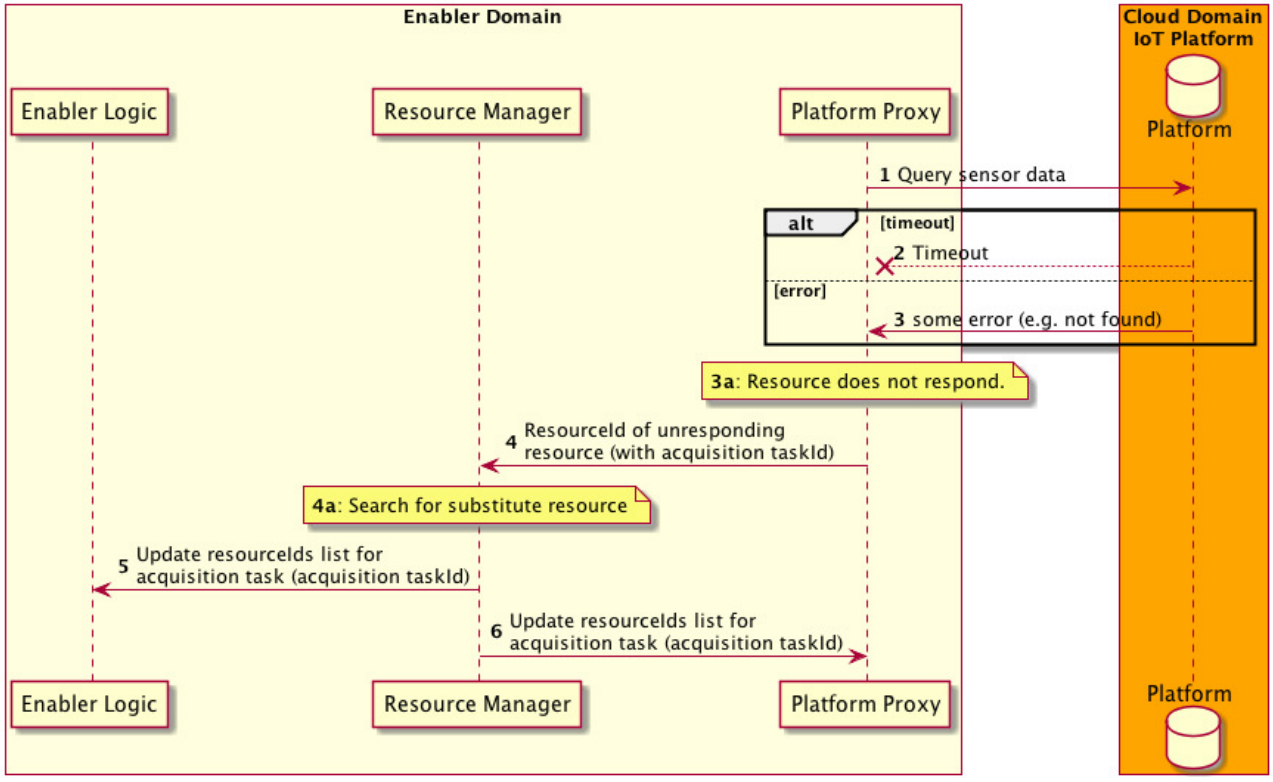

Figure 13. Unresponding resource

Description:

- Message 1: When Platform Proxy schedules a data acquisition task, it sends this message to platform.

- Message 2: To prevent deadlock when a response message from the IoT platform is not received by Platform Proxy, a timeout is used. This message triggers timeout.
- Message 3: When there is an error in resource access, this message is received by Platform Proxy.
- Procedure 3a: Platform Proxy identifies that the resource is not responding (either because of timeout or error) and starts sending the following message.
- Message 4: Platform Proxy sends message to Resource Manager that a specific resource is not responding.
- Procedure 4a: Resource Manager starts the procedure to find another resource for the specified task. It could have cached potential resources and can chose from them or it can start search for resources in the Core (using the diagram Search)
- Message 5: After Resource Manager has found a substitute resource, it informs Enabler Logic about that.
- Message 6: Resource Manager informs Platform Proxy about updates of resources for specific acquisition taskId.

3.3.12 Replacement of a malfunctioning resource

Enabler Logic in some cases can detect that a resource is malfunctioning (e.g., it may send erroneous data) and in such case it can ask the Resource Manager to replace this resource with another one.

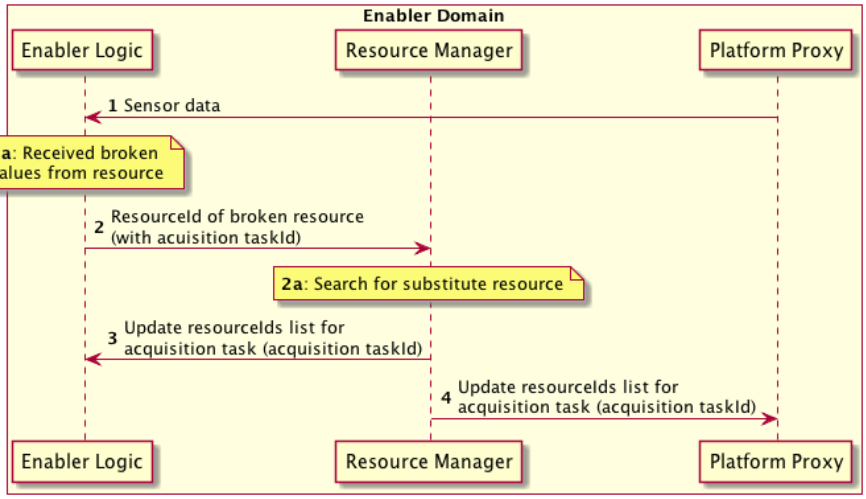

Figure 14. Replacement of a malfunctioning resource

Description:

- Message 1: Enabler Logic receives resource readings from Platform Proxy.
- Procedure 1a: Enabler Logic detects that the data is erroneous.
- Message 2: Enabler Logic asks Resource Manager to find a substitute resource.
- Procedure 2a: Resource Manager searches for another resource. It could have cached potential resources and can choose either from an existing list of potential

resources or can start the search procedure in the Core Domain (using the previous sequence diagram Search).

- Message 3: Resource Manager sends information about a substitute resource to Enabler Logic.
- Message 4: Resource Manager informs Platform Proxy about updates of resources for a specific acquisition taskId.

3.3.13 Access to Enabler resource using Enabler RAP

This is a typical Enabler usage scenario when a symbIoTe application uses resources offered by the Enabler. Access is enabled the Enabler RAP. The Core Resource Access Monitor (Core RAM) in symbIoTe Core needs to be informed of this interaction.

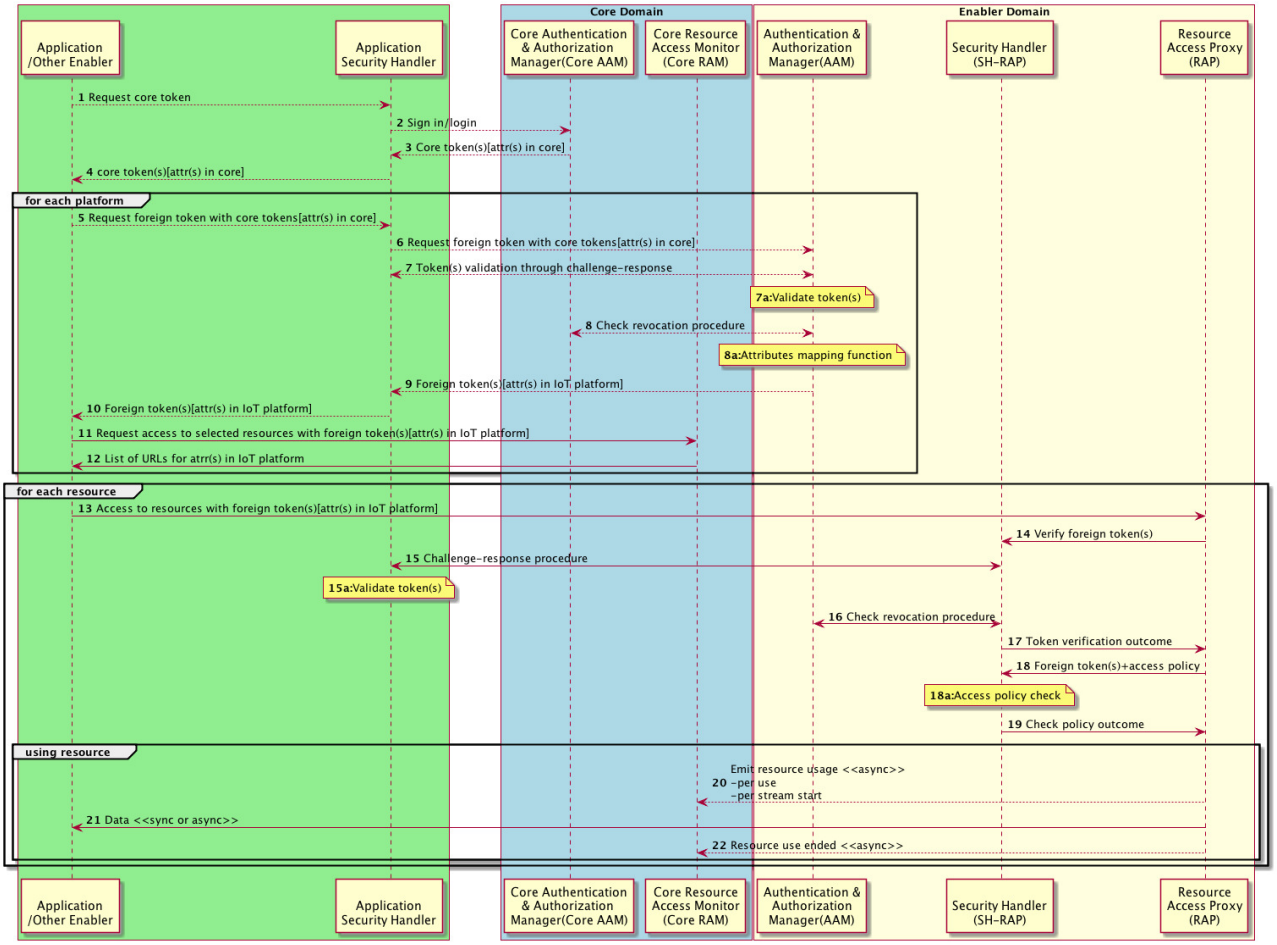

Figure 15. Access to Enabler resource using its RAP

Description:

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

- Message 2 (optional): generated by the Security Handler and sent to the home AAM in which the Application/Other Enabler is registered. It is used to authenticate the Application/Other Enabler. If the Application/Other Enabler is already logged in, it is not necessary.
- Message 3 (optional): generated by the home AAM in the IoT platform and sent to the Security Handler. It is used to provide the home token(s) with attributes included. If the Application/Other Enabler is already logged in, it is not necessary.
- Message 4 (optional): generated by the Security Handler and sent to the Application/Other Enabler. It is used to deliver the core token(s).
- Message 5 (optional): generated by the Application/Other Enabler and sent to Application Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform. If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Message 6 (optional): generated by the Application Security Handler and sent to the foreign AAM in IoT platform. It is used to trigger the operations for obtaining the foreign token(s). If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Procedure 7 (optional) (AppSecurityInterface): procedure that allows the Security Handler that is acting on behalf of the Application/Other Enabler to demonstrate that it is the real owner of the token(s). If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Procedure 7a (optional): verification of the time validity, authenticity and integrity of the provided token(s). If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Procedure 8 (optional): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Procedure 8a (optional): procedure that, in case it is needed, translates attributes that the Application/Other Enabler has in the home IoT platform in a new set of attributes that it has in the Core Domain. If attributes are the same or the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Message 9 (optional): generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Message 10 (optional): generated by the Application Security Handler and sent to the Application/Other Enabler. It is used to forward the foreign token generated at the previous step.
- Message 11: Application/Other Enabler sends request access to selected resources to Core Resource Access Monitor. Message includes foreign token obtained in previous message

- Message 12: Core Resource Access Monitor returns list of URLs for selected resources in IoT platform
- Message 13: generated by the Application/Other Enabler and sent to the Resource Access Proxy in the foreign IoT platform. It is used to access resources, while providing the foreign token previously obtained.
- Message 14: generated by the Resource Access Proxy and sent to the Security Handler in the foreign IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 15: procedure that allows the Application Security Handler that is acting on behalf of the Application/Other Enabler to demonstrate that it is the real owner of the token(s).
- Procedure 15a: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 16: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself).
- Message 17: generated by the Security Handler in the foreign IoT platform and sent to the Resource Access Proxy. It is used to communicate the outcome of the token validation procedures performed by the Foreign Security Handler.
- Message 18: generated by the Resource Access Proxy and sent to the Security Handler. It is used to deliver the core token(s) previously verified and the access policy of the requested resource to the Security Handler.
- Procedure 18a: it is used to check if the attributes included in the core token(s) satisfy the access policy associated to the requested resource.
- Message 19: generated by the Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- Procedure: In this procedure, the Enabler internally calculates results or fetch data from IoT platforms or services and generates messages 20-22. This procedure is defined in Using Resource diagram.
- Message 20: asynchronously emit resource usage per use/per stream start
- Message 21: this message can be synchronous, then Resource Access Proxy returns data. If it is asynchronously then it can emit asynchronous messages for some time
- Message 22: if previous message is asynchronous then this message informs Core Resource Access Monitor when the stream is ended

3.3.14 Access to Enabler resource using its Domain-specific Interface

This is another typical usage scenario when a symbIoTe application uses resources offered by the Enabler. In this case the application uses Enabler's Domain-specific Interface. The application needs to be registered only with the Enabler.

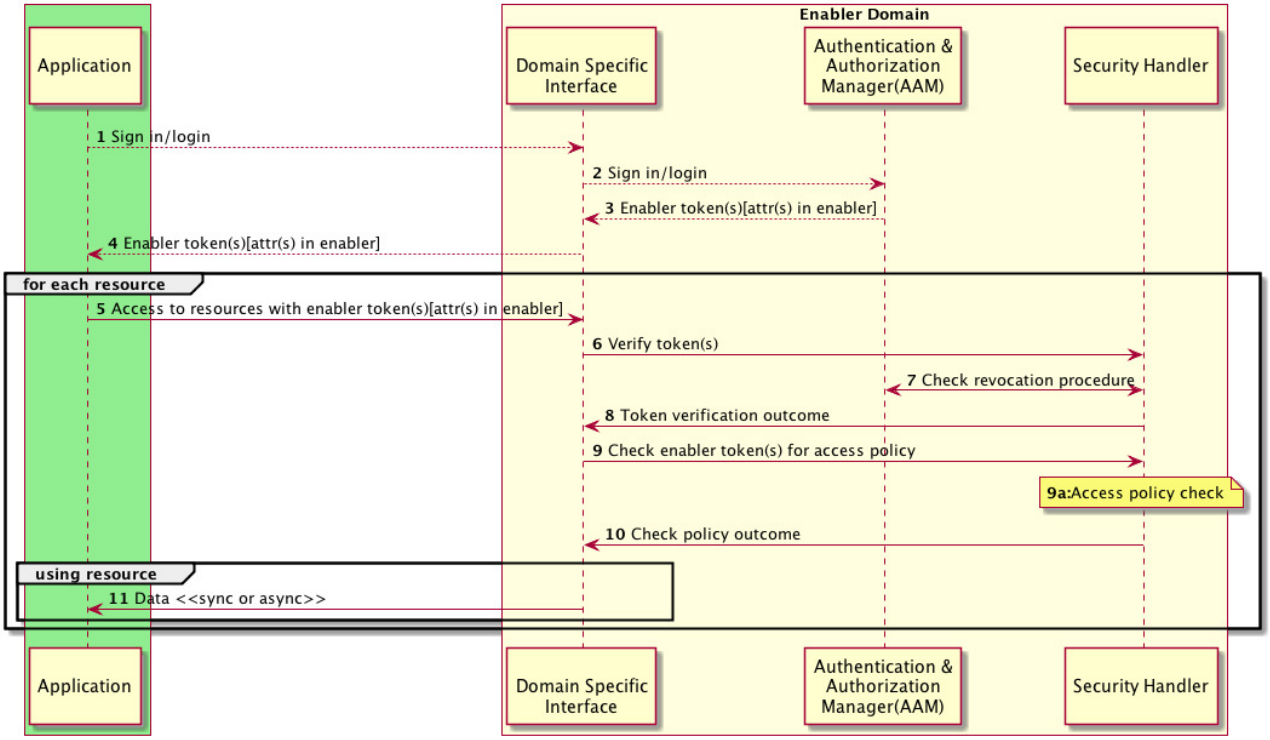

Figure 16. Access to Enabler resource using Domain-specific Interface

Description:

- Message 1 (optional): generated by the application and sent to the Domain-specific Interface. It is used to get Enabler token for accessing its services. If the application is already logged in, it is not necessary.
- Message 2 (optional): generated by the Domain-specific Interface and sent to the Enabler's AAM in which the application is registered. It is used to authenticate the application. If the application is already logged in, it is not necessary.
- Message 3 (optional): generated by the Enabler's AAM and sent to the Domain-specific Interface. It is used to provide the token(s) with attributes included. If the application is already logged in, it is not necessary.
- Message 4 (optional): generated by the Domain-specific Interface and sent to the application. It is used to deliver the token(s).
- Message 5: generated by the application and sent to the Domain-specific Interface. It is used to access resources, while providing the token previously obtained.
- Message 6: generated by the Domain-specific Interface and sent to the Security Handler in the Enabler. It is used to ask the Security Handler to verify the complete validity of the token.
- Procedure 7: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Enabler's AAM before the expiration time indicated within the token itself).
- Message 8: generated by the Security Handler in the Enabler and sent to the Domain-specific Interface. It is used to communicate the outcome of the token validation procedures performed by the Enabler Security Handler.

- Message 9: generated by the Domain-specific Interface and sent to the Security Handler. It is used to deliver the token(s) previously verified and the access policy of the requested resource to the Security Handler.
- Procedure 9a: it is used to check if the attributes included in the token(s) satisfy the access policy associated to the requested resource.
- Message 10: generated by the Security Handler and sent to the Domain-specific Interface. It is used to deliver the result of the operation executed during the previous step.
- Procedure: In this procedure, the Enabler internally calculates results or fetches data from IoT platforms or services and generates message 11. This procedure is defined in the diagram Using Resource from enabler application.
- Message 11: If this message is synchronous, then Domain-specific Interface returns processed sensor data. If it is asynchronously then it emits asynchronous message with processed sensor data.

3.3.15 Reporting Enabler resource usage

Enabler is reporting on usage of resources it offers to applications.

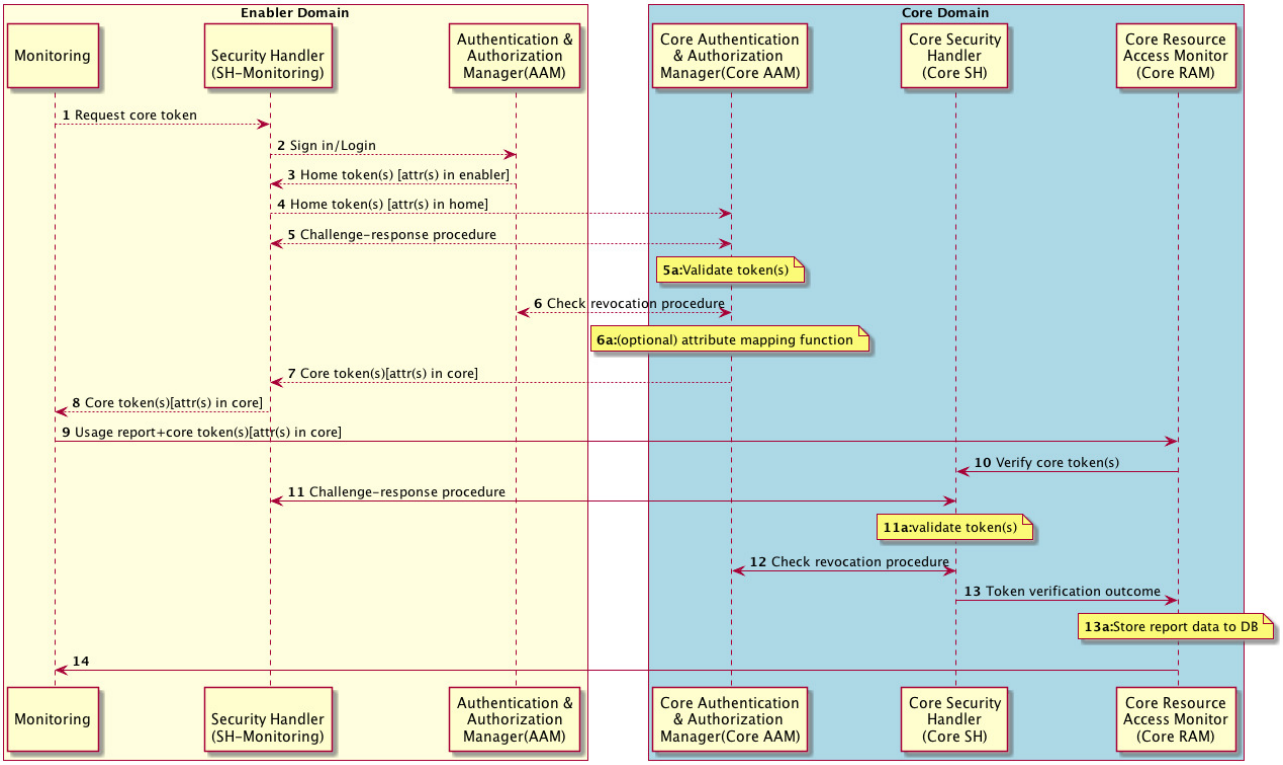

Figure 17. Enabler resource usage reporting

Description:

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

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

- Message 13: Generated by the Security Handler in the Core Domain and sent to the Core Resource Access Monitoring. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 13a: Stores report data to database
- Message 14: returns call

4 Components basic information tables

This chapter presents the basic information about symbIoTe Enablers' components implemented for the v1.0.0 and presents it in tabular style for each component in alphabetical order.

4.1 Authentication and Authorization Manager

Component/service name	Authentication and Authorization Manager
URL of source codes	https://github.com/symbiote-h2020/AuthenticationAuthorizationManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/AuthenticationAuthorizationManager/doxygen/
List of release v1.0.0 features included	Registration of a new application (actor/user) in the core AAM through Administration UI and in the platform AAM by means of the platform owner Obtaining foreign token using home token Retrieval of the AAM certificates Tokens & certificates validation against: - signatures and expirations, - cross-aam comms for foreign tokens validation, - (WIP) forged trust chain (offline validation), - (WIP) revoked credentials. (WIP) Fine grained Issuing actors clients' certificates (using CSRs, user, password) (WIP) Login with signed username and clientId tupple (issuing home tokens) Login with home tokens (issuing foreign tokens) (WIP) Login for guest actors (issuing guest tokens) • (WIP) Revoking credentials

4.2 Enabler Logic

Component/service name	Enabler Logic
URL of source codes	https://github.com/symbiote-h2020/EnablerLogic
URL of Javadoc documentation	https://symbiote-h2020.github.io/EnablerLogic/doxygen/
List of release v1.0.0. features included	• RAP plugin (resource read, actuation, service calling) • Resource registration, update, unregistration • Spring boot starter implementation • Handling broken or unresponsive resources • Stopping data acquisition

4.3 Monitoring

Component/service name	Monitoring
URL of source codes	https://github.com/symbiote-h2020/Monitoring
URL of Javadoc documentation	https://symbiote-h2020.github.io/Monitoring/doxygen/
List of release v1.0.0. features	• Icinga2 server has been installed in a public machine in order to enable test by partners without the need of

included	installing Icinga2.

4.4 Platform Proxy

Component/service name	Platform Proxy
URL of source codes	https://github.com/symbiote-h2020/EnablerPlatformProxy
URL of Javadoc documentation	https://symbiote-h2020.github.io/EnablerPlatformProxy/doxygen/
List of release v1.0.0. features included	• Allows scheduling and managing data acquisition tasks • Periodically contacts the resources for each acquisition task to download observation data • Informs Enabler Logic about new observation data • Allows getting observation data on request

4.5 Registration Handler

Component/service name	Registration Handler
URL of source codes	https://github.com/symbiote-h2020/RegistrationHandler
URL of Javadoc documentation	https://symbiote-h2020.github.io/RegistrationHandler/doxygen/
List of release v1.0.0. features included	• Rest interfaces has been implemented • Storage in *mongo db* is being done • messaging (AMQP) has been implemented. They you communicate using the RPC RabbitMQ communication pattern with Interworking Interface and Routing-direct RabbitMQ communication pattern with RAP component.

4.6 Resource Access Proxy

Component/service name	Resource Access Proxy
URL of source codes	https://github.com/symbiote-h2020/ResourceAccessProxy
URL of Javadoc documentation	https://symbiote-h2020.github.io/ResourceAccessProxy/doxygen/
List of release v1.0.0. features included	• asynchronous messaging (AMQP) interface implemented (to receive registration messages from Resource Handler) • Internal storage of Platform and Resource IDs mapping • translating Platform and Resource IDs to forward requests to IoT platforms • REST support to access to resources • OData support to access to resources • Filtering support for resources historical data • Internal asynchronous messaging (AMQP) interface implemented (to decouple generic rap features from platforms' proprietary access to resources) • WebSocket implementation for push mechanism • Custom information model support (BIM, PIM) with OData

4.7 Resource Manager

Component/service name	Resource Manager
URL of source codes	https://github.com/symbiote-h2020/EnablerResourceManager
URL of Javadoc documentation	https://symbiote-h2020.github.io/EnablerResourceManager/doxygen/
List of release v1.0.0. features included	• Update Task • Cancel Task • Support for caching resource ids • Support for not informing Platform Proxy about tasks • Enhance search to add more parameters, request rank results and support SPARQL queries • Listen for unavailable resources from Platform Proxy • Listen for resources with wrong data from Enabler Logic • Integrate new Security Handler

5 Design of symbIoTe Use Case Enablers

This section focuses on generic Enabler components (Resource Manager, Platform Proxy, and Enabler Logic) which are domain-specific and relevant to symbIoTe use cases. It is envisioned that symbIoTe use cases will use Enablers (if and when appropriate) to facilitate application development. Enablers perform all processes to identify, access and analyze sensors data originated from the underlying IoT resources.

5.1 Smart Mobility & Ecological Urban Routing Use Case

The Smart Mobility and Ecological Urban Routing (SMEUR) use-case addresses the problem of inefficient transportation and poor air quality that many European cities face nowadays. This use case offers the ecologically most preferable routes for motorists, bicyclists and pedestrians based on the available traffic and environmental data acquired through various platforms. This scenario is extremely relevant for people who travel within the major European cities, since a constant exposure to pollutants can cause severe health problems. It is also of the interest of the municipalities' governing bodies that, by helping their citizens to avoid these health problems, they can reduce health care costs. Additionally, the use case will provide a way for users to search for Points of Interests, filtered by certain factors such as air quality, noise pollution and parking availability. SymbIoTe empowers this use case by providing platform interoperability, allowing for developers to easily access and handle data from different platforms and domains in the same manner.

The use case showcases platform interoperability within the Application and Cloud Domain with a potential for business models for bartering and trading of resources, which also require IoT platform federations. Through symbIoTe, it is possible to obtain and use air quality data from different platforms without having to worry about their format. Additionally, it facilitates the development of reusable applications for urban services.

Enablers facilitate the implementation of SMEUR applications by handling air quality data from underlying IoT platforms, and by integrating this data with external Routing Services to find ecological routes. Furthermore, integrated air quality data exposed by the Enabler can also serve as input to other applications, not just the one envisaged within this use case. An example application could be the monitoring of air quality in the city to alarm citizens in the event of pollution with potential peril for human health.

5.1.1 SMEUR workflows and Enabler Architecture

Workflows of the SMEUR use case are mapped to the Enabler Architecture, and Enabler components are identified which are responsible for executing processes within the use case. Workflows defined within the use case are the following:

1. Data Acquisition
2. Data Interpolation
3. Calculation of Green Route
4. Point of Interest Search.

Each of the workflows is mapped to Enabler-specific components shown in Figure 18. As mentioned in Section 3, Enabler-specific components (Enabler Logic, Platform Proxy and Resource Manager) need to be created and implemented according to domain-specific

requirements. In SMEUR domain, Enabler-specific components are defined based on the aforementioned workflows.

Data Acquisition workflow is implemented within Resource Manager and Platform Proxy. Data Interpolation, Calculation of Green Route and Point of Interest Search are functionalities implemented within Enabler Logic, as shown in Figure 18. IoT platforms used within the use case are sources of air-quality data. These platforms are OpenIoT, UWEDAT, and MoBaaS. In the use case there also exist external services for adding value to air-quality data from the aforementioned IoT platforms and exposed through symbIoTe. These external services are the Routing Service (RS), and Point of Interest (PoI) Search Service. Routes and PoIs are found by combining the best routes and PoIs with air-quality data. Those external services can also be situated within IoT platforms (as RS within MoBaas). Even such service within a specific IoT platform is "enriched" by using symbIoTe because it can use resources from other symbIoTe Compliant Platforms, not only from its own platform.

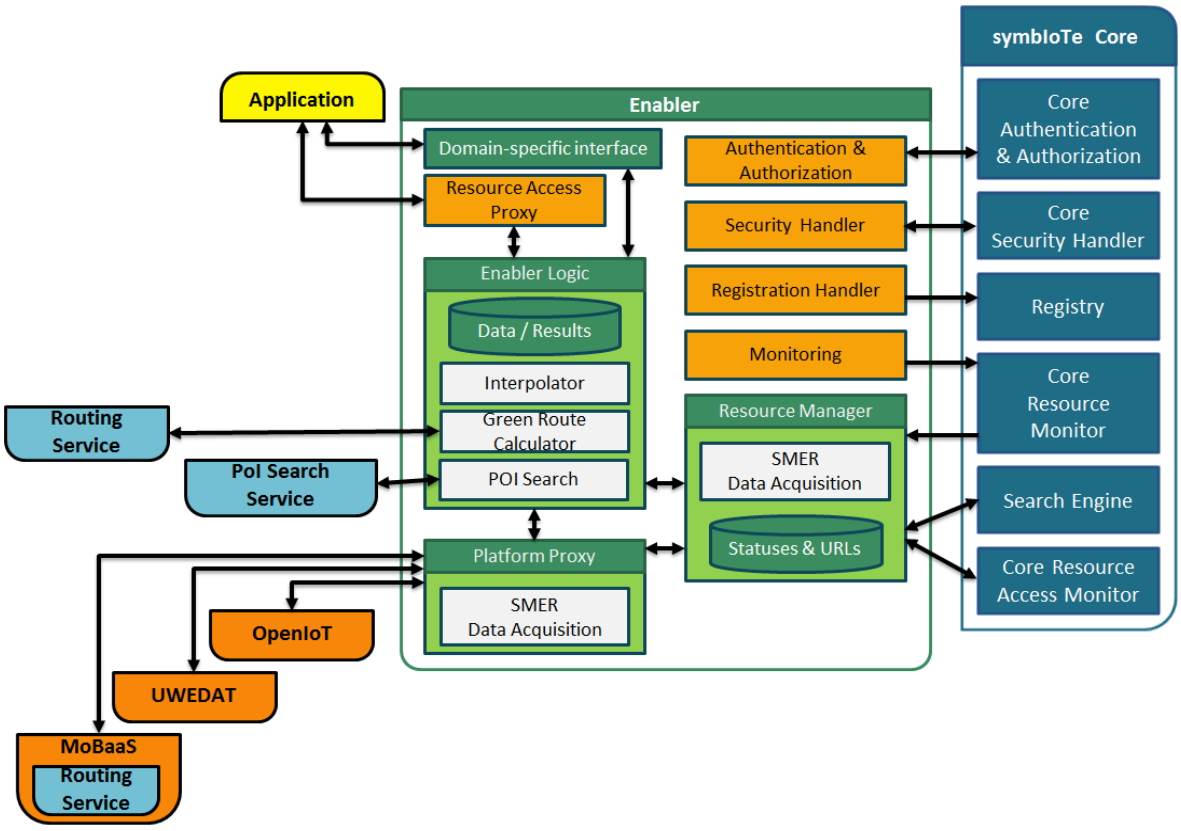

Figure 18. Architecture of Smart Mobility & Ecological Urban Routing Enabler

In the following parts of this Section workflows of the SMEUR use case are described in detail, and sequence diagrams showing how these functionalities are implemented within Enablers are introduced.

5.1.2 Data Acquisition

The workflow Data acquisition is responsible for finding resources and acquiring sensor data from underlying IoT platforms: wearable sensors from the OpenIoT platform hosted by UNIZG-FER, fixed stations from AIT's UWEDAT System and sensors from Ubiwhere's

MoBaaS platform. Resource Manager is responsible for finding the resources and Platform Proxy for acquiring data from resources and forwarding it to Enabler Logic, where the data is stored. All three platforms provide different sensor readings through the harmonized interface so they can be used in a uniform way for the different locations. Acquired data is handled by components within Enabler Logic, and offered to applications.

5.1.3 Data Interpolation

The workflow Data Interpolation finds relevant resources by using Resource Manager and takes sensor readings as input. It has an internal state, which consists mostly of interpolated values together with some auxiliary values (e.g., date of acquisition, version of the underlying street grid, etc.). The component is planned to be implemented in the Enabler-specific component – Enabler Logic.

It produces air quality indexes for street segments. The set of street segments is aligned with those used by the Routing Service (Figure 19).

5.1.3.1 Interpolator initialization and workflows

The Interpolator and the RS use a network of street segments. Usually the RS already works with such a network thus it makes sense to let the RS provide the network for the Interpolator. The Interpolator component obtains its data from the underlying IoT platforms. The selection of the resources to be used within Interpolator is done through Resource Manager which contacts the symbIoTe Search Engine in Core Services (Figure 19). Platform Proxy is responsible for acquiring the necessary data, so that interpolation is made possible (Figure 20).

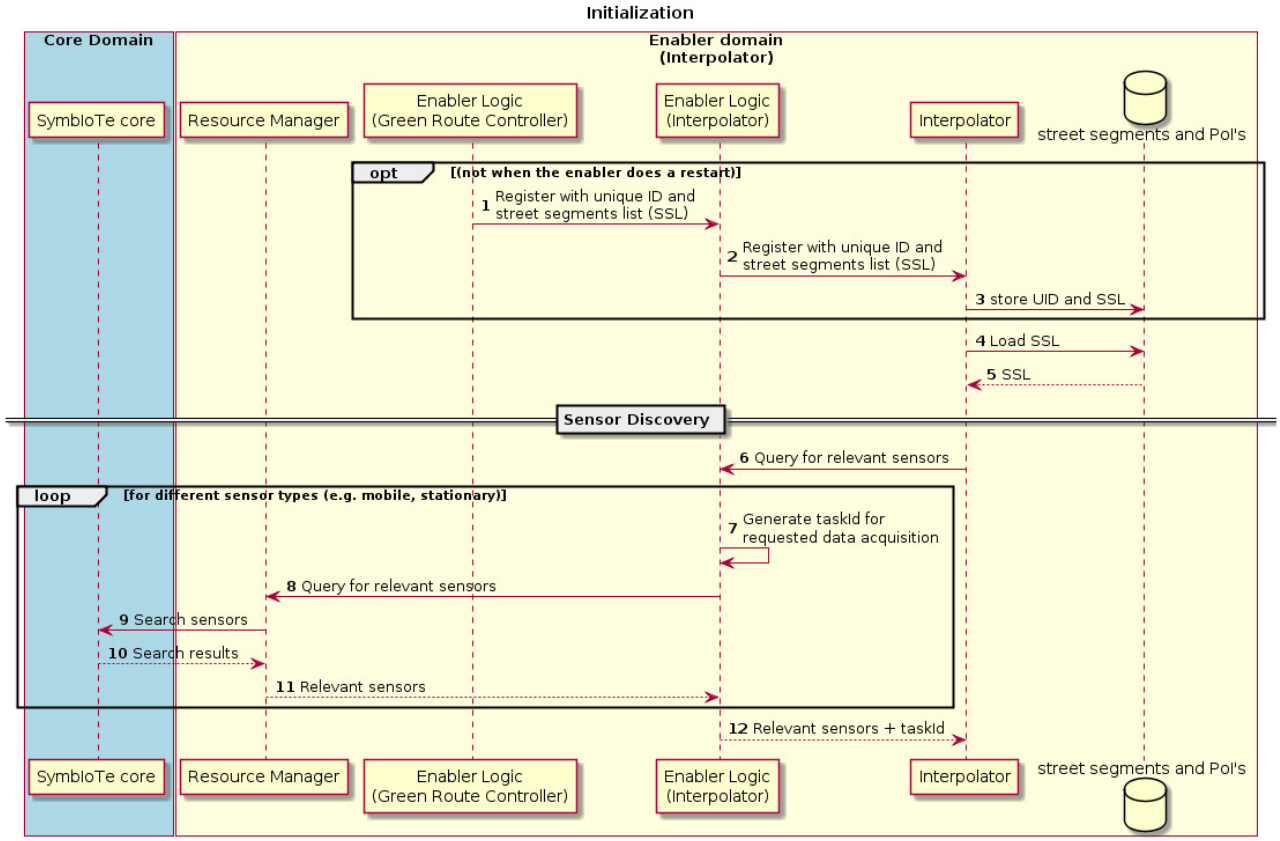

Figure 19. Initialization of Interpolator

Description:

- Message 1: Enabler Logic's Green Route Controller sends a registration message with street segment list of Routing Service.
- Message 2: Unique Id and Street Segment List is forwarded to Interpolator.
- Procedure 3: Interpolator saves the SSL with UID received from EL Interpolator.
- Procedure 4: Interpolator fetches the SSL to start the interpolation on each segment.
- Message 5: SSL list is delivered to Interpolator.
- Message 6: Interpolator sends a request to fetch relevant sensors on each segment to EL Interpolator.
- Procedure 7: Acquisition task Id is generated for the requested data acquisition.
- Message 8: EL Interpolator sends a request to Resource Manager to find relevant sensors in SymbIoTe Core.
- Message 9: RM searches the SymbIoTe Core to find requested relevant sensors.
- Message 10: List of relevant sensors in a given area is returned to RM.
- Message 11: RM returns a search result list of sensors to EL interpolator.
- Message 12: EL Interpolator returns a list of relevant sensors with generated taskId to Interpolator.

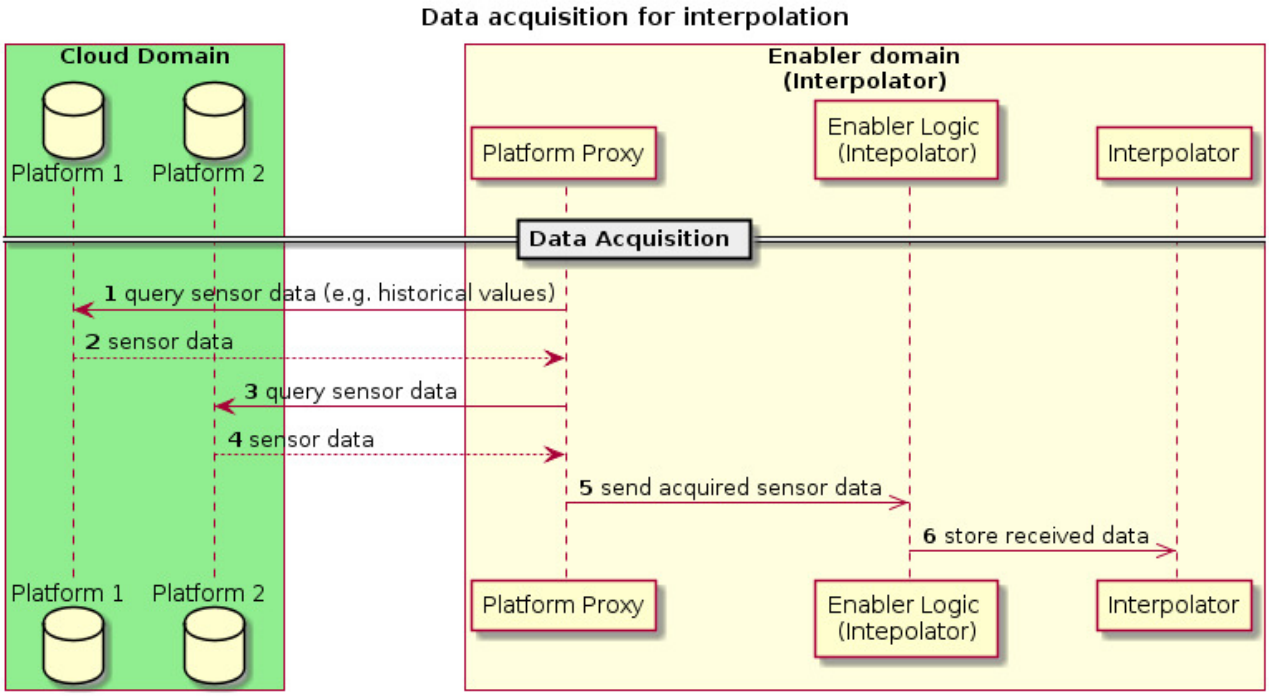

Figure 20. Data Acquisition for Interpolation

Description:

- Message 1,3: Platform Proxy sends a message to query a specified sensor registered under Platform in Cloud Domain.
- Message 2,4: Sensor data is returned to PP.
- Message 5: Acquired sensor data is sent to Enabler Logic's Interpolator.
- Message 6: Received sensor data is forwarded to Interpolator where it is stored in database.

There are two alternatives for triggering the Interpolator to do a new interpolation:

1. Time trigger. The Interpolator will be started on a regular base using a timer. This method can be used to simplify the effort of implementation (Figure 21), which is why the first version will be implemented following this scenario.
2. Event triggered. The Interpolator registers itself with the platforms and will be informed via an event when new relevant data is available. This method ensures the use of the most up-to-date data.

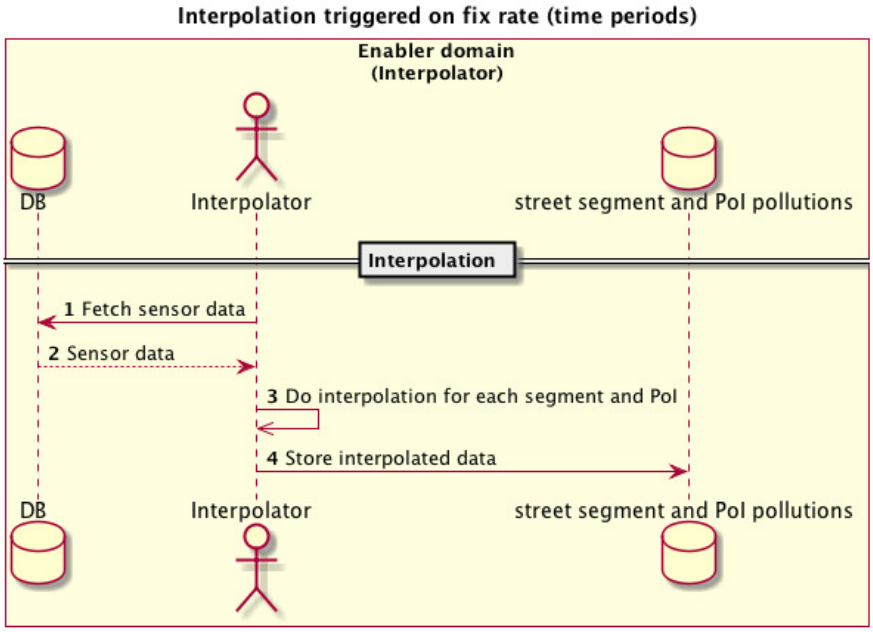

Figure 21. Data Interpolation

Description:

- Message 1: Interpolator fetches the sensor data for each street segment from database.
- Message 2: Sensor data is returned to Interpolator.
- Message 3: Interpolation on available sensor data is done for each street segment.
- Message 4: Interpolated value for each street segment is stored to database.

5.1.3.2 Interface for providing Interpolated Data

Once an interpolation is available, the Green Route Controller obtains it and sends the results to the RS which requires another interface/method. Interpolated data for street segments can be treated as if there is an (artificial) sensor for each street segment. To keep the different interfaces homogenous with other components of symbIoTe it is desirable to shape this interface the same way as other IoT platforms. Thus, the interface will use the same design and behave as a RAP. This allows the Green Route Controller to obtain data in bulk. It also allows the Green Route Controller to register for various artificial "sensors" events and get (only) relevant updates in near real-time. This data can then be sent to the corresponding Routing Services.

5.1.3.3 Implementation architecture and environment

A convenient set of libraries for the interpolation task is available for the Python (CPython) environment. Since Java does not provide a similar convenient functionality, the implementation is done in Python.

5.1.4 Calculation of Green Route

The workflow Calculation of Green Route is implemented in Enabler-specific component Green Route Controller. The workflow for the calculation of Green Routes allows users to obtain routes that avoid areas with high pollution (and, possibly, other factors, such as traffic). These services (Routing Service within MoBaaS platform or external Routing Service, as shown in Figure 18) use the Enabler to obtain air quality data associated with street segments (as described in the previous Data Interpolation section). This data is used by the services in the calculations of the routes, penalizing routes that go through highly polluted areas. As such, an additional component Green Route Controller is developed to serve as a bridge between the Interpolator, the RS (whether as external services or services within IoT platform) and the applications requesting ecological routes.

Note that the difference between external Routing Services and platform Routing Services is in the following: The platform service must communicate with the Enabler through the Platform Proxy while the external service communicates with the Green Route Controller directly.

5.1.4.1 Obtaining Data

The RS consumes the street air quality data provided by the Interpolator so it could provide routes through areas with low pollution. The main concern with this process is that, whenever a route request is made, it is neither feasible nor efficient that the entire city air quality data is obtained from IoT platforms.

As such, it is envisioned that, after the first bulk of air quality data is obtained during the initialization of the Green Route Controller (Figure 22), only updates (and not the whole data set) are then obtained in the future. This way, data exchange between the various services is reduced and, by having the data stored and immediately available when a route request is made, the whole process is done more quickly. Additionally, RS may operate only in a restricted area and might not want to receive data from areas it does not operate in. It is also expected that, on registering with the Enabler, RS can specify which data it wants to receive (street id, restricted area, city, etc).

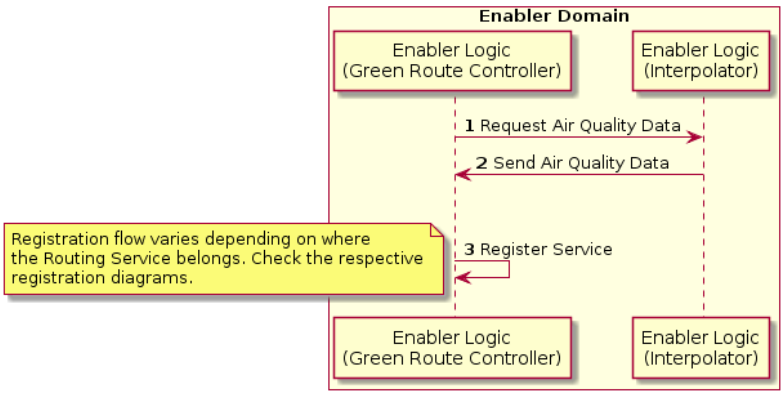

Figure 22. Initialization of the Green Route Controller

Description:

- Message 1: Enabler Logic's Green Route Controller service sends a request to fetch air-quality data of street segments in specified area.

- Message 2: Enabler Logic's Interpolator returns requested interpolated air-quality data.
- Procedure 3: Enabler Logic's GRC service proceeds with registration (described in following diagrams).

There are two distinct flows designed to implement the functionalities mentioned above. The first one is presented in Figure 23, where the Green Route Controller registers a RS. In return, the RS send its preferences describing the data it wants to receive. Alternatively, this can simply be a data request (e.g. the service lost all of its data and needs the Enabler to send it again). The Green Route Controller will store the platform's preferences and, return the street segment list for the preferred area. Afterwards, available data for specified street segments is sent to the platform where it is then stored and used for the calculations of routes.

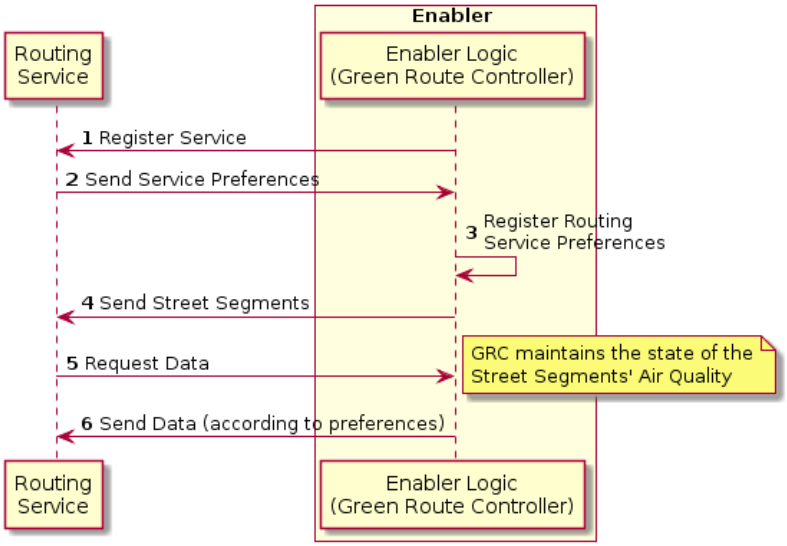

Figure 23. Registration and Data Request – external RS

Description:

- Message 1: Enabler Logic's GRC service sends registration request directly to external Routing Service.
- Message 2: Routing Service sends its preferences for data it wants to receive.
- Procedure 3: EL GRC registers RS with received preferences and saves it internally.
- Message 4: EL GRC sends street segments to external RS where requested data is available.
- Message 5: RS requests interpolated data for specified street segments.
- Message 6: EL GRC sends a response containing the requested data.

Figure 24 shows the same feature as in Figure 23 but this time the RS is part of the platform and Green Routing Controller needs to communicate with it by using the Platform Proxy and platform's RAP.

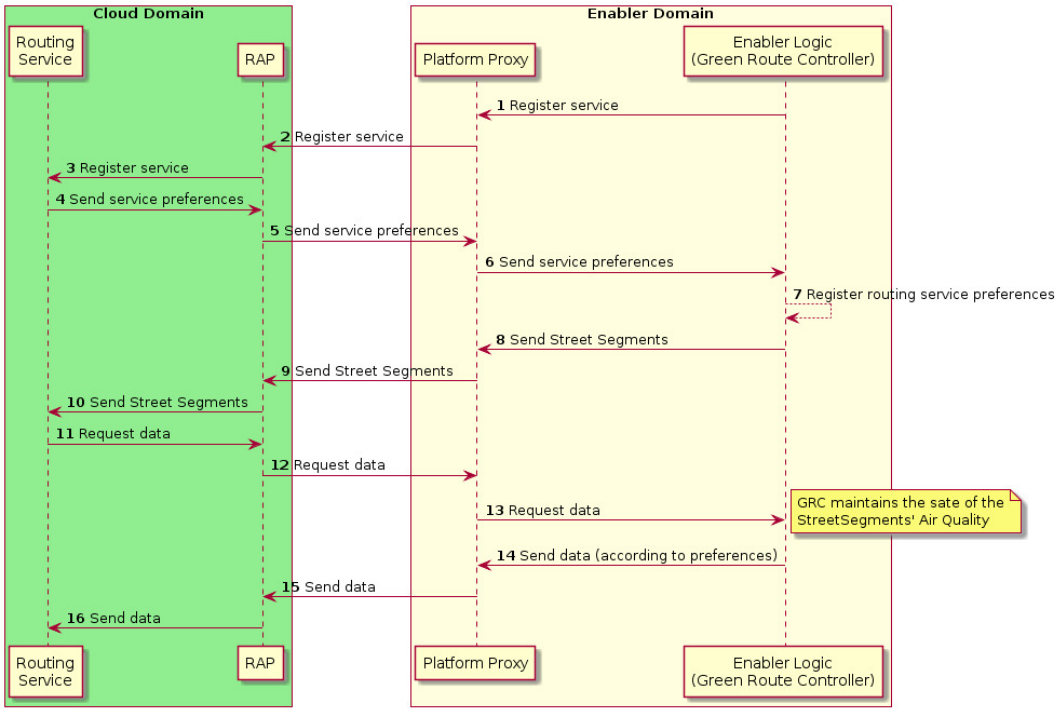

Figure 24. Registration and Data Request – RAP

Description:

- Message 1: Enabler Logic's GRC service sends registration request to Platform Proxy.
- Message 2: PP forwards the received registration request to Resource Access Proxy.
- Message 3: RAP forwards the registration request to Routing Service in Cloud Domain.
- Message 4: Routing Service sends its preferences for data it wants to receive to RAP.
- Message 5: RAP forwards the received preferences to PP.
- Message 6: PP forwards the received preferences to EL GRC.
- Procedure 7: EL GRC registers RS with received preferences and saves it internally.
- Message 8: EL GRC sends street segments where requested data is available, to PP.
- Message 9: PP forwards received street segments to RAP.
- Message 10. RAP forwards received street segments to RS in Cloud Domain.
- Message 11: RS in Cloud Domain sends a request for interpolated data to RAP.
- Message 12: RAP forwards the received request to PP.
- Message 13: PP forwards request to EL GRC.
- Message 14: EL GRC sends a response containing the requested data to PP.

- Message 15: PP forwards the response to RAP.
- Message 16: RAP delivers a response with the requested data to RS in Cloud Domain.

The second flow relates to obtaining data updates from the Interpolator. Only changes of the air quality state in the street segments are needed, since the rest of the data is already stored by the RS. This reduces the amount of data that needs to be sent between the Interpolator and the RS. Additionally, it reduces the time for the processing of a route request, since the RS containing the service only updates the information it needs for processing. As can be seen in Figure 25, the Interpolator sends its data updates to the Green Route Controller, which sends the relevant information to RS.

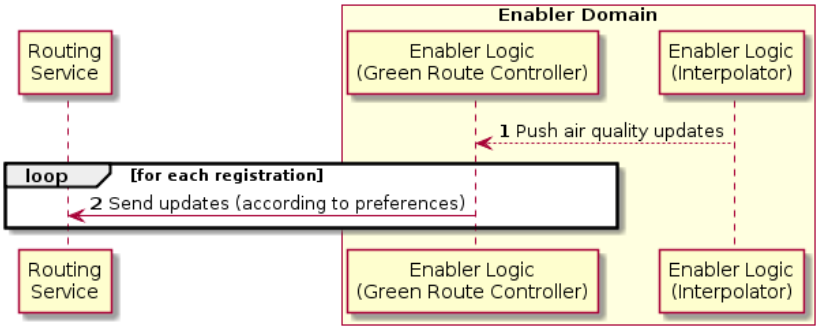

Figure 25. Air Quality Data Updates – external RS

Description:

- Message 1: Enabler Logic's Interpolator sends refreshed interpolated data for street segments where changes occurred.
- Message 2: For each registered external Routing Service updates with new values of interpolated data for street segments are sent accordingly.

Error! Reference source not found. shows the same flow as in Figure 25 but this time the RS is part of the platform and Green Route Controller needs to communicate using its Platform Proxy and platform's RAP

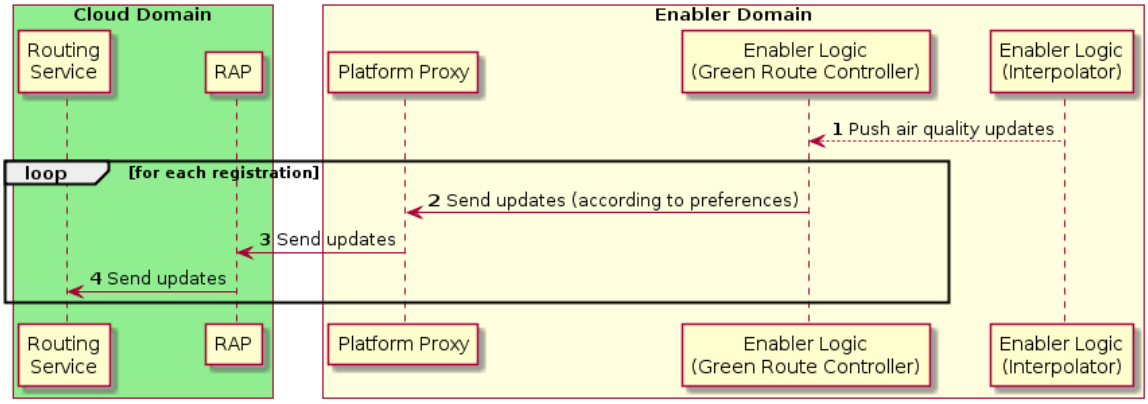

Figure 26. Air Quality Data Updates – RAP

Description:

- Message 1: Enabler Logic's Interpolator sends refreshed interpolated data for street segments where changes occurred.
- Message 2: For each registered Routing Service in Cloud Domain updates with new values of interpolated data for street segments are sent accordingly to Platform Proxy first.
- Message 3: PP forwards received updates to Resource Access Proxy.
- Message 4: RAP forwards received updates to RS in Cloud Domain.

Using these diagrams, the RS will have an updated set of data regarding the air quality for the relevant street segments.

5.1.4.2 Obtaining Route

As can be seen in Figure 27, the process of obtaining an ecological route is performed with a low overhead since an RS already stores up-to-date information needed for route calculation, as explained in the previous subsection. A registered application makes a request to the Green Route Controller within the Enabler, who directs it to the appropriate RS which can answer its request. The RS uses the collected air quality data, plus any other available data (e.g. traffic density data) to calculate the route. For this particular use case, the used RS will be provided by the AIT's routing service and Ubiwhere's MoBaaS platform.

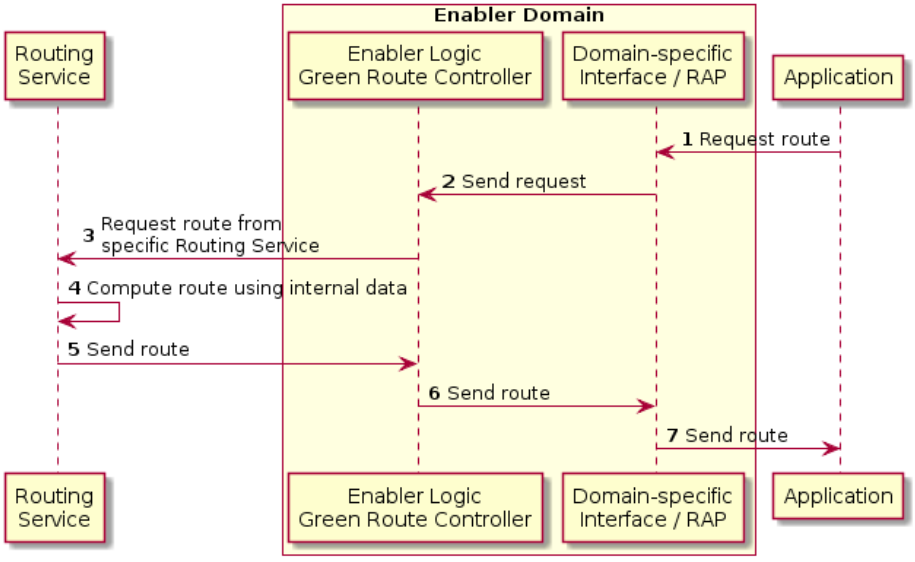

Figure 27. Obtaining Ecological Route – external RS

Description:

- Message 1: Application sends a request for green route to Domain-Specific Interface or Resource Access Proxy.
- Message 2: Request is forwarded towards Enabler Logic's Green Route Controller.
- Message 3: EL GRC contacts directly external Routing Service to calculate best route according to its preferences.

- Procedure 4: RS calculates the best route based on the available interpolated data on street segments between starting point and end point.
- Message 5: RS sends the calculated route to EL GRC.
- Message 6: EL GRC forwards received route to DSI/RAP.
- Message 7: Requested route is sent to application that requested it as a response.

Figure 28 is the same as Figure 27 but this time the RS is in the platform and Green Route Controller needs to communicate by using Platform Proxy and RAP.

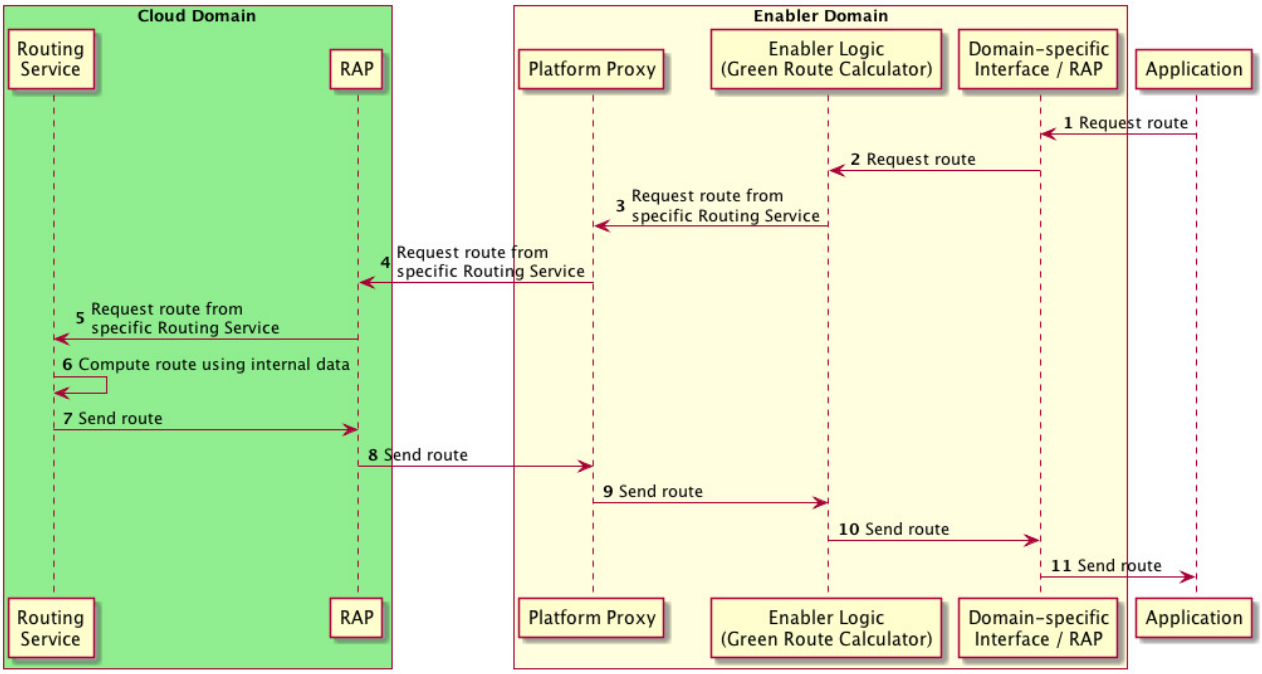

Figure 28. Obtaining Ecological Route – RAP

Description:

- Message 1: Application sends a request for green route to Domain-Specific Interface or Resource Access Proxy.
- Message 2: Request is forwarded towards Enabler Logic's Green Route Controller.
- Message 3: EL GRC forwards the received request to Platform Proxy.
- Message 4: PP forwards the request to Resource Access Proxy.
- Message 5: RAP forwards the message to Routing Service in Cloud Domain.
- Procedure 6: RS calculates the best route based on the available interpolated data on street segments between starting point and end point.
- Message 7: RS sends the calculated route to RAP.
- Message 8: Rap forwards the received route to PP.
- Message 9: PP forwards the received route to EL GRC.
- Message 10: Received route is returned to DSI/RAP.

- Message 11: Requested route is sent to application that requested it as a response.

5.1.5 Point of Interest Search

The workflow Point of Interest (PoI) Search is planned to be implemented in an Enabler-specific component named PoI Search. The workflow allows users to search for PoIs, such as restaurants or bars. Furthermore, for the PoI location air quality data is fetched (and any other available data) so the user can choose and filter received results according to his/her preferences. The required data characterizing a PoI is obtained through symbIoTe-compliant platforms. It can then be possible for an application to use the PoI Search component to obtain a destination and subsequently to use the Green Route Calculator component to find a way to reach it.

As can be observed in Figure 29, an application makes a request to the Enabler, asking for preferred PoIs near a certain location. The PoI Search component requests from an external PoI Search Service the PoIs near the location. The PoI Search component abstracts this PoI Search Service for the application. For this use case, the OpenStreetMap (OSM) APIs is used, but others, such as Foursquare, can also be integrated and used. After obtaining requested PoIs, the PoI Search component, for each item, requests the interpolated air quality data from Interpolator on a specified segment where the PoI is located. PoIs are then filtered according to proposed preferences and finally returned to the application with all available data so the user can easily choose a desired PoI according to his/her preferences.

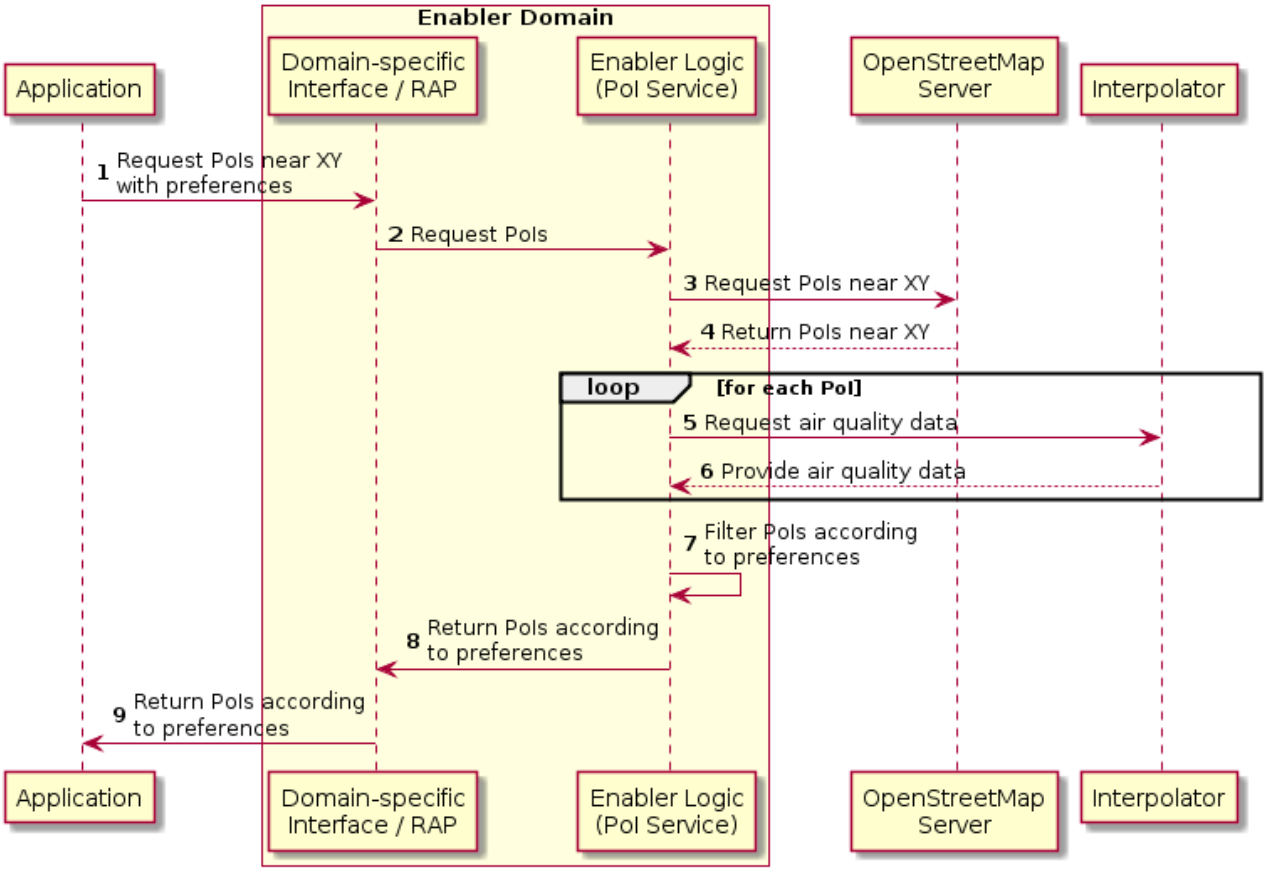

Figure 29. Point of Interest Search

Description:

- Message 1: User/Application sends a request (REST) for specified points of interest in a specified area to Domain-Specific Interface or Resource Access Proxy.
- Message 2: DSI/RAP forwards the request to Enabler Logic's Point of Interest service (RabbitMQ).
- Message 3: EL-PoI service sends a request to OSM APIs to fetch the required PoIs in a specified area.
- Message 4: OSM returns a list of found matching PoIs.
- Message 5,6: For each received PoI EL-PoI service fetches interpolated air-quality data on a street segment where PoI is located.
- Procedure 7 (optional): PoIs are filtered according to preferences if needed.
- Message 8: Results are returned to DSI/RAP (RabbitMQ).
- Message 9: Response is forwarded to Application as a HTTP-response.

5.2 Smart Yachting – Use Case

The focus of Smart Yachting is to provide advanced services for the Yachting industry based on IoT solutions. The use case is divided in two specific showcases:

- **Smart Mooring** aims to automate the mooring procedure of the port, which is quite a bureaucratic and tedious process, since Marinas operate in strongly regulated contexts. For this use case, the workflow logic is provided by a Navigo application (Portnet).
- **Automated Supply Chain** aims to automatically identify the needs for goods and services on board of the Yacht, so that automated requests for offers can be issued on the marketplace platform of the Port, provided in the use case by another application of the Navigo infrastructure (Centrale Acquisti).

Both showcases exploit data from IoT sensors to automatically acquire information from the Yacht and to pass them to the aforementioned business applications that are connected to the Port infrastructure.

We assume that the Mooring and the Supply Chain Management systems are connected to the symbIoTe ecosystem through an Enabler. This should facilitate the integration of the two business applications with symbIoTe, by encapsulating the technical details of the whole IoT infrastructure within the Enabler to expose only the minimum set of methods to the applications, while guaranteeing that the required communication and data exchange is performed to facilitate the use case.

This is particularly important, since each Port that in the future might adhere to symbIoTe for implementing the Smart Yachting use case, must integrate their Mooring and Supply Chain Management systems. We are not aiming in fact to integrate only Navigo's port applications but other software systems used in Ports: it is therefore of paramount importance to simplify how these applications can integrate symbIoTe-enabled IoT solutions into their use case.

There is a huge variety of software systems in Marinas. Their integration of the Smart Yachting use case must be as simple as possible, and totally hassle-free. In theory, they must be able to ignore the inner details of the M2M interactions (and of symbIoTe) and that's exactly the main "added value" that an Enabler provides in this use case.

As already stated, currently the Smart Yachting Enabler eases the integration of data from the yachts with data in the ports and port applications, to facilitate the mooring and supply process. Looking in perspective, this specific Enabler could offer integrated data from a larger number of ports and a larger number of yachts on a wider area, to better match the needs of yachtsmen with port facilities and offer alternative services to the yachts according to the current conditions and offerings. For instance, based on the offering in marinas, the location of the yacht and the preferences of the passengers onboard (e.g., they look for specific services in the port they are about to stop), the Enabler could recommend to the yachtsman the port at which to moor. Moreover, the enabler could present in real time the number of free places for boats or the weather forecasts in all the ports integrated in symbIoTe's Smart Yachting, so that the yachtsman can choose the best port for mooring.

The Smart Yachting use case assumes to be Level 1 (L1), Level 3 (L3) and Level 4 (L4) compliant for Smart Mooring and L1 for the Automated Supply Chain showcase: at present the implementation of the Smart Space middleware (which will be used in Smart Mooring) is still in the design phase, therefore the hypotheses presented herein are based on the current state of the analysis for L3 and L4 symbIoTe components.

5.2.1 Smart Mooring

Smart Mooring simplifies, through M2M interactions, the mooring authorization workflow. It allows the Port's workflow management system to automatically retrieve data from the Yacht that is needed for the workflow authorization.

We want to intercept a particular phase of the Mooring process, which starts when the Yacht is approaching to a destination port and ends when it finally berths into one of its piers. We assume that the initial mooring request (a sort of "booking" for the boat in the Port) always starts off-line or in any case outside symbIoTe.

The hypotheses that we are considering for Smart Mooring involve several interactions amongst the Boat, the Port IoT System and the symbIoTe components. In details:

- Since the Yacht is a roaming Smart Device (symbIoTe L4 device), its ID must be maintained in the Registry. Here two specific properties will be associated (and kept updated) for each Yacht: ConnectionStatus and ConnectedInPort. The former registers how (and if) the Yacht is connected and can assume as possible values: "disconnected", "LoRaWAN" and "WiFi". The latter, when ConnectionStatus is not "disconnected", will take as the value the ID of the Port where the Yacht is connected (note: in WP4 it has been assumed that Roaming Devices' properties are maintained in the Core Registry).
- LoRaWAN connectivity will be used: we assume therefore that a specific LoRaWAN controller is attached to the Smart Space (SSP) middleware.
- When approaching the port, the vessel – as a Smart Device (SDEV) – is detected by the SSP LoRaWAN controller and registered by it to the SSP Innkeeper (which, in turn, updates the Yacht/SDEV properties in the symbIoTe Core Registry).

- The mooring application receives through its enabler a notification that a new Smart Yacht SDEV has been registered to the SSP of the port: this event activates the mooring workflow.

When the Yacht is near the port, again through LoRaWAN, a Wi-Fi password is transmitted to the Yacht SDEV, which starts a full Internet connection. Therefore, the Yacht, as a SDEV, first connects via LoRaWAN and then connects via Wi-Fi when the latter signal is strong enough. This connection change allows PortNet's Enabler to request the retrieval of data from boat sensors.

The interactions described above are depicted in the following UML sequence diagrams. We have divided the mooring process in two parts: the first one describes the moment in which the Yacht is approaching the port. In details:

1. The Mooring Management System (PortNet) waits for the incoming Yacht; it requests to the Enabler Logic (via a Domain-specific Interface or the RAP) to detect the Yacht arrival.
2. The Enabler Logic will request the Resource Manager to query at a specified frequency (e.g. every 5 minutes), the ConnectionStatus property of the Yacht in the symbIoTe Core Registry (via the Core Search Engine), so that it can detect when it becomes different from "disconnected".
3. The Yacht approaches the port; its LoRaWAN sensor connects to the Port's Antenna and sends its ID.
4. The LoRaWAN controller, which will be supervised by the Smart Space Middleware, will update the ConnectionStatus and ConnectedInPort properties of the Yacht in the Registry.
5. The Resource Manager queries once again the symbIoTe Core and retrieves the values for the properties ConnectionStatus and ConnectedInPort to the Enabler Logic. This time the ConnectionStatus == "LoRaWAN", so Enabler Logic notifies Portnet, via the Domain-specific Interface/RAP, of the incoming ship.
6. PortNet automatically starts the mooring procedures for the specific Boat and alerts the Port Authority operators and Port personnel.

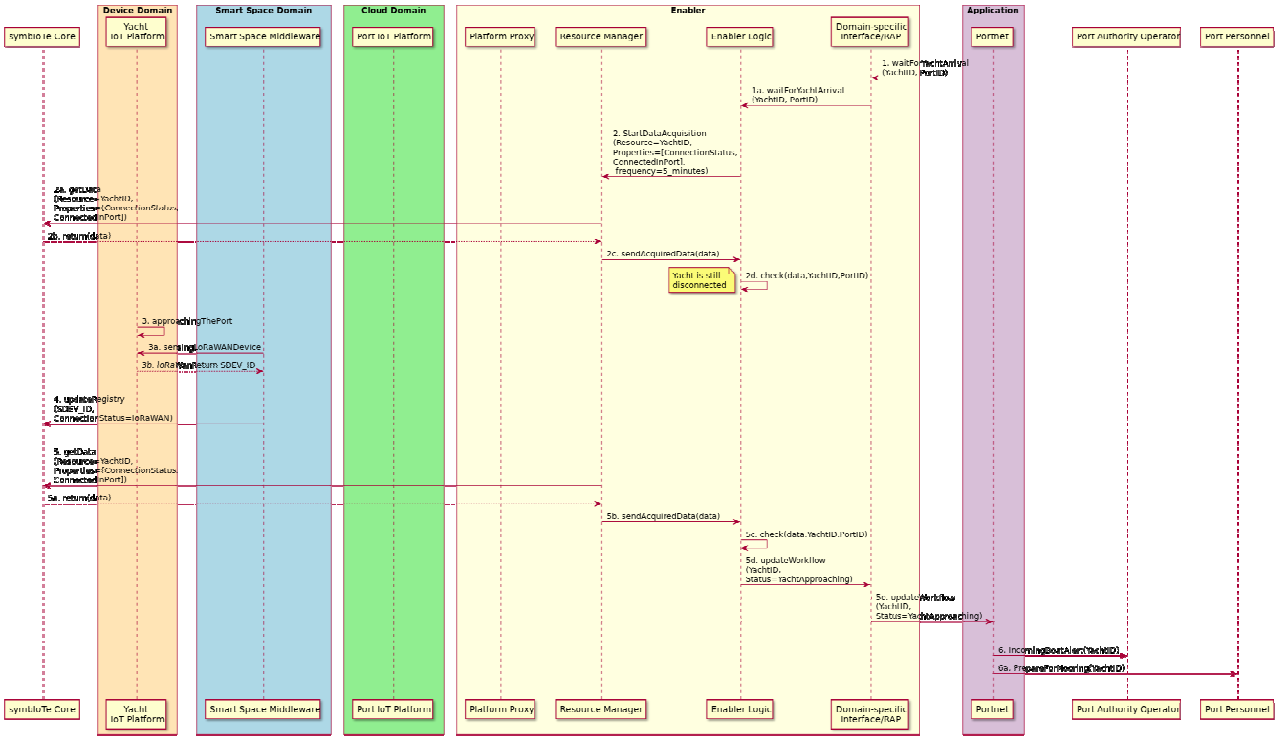

Figure 30. Sequence Diagram of the Smart Mooring showcase – 1. The Yacht approaches the Port

The second sequence diagram shows the arrival of the Yacht in the harbour and the accomplishment of the mooring procedure.

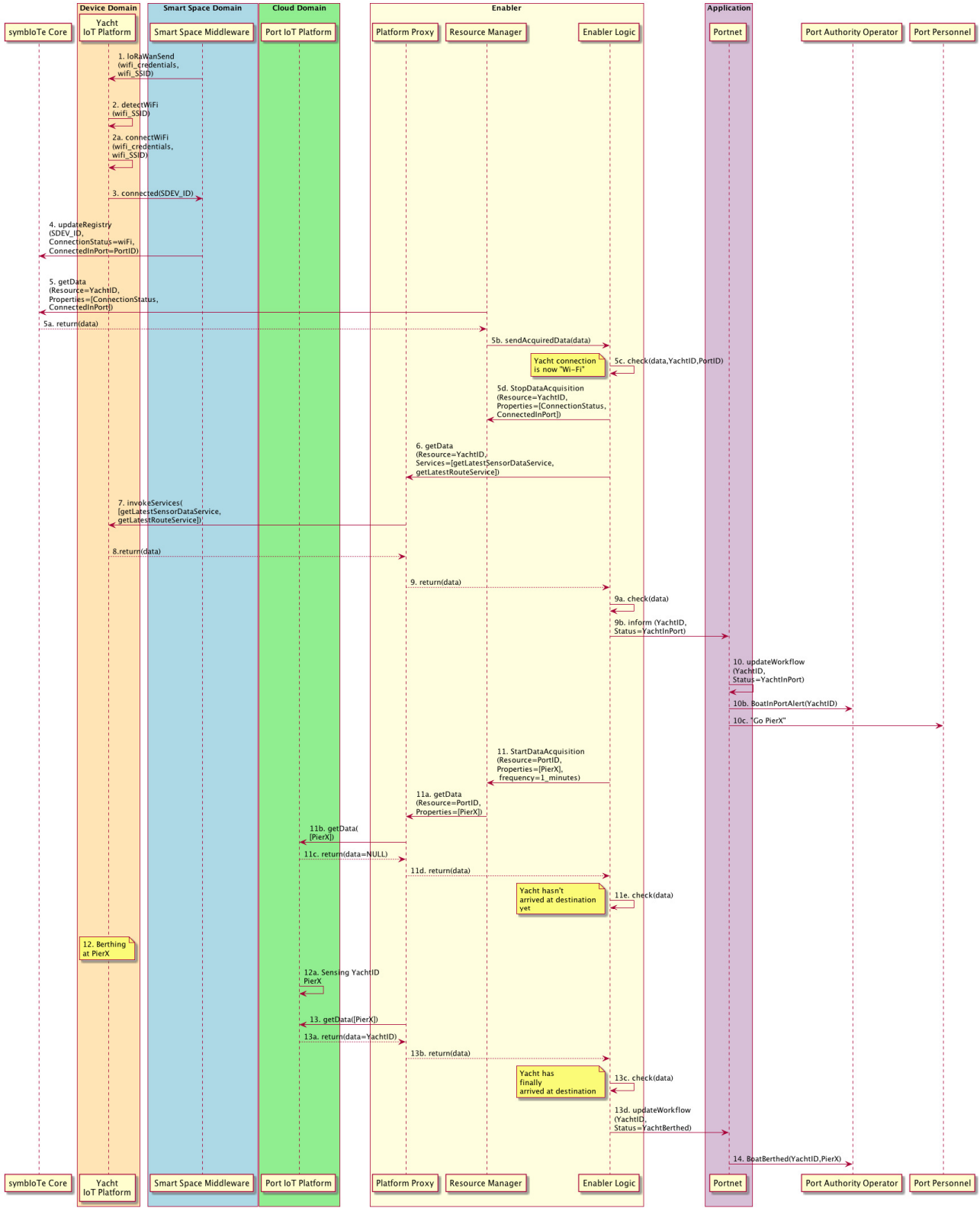

Figure 31. Sequence Diagram of the Smart Mooring showcase – 2. The Yacht arrives in the Port

In details:

- Message 1: Through LoRaWAN the Yacht receives the credentials to connect to the Port's Wi-Fi network: they will be sent by the SSP middleware (note: the actual process to implement this is still under investigation in WP4).
- Message 2: Yacht detects strong Wi-Fi signal.
- Message 2a: The Yacht connects to Wi-Fi.
- Message 3: When Yacht connects to Wi-Fi network the symbIoTe-Agent of the Yacht connects to the SSP.
- Message 4: The SSP middleware updates the Yacht property ConnectionStatus to "WiFi" in the symbIoTe Core's Registry.
- Message 5: The Resource Manager queries once again the symbIoTe Core and retrieves the values for the properties ConnectionStatus and ConnectedInPort.
- Message 5a: The symbIoTe Core returns requested data (ConnectionStatus, ConnectedInPort).
- Message 5b: Resource Manager sends to the Enabler Logic search results from the symbIoTe Core
- Message 5c: Enabler logic checks if the ConnectionStatus == "WiFi". This time it is true so it continues with following messages.
- Message 5d: Enabler Logic sends request to the Resource Manager to stop data acquisition for the Yacht's properties.
- Message 6: The Enabler Logic requests the Platform Proxy to acquire data from the Yacht IoT platform, by invoking once its two services getLatestSensorDataService and getLatestRouteService.
- Message 7: The Platform Proxy invokes the services of the Yacht.
- Message 8: The Yacht grants access and returns its data.
- Message 9: The Platform Proxy receives these data and passes them to the Enabler Logic.
- Message 9a: Enabler Logic verifies the data.
- Message 9b: Enabler Logic informs Portnet about the Yacht by sending YachtID and status.
- Message 10: Portnet updates the workflow, processes the incoming data.
- Message 10a: Portnet sends alert to the Port Authority Operator.
- Message 10b: Portnet sends alert to the Port Personnel.
- Message 11: Enabler Logic requests Resource Manager to start fetch data (with a short frequency, e.g. every minute) from the Port IoT platform, to verify if the Yacht has arrived at the specified pier.
- Message 11a: Resource Manager starts data acquisition by sending request to Platform Proxy.

- Message 11b: Platform Proxy periodically acquires data of the presence sensor attached to the Port's pier (Port IoT Platform).
- Message 11c: Port IoT Platform returns sensor reading to Platform Proxy.
- Message 11d: Platform Proxy forwards data to Enabler Logic.
- Message 11e: Enabler Logic checks data.
- Procedure 12: Yacht IoT Platform is berthing in Pier X.
- Message 12a: The Yacht arrives at the pier and through RFID sensors the Port IoT platform acknowledges its presence.
- Message 13: Platform Proxy queries the Port IoT platform.
- Message 13a: Port IoT Platform returns sensor reading.
- Message 13b: Platform Proxy sends acquired data to Enabler Logic.
- Message 13c: Platform proxy verifies that the Yacht has arrived at the correct destination.
- Message 13d: Enabler Logic updates workflow data with the yacht status to the Portnet.
- Message 14: Portnet sends an alert to the Port Authority Operator and closes the mooring workflow.

Currently the use case has been concentrated on the process of intercepting the Yacht arrival in the Port, which is the only action that the Mooring workflow applications (like Navigo's Portnet) consider. From a Smart Space viewpoint, it should be also taken in consideration the case of the Yacht leaving the port, which must induce to update the Yacht properties in the Core. At present, it hasn't been finalized yet how to face this situation: we can expect for example an explicit "departure" action performed by Yacht's symbIoTe Agent but generally speaking we will use the same design pattern that emerges in WP4 to manage the use case of a Roaming Device leaving a Smart Space.

5.2.2 Automated Supply Chain

The Automated Supply Chain (ASC) workflow can be seen as a particular case of the latter part of the Mooring use case. The Centrale Acquisti web application (by Navigo), through an Enabler, accesses the resources of the Yacht (sensors) to retrieve information about the needs (of goods or services) onboard.

We will integrate Navigo's Supply Chain application (Centrale Acquisti) but the overall goal is to implement a general solution, as simple and straightforward as possible, that allows any software vendor that has a Supply Chain system for Marinas to make it compatible with this use case. We want to hide all the inner details about the interactions with the yacht sensors and the symbIoTe infrastructure by encapsulating the integration logic in the Enabler.

In order to make Smart Yachting successful we have to guarantee that the largest numbers of ports (and therefore of their IT systems) can support easily the use case. We are not aiming in fact to integrate only Navigo's port applications but other software

systems used in Ports as well, which are quite often implemented with simple and not modern technologies (e.g. Access, Visual Basic, …).

Like in the previous case, we assume that the showcase will always start off-line or in any case outside symbIoTe.

The following sequence diagrams illustrate the steps of the Automated Supply Chain showcase.

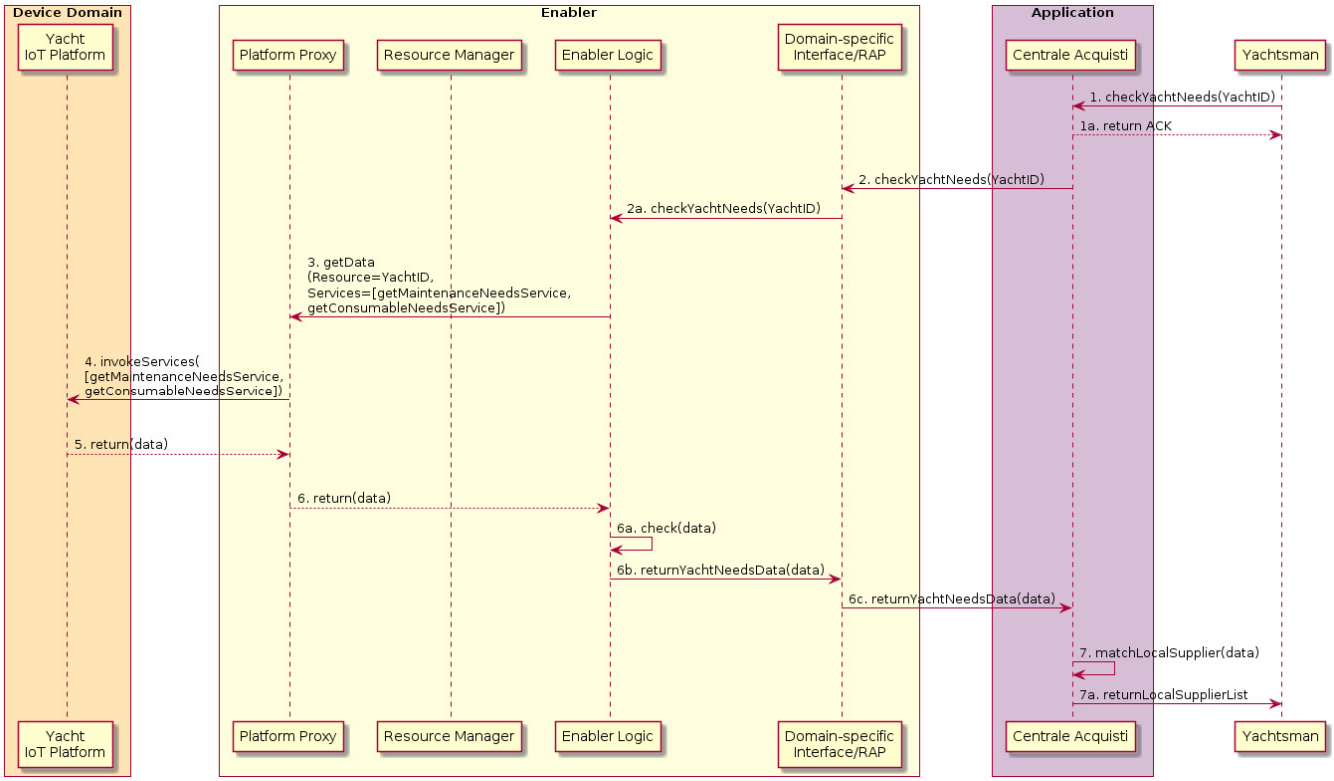

Figure 32. Sequence Diagram of the Automated Supply Chain showcase

Message details:

- Message 1: While the yacht is berthed in a "smart" port, a yachtsman connects to the Centrale Acquisti (CA) supply chain application through a common web browser.
- Message 1a: CA acknowledges the receipt of yacht's registration.
- Message 2: CA sends request to the Domain-specific Interface/RAP to acquire data from the Yacht.
- Message 2a: Domain-specific Interface/RAP forwards request to Enabler Logic.
- Message 3: Enabler Logic requests Platform Proxy to invoke once two services of the Yacht, getMaintenanceNeedsService and getConsumableNeedsService.
- Message 4: Platform Proxy sends request to the yacht's IoT platform to invoke those services.
- Message 5: The yacht's IoT Platform grants access, invokes the services and returns the results to the Platform Proxy.

- Message 6: Platform Proxy returns these data to the Enabler Logic.
- Message 6a: Enabler Logic verifies received data.
- Message 6b: Enabler Logic returns the information to the Domain-specific Interface/RAP.
- Message 6c: Domain-specific Interface/RAP returns data to CA.
- Message 7: CA tries to find a possible match with the suppliers of goods and services in the port area.

Message 7a: CA returns to the yachtsman a list of proposals for supplies. The yachtsman will manage these proposals in the CA web application.

5.3 Smart Residence – Use Case

The Smart Residence use case aims to demonstrate interoperability across different smart home IoT solutions through a generalized abstract model to describe inter-connected objects, providing a dynamic configuration of available services and a natural and homogeneous user experience.

A health monitoring system, in addition to the smart living platform, has the ability to create a comfortable, safe and helpful living/residence environment, supporting a scenario where residents are provided with context-aware and personalized health and comfort services at home.

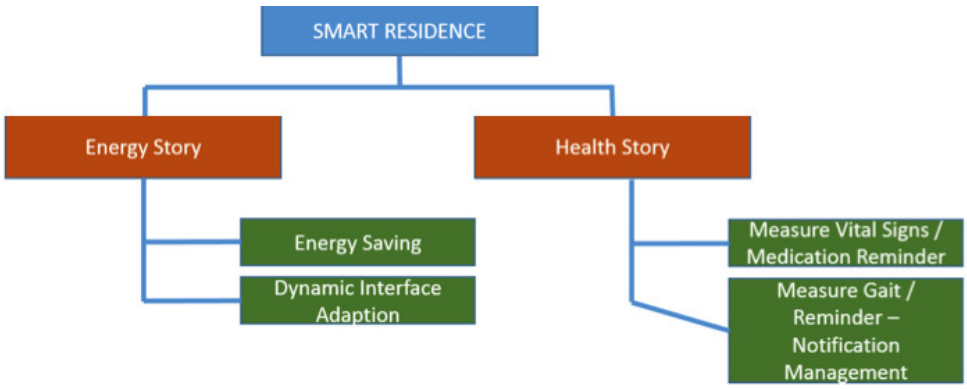

Figure 33. Smart Residence showcases

5.3.1 Location based resource filtering

The Energy Story showcase (Figure 33Figure 33. Smart Residence showcases

) presents a couple of typical home automation situation, where the environment seamlessly adapts to meet its users' needs: the first scenario is the Energy Saving, where the surrounding changes according to predefined user comfort values, the second scenario is the Dynamic Interface Adaptation, with a user control interface able to manage the devices in range. In both cases the system must be able to select physical devices based on their location inside the space; moreover, this property must be addressed as a symbolic location (e.g. room, floor, etc.) and not by coordinates. Enabler facilitates the implementation of Smart Residence applications by offering an indoor location system for

the Smart Space (implies L3 compliance): this was necessary as the GPS system is not suitable for indoor location purposes.

SSPs are defined as physical areas where one or more IoT platforms coexist, each of them providing some kind of service: in this way, they define abstract boundaries for the IoT services and platforms they embrace. Different platforms, though, can use different names for the same spaces, areas and locations. For example, a platform could split a building in many *floors*, and register devices indicating to which floor they belong, as well as another platform can use *rooms* to indicate symbolic locations for its resources. This presents is a different way to subdivide the same building.

It is necessary for this use case to manage all the symbolic locations defined for an SSP, to classify them in some kind of a tree view, to allow the Enabler to understand the actual hierarchy of the SSP's topology. This operation is provided by a manual configuration made by the SSP administrator, that must be able to rearrange all the registered areas to specify whether one *contains* or *is contained* by another (e.g. floors in buildings, rooms in floors, etc.); for this purpose, an administration console has to be provided to the SSP supervisor.

The Location based resource filtering Enabler must be able to filter devices of the Smart Space basing on their registered position (which can be a building, floor, room, etc.), after the results gathered from the Search component. This means that an additional filtering is applied to the result coming from the symbIoTe Core Services. To achieve this goal, the Enabler will implement and register a Service in the Core.

For instance, we could have an SSP where some resources are registered with *bedroom A* as location and some others with a more generic *first floor* (obviously bedroom A is a room located at the first floor). Whether the application has searched for resources at *first floor*, the Enabler Logic, by knowing the hierarchy of locations, will include bedroom resources in results, in addition to the ones registered as located at the *first floor*.

The use of an Enabler is motivated by the fact that not all Smart Residence environments need to have such hierarchy and that, in cases it is needed, it is obviously specific to this particular use case.

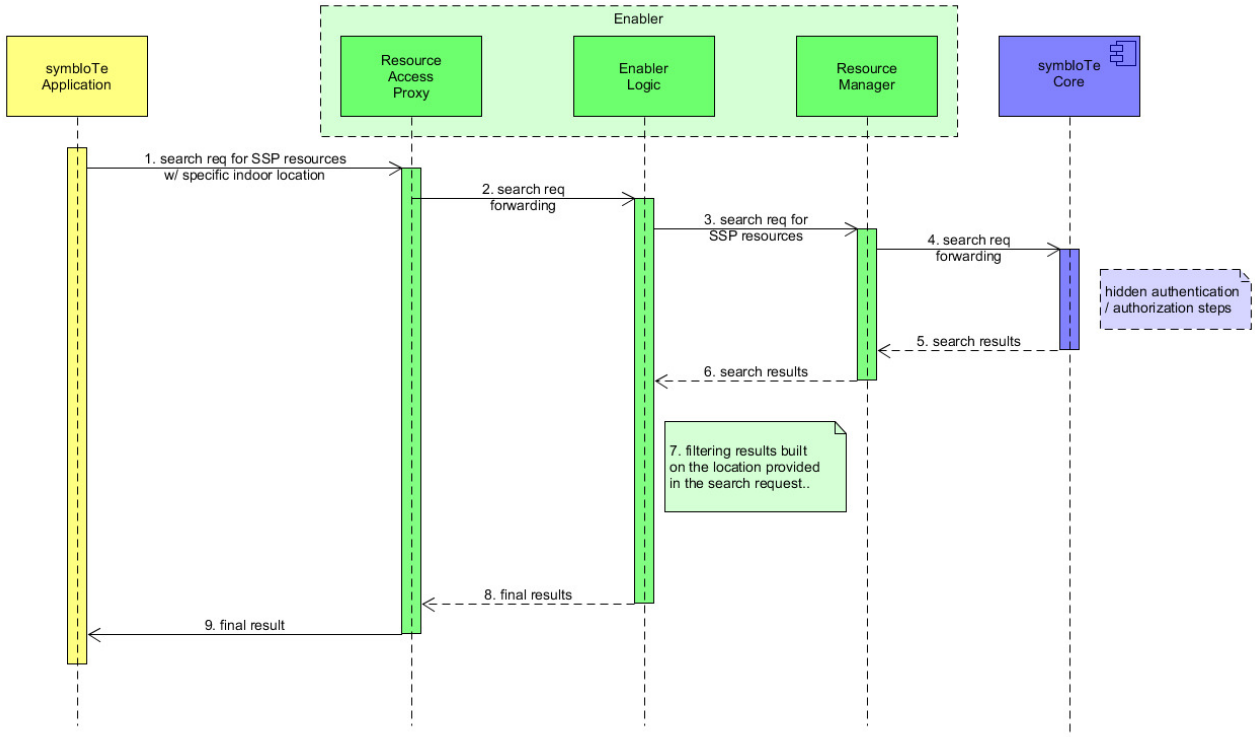

Figure 34. Indoor Location search

Every time an application needs the list of resources based on their location, the sequence of events is triggered, as shown in Figure 34 (for hidden steps see Section 3.3.8 and Figure 10):

- Massage 1: Application sends a search request specifying the SSP and one of its locations (e.g. "bedroom").
- Message 2: The Enabler Resource Access Proxy receives the request and forwards it to the Enabler Logic.
- Message 3: Then the Enabler Logic, through the Resource Manager, sends the following message.
- Message 4: Resource Manager queries the Core Search component in order to retrieve all the resources registered in the SSP.
- Message 5: Core Search component returns search results.
- Message 6: Resource Manager forwards search results to Enabler Logic.
- Procedure 7: The results are filtered by the Enabler Logic basing on their indoor location.
- Message 8: The final list of resources in specified location is sent to the Resource Access Proxy.
- Message 9: Resource Access Proxy returns filtered results to the Application

Locations need to be registered within the Core Services in accordance with the Information Model specified by the platforms involved in this use case and those locations must be *human understandable definitions*, to allow the SSP administrator to be able to manage their relationships.

5.3.2 Smart healthy indoor air

S&C will develop an application based on the indoor/outdoor air quality monitoring and pursues to improve indoor air quality through recommendations. Also, the application will monitor the sleep quality by using a sensor that will be located beneath a mattress. The device tracks heart rate, breathing, and movement without requiring the user to wear anything.

Air pollution does not just have a negative impact on the environment. The World Health Organization has found that air pollution may be a predictor of poor sleep. Improving indoor air quality may be one way to enhance sleep health and comfort.

The S&C's platform named nAssist (symbIoTe L2-compliant) will acquire, store and process all data characterizing indoor air quality, e.g. temperature, humidity and CO sensors will be used. Without this information, usually we ventilate rooms late and for too long. This application (smart healthy indoor air) will indicate when, how and for how long a room should be ventilated taking into consideration that windows is the easiest ventilation option but not the healthiest one depending on the outdoor air quality. There are other ways to provide healthy flow of air throughout the room, such as turning on the air conditioner or individual air purifiers. Also, this application will correlate sleep quality with indoor air quality by monitoring the sleep pattern.

Given the limited number of fixed monitoring stations available and placed at representative spots, an accurate assessment of outdoor spatial variation of air quality is highly required. Spatial interpolation techniques applied to the available monitoring data to provide air quality information closest to the location of the smart home to be monitored. This functionality will be provided by the component named as Interpolator, implemented within the module Enabler Logic. The outdoor air quality can be measured through the nAssist platform or by using external services, e.g. the Network of Vigilance and Forecast of the Quality of the Air provided by the Catalan Government.

nAssist will make the outdoor sensors and their data available to symbIoTe. This application will send the GPS location of the smart home and will get from symbIoTe the estimated value about the air quality for this specific location which will use the SMEUR Enabler – Data Interpolation workflow (section 5.1.3).

Also, nAssist will offer the sleep quality information acquired by the sensor located beneath the mattress. The monitoring of sleep quality and vital signs could be done through the Smart Health Mirror developed by AIT in combination with the health parameters monitored by SMILA (Smart Mirror Living Assistant). In this way, symbIoTe will enable to extend the current SMILA application by including new health indicators acquired by the platform nAssist and also extend the 'Smart Healthy Indoor Air' application by including other health indicators that could extend the assessment of the environmental impact of poor indoor air quality on human health. Details of the complete use case are included in D5.2 [8].

5.4 Smart Stadium – Use Case

Indoor location (and subsequently navigation) is a field in which existing solutions are not providing a universal and generally applicable solution, though there is a growing need to provide a universal and generally useable solution.

Outdoor location and navigation is generally provided by Global Navigation Satellite Systems (GNSS) such as GPS, Galileo and GLONASS which provide very precise outdoor positioning. With the introduction of satellite positioning functions in almost all smartphone platforms, coupled with internet-based maps and navigation systems, many very useful applications have been developed and are widely used (from general positioning, street turn-by-turn navigation, to live route recalculation coupled with real-time traffic information). However, outdoor location and navigation cannot be used for indoor location where GNSS signals cannot penetrate. Satellite navigation also cannot be used in long street/urban canyons with high aspect ratio or near large infrastructures which obscure large portions of the sky.

Various proposals and methods for indoor location exist today, but most of them are limited in precision or scope. Typical solutions are based on WiFi (discovering nearest WiFi hotspots) which can provide location with a precision of 5 to 10m. Such systems can be augmented with Bluetooth beacons which can improve the precision to 1-3m range, but require a large number of additional beacons to be installed and managed in indoor spaces.

The best precision is achieved by proprietary solutions using Ultra-Wideband transmitters and locator tags, bringing the precision to 10-30cm in controlled environments where such precision is required usually by industrial application – locating and managing autonomous fork-lifts in warehouses or manufacturing spaces.

Within 3GPP-based mobile networks, Location Services (LCS) are included as part of the standards since the beginning, mostly based on the data extracted from the OSS (Operations Support Systems) environment. Lower precision is obtained by detecting the cell in which the user equipment (UE) currently resides (typically by capturing data on the probes monitoring the signaling traffic and detecting the events such as Location Updates when a user changes an active cell, or other types of events such as calls, text messages, data sessions, which also contain the information about the active cell). Enhanced positioning tools extract the data from radio interfaces and can calculate a position more precisely by triangulation of the signals received from multiple neighboring radio towers.

Within initiatives for further development of next-generation mobile networks (grouped within 5G), technical specification (TS) 22.261 for Release 16 defines the requirements for UE positioning based on hybrid approaches using both the 3GPP and non-3GPP technologies. The aim is to provide a standardized solution for a “higher-accuracy positioning” with an expected precision of 30-50cm (even 10 cm in autonomous vehicle use case), covering both indoor and outdoor usages. Since deployments of 5G networks based on Release 15 are expected in the end of 2018 and based on Release 16 in 2020, standardized solutions for higher-accuracy UE positioning in 5G networks will not be available during the lifetime of the symbIoTe project.

5.4.1 Network-based Location Enabler

Similar to the requirements for UE positioning in Release 16, a Network-based Location Enabler (NLE) is using a hybrid solution for indoor location, collecting the data from all available location providers (other Platforms) and then performing the calculation of the UE position which is then returned to a requesting user.

A typical problem for indoor location and navigation is also that actual coordinates in the indoor environment are not very useful, and that a “human-readable” definition of indoor location is required. Thus, symbIoTe applies the Smart Space (SSP) concept to different indoor environments, e.g., campus, residence, shopping mall, parking space or stadium/arena, proposing a tree-like structure to define symbolic locations within SSPs, as defined in Section 5.3.1, where the tagging of areas within an SSP needs to be performed by an administrator for each SSP.

Indoor location is performed by a user device (typically a smartphone) as a sensor which detects various network technologies present in its environment (Cell information from mobile network, nearby WiFi access points, visible Bluetooth beacons and any other possible sources) and sends this information as a parameter in a location query to a NLE. NLE needs to know which platforms it can query to receive relevant information for a certain SSP. It can query the symbIoTe Core Services to identify relevant platforms for the SSP. Alternatively, the list of such platforms can be predefined since indoor navigation is typically used by smartphone applications tailored for a certain SSP (e.g., applications provided by event organizers in Smart Stadium/Arena, shopping mall guide application, home management system for Smart Residence).

5.4.2 Source platforms for an Enabler

Main problem of all indoor location systems (and also in general of WiFi-based location services) is quality of the underlying data where data is typically collected by wardriving or crowd sourcing where even if data is collected from the air, when it’s sent to the collecting system it’s not precisely located (as sensor collecting it cannot know from GPS or any other system its precise indoor location). Only way to improve this is by infrastructure owners or network owners to publish exact positions (including the vertical coordinate and symbolic location description) of WiFi hotspots, BLE beacons and mobile network components, together with the indoor map of the area.

The usability of mobile network data for indoor positioning and navigation dramatically improves with new designs of indoor coverage systems, which are deployed by utilizing smaller cells and more precise antenna systems (beamforming and massive MIMO), especially in the areas where high user density is expected (stadiums and closed arenas, airports, shopping malls, office buildings, public buildings) or in the outdoor cases where sky visibility is limited (urban canyons, large structures).

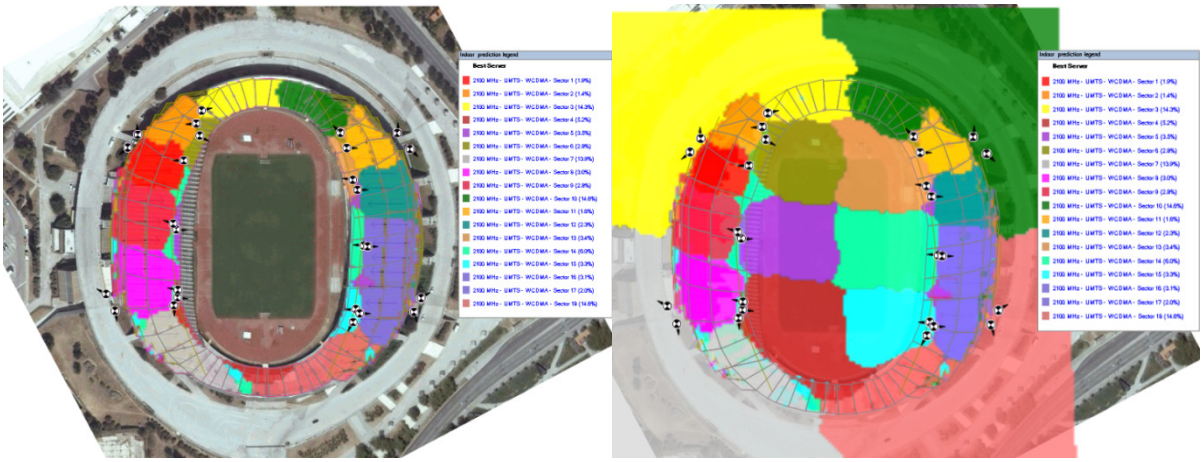

Figure 35. Cellular network coverage map (semi-closed stadium)

Therefore, an NLE would need to send a query for indoor location to various sources, which could (but not limited to) include the following:

- Mobile network OSS – reports UE's currently active and neighboring cells, especially in small cell indoor scenario where location and coverage of the small cell can be exactly mapped from the design and installation documentation to the location and provided indoor map.
- WiFi infrastructure from the manager of the venue/campus – extracting and publishing information from installed WiFi infrastructure, combining the physical installation information with installation from WiFi network controllers (such as Cisco WLC platform or Ubiqutiy airControl or other such systems).
- Bluetooth beacons deployed and managed by the venue/facility which are precisely mapped to the indoor map.

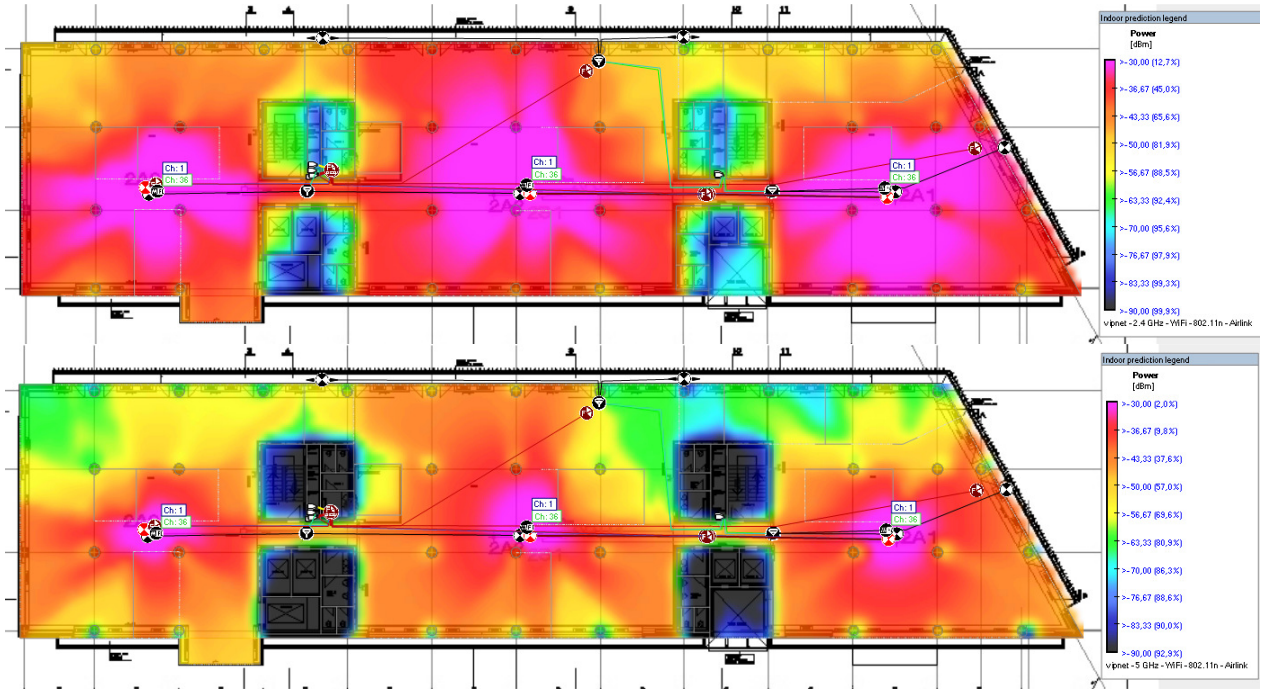

Figure 36. Indoor WiFi coverage map (an office building example)

Due to fragmentation of potential sources, an Enabler is appropriate to provide symbolic location information for indoor spaces since the provided service is indeed generic, but also domain-specific as it depends on the configuration of each SSP where location information is needed. For example, a NLE can query the symbIoTe Core to find adequate platforms which provide location or positioning services for a certain SSP, and each platform responds with a position (e.g., physical coordinates, including the height). The NLE then utilizes a workflow similar to the Interpolator from SMEUR use case to provide an estimate of symbolic location which is usable within the SSP for end user applications.

5.4.2.1 Enabler Workflow

Indoor location service would be deployed on a UE as part of a Client Application provided by an event/facility/campus manager (Music Festival application, Sport club, shopping mall, campus/organization application) or abstractly as a service to other applications. The NLE workflow is depicted in the diagram included in Figure 37.

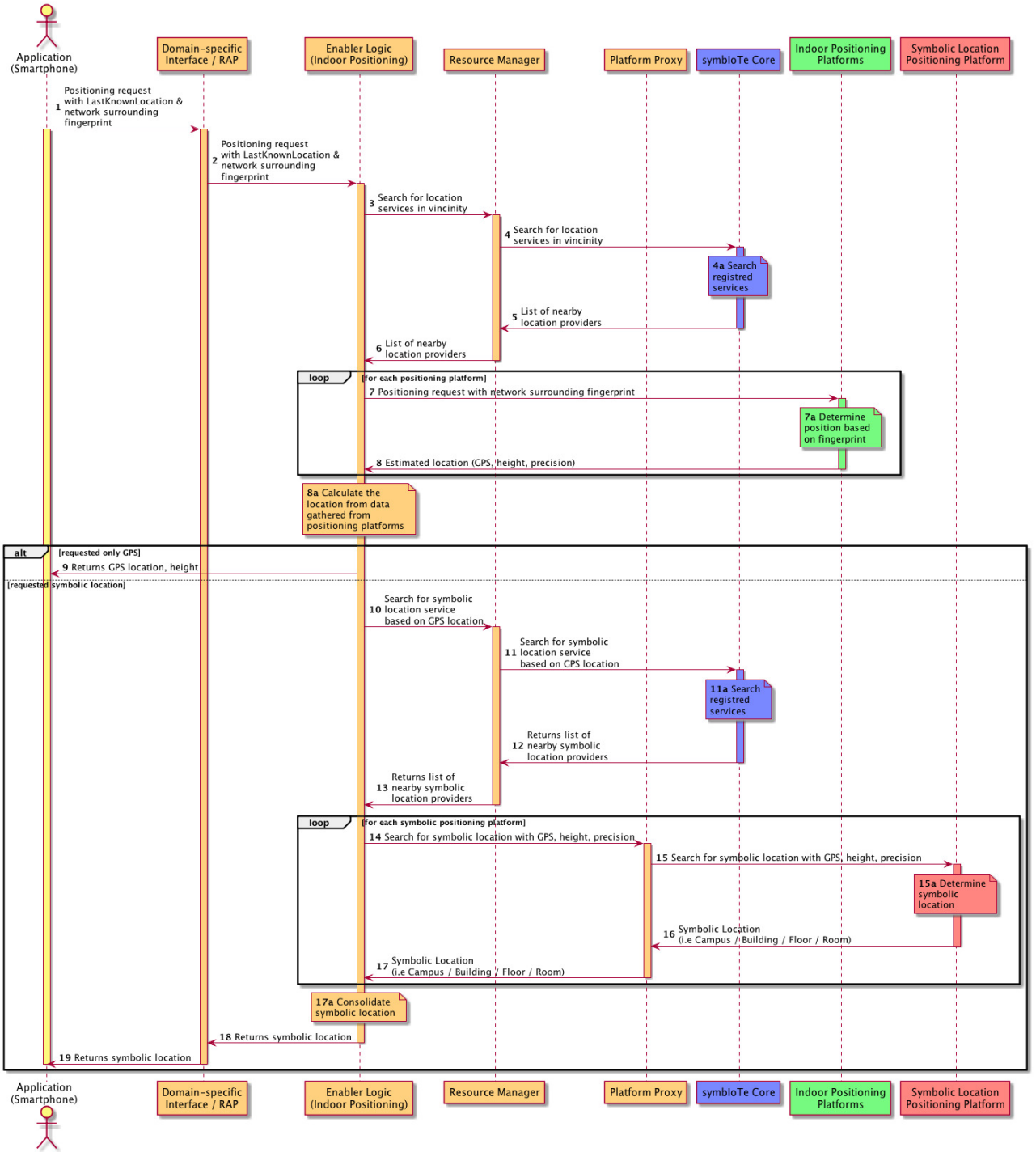

Figure 37. NLE workflow for indoor positioning

- Message 1: A client detects its approximate last-known position from the user device (can be read on certain platforms) and the network fingerprint information. The client sends them in a positioning request to the Enabler (Domain-specific Interface or Resource Access Proxy, DSI/RAP).
- Message 2: DSI/RAP forwards the positioning request to Enabler Logic which includes an Indoor Positioning component.
- Message 3: Enabler Logic sends the request to Resource Manager to search for indoor location services in the vicinity of client's last-known position.

- Message 4: Resource Manager sends request to the symbIoTe Core for indoor location services in the vicinity of client's last-known position.
- Procedure 4a: symbIoTe Core searches its DB.
- Message 5: The list of nearby Indoor Location Provider services is returned to the Resource Manager.
- Message 6: Result from the symbIoTe Core is returned to Enabler Logic.
- Messages 7-8 are sent to each indoor positioning platform.
- Message 7: Enabler Logic sends the network fingerprint information to an indoor positioning platform.
- Procedure 7a: Positioning service determines client position based on the network fingerprint information.
- Messages 8: Indoor positioning platform returns location estimation (GPS position, vertical position and precision).
- Procedure 8a: Enabler Logic calculates the position from the received data by doing a weighted average of the data received from multiple positioning platforms.
- Message 9: Enabler could return the numerical location to the Client,
- Message 10: An alternative is to find symbolic location from numeric position. Enabler Logic sends request to Resource Manager to search for symbolic location service.
- Message 11: Resource Manager searches the symbIoTe Core for symbolic location service for a specified GPS location.
- Procedure 11a: symbIoTe Core searches its DB.
- Message 12: symbIoTe Core returns list of nearby symbolic location service providers.
- Message 14: Enabler Logic sends request to Platform Proxy to send one time request to the Symbolic Location Positioning Platform with GPS location, height and precision.
- Message 15: Platform Proxy forwards request to Symbolic Location Positioning Platform.
- Procedure 15a: Symbolic Location Positioning Platform converts location based on numeric position data (GPS location, vertical position, precision) to the symbolic location (building/floor/room).
- Message 16: Symbolic Location Positioning Platform returns symbolic location to the Platform Proxy.
- Message 17: Platform Proxy returns result to Enabler Logic.
- Procedure 17a: Enabler needs to consolidate symbolic location if there is more than one symbolic location determined by Symbolic Location Positioning Platform
- Message 18: Enabler Logic returns to DSI/RAP consolidated symbolic location
- Message 19: DSI/RAP returns symbolic location to the Client Application.

Exact numerical location can be useful for displaying client position on an indoor map provided within the Client Application (especially if Client Application uses an indoor map), while additional value is given by the symbolic location information, provided by a facility management Platform that can translate the numerical positions into symbolic tree (campus/building/floor/room or stadium/stand/sector), as human-readable information can be more appropriate for a number of end-user applications, as already stated in Section 5.3.1use case.

6 Implementation of Smart Mobility & Ecological Urban Enabler

The design of SMEUR Enabler is included in Section 5.1. This section includes implementation details of the SMEUR-specific components with pointers to component source code and Javadoc on symbIoTe GitHub repository.

6.1 Domain-Specific Interface

Component/service name	Domain-Specific Interface
URL of source codes	https://github.com/symbiote-h2020/DomainSpecificInterfaceSMEUR
URL of Javadoc documentation	https://symbiote-h2020.github.io/DomainSpecificInterfaceSMEUR/doxygen/
List of 3rd release features included	• Expose REST-interface for the usage of Point of Interest service • Synchronous communication with EL-PoI component to process user requests

Table 15. Enabler Logic - Domain-Specific Interface interfaces

#	Interface	Name	Msg type	From	Msg Consumers	Address/Queue	Payload	Description
1	POI search request	POI request	HTTP	Application/ User	DSI	HTTP endpoint	Latitude, longitude, radius, amenity	POI request with parameters
2	Green route request	GRC request	HTTP	Application/ User	DSI	HTTP endpoint	From & To Locations (lat/lon) Mode of Transportation Optimization Criteria	Green route request with parameters

6.2 Enabler Logic – Green Route Controller

Component/service name	Enabler Logic – Green Route Controller
URL of source codes	https://github.com/symbiote-h2020/EnablerLogicGreenRouteController
URL of Javadoc documentation	https://symbiote-h2020.github.io/EnablerLogicGreenRouteController/doxygen/
• **List of 3rd release features included**	• Provides Interpolator with Street Segments to be used • Provides Routing Services with Interpolated data • Provides interface to request routes from Routing Services

Table 16. Enabler Logic - Green Route Controller interfaces

#	Interface	Name	Msg type	From	Msg	Address /	Payload	Description

					Consumers	Queue		
1	Push Air Quality Data		RabbitMQ	Interpolator	Green Route Controller		Updated list of streets' OSM ids and respective air quality (for areas of preference of the service)	End point to receive updates from the interpolator, that then propagates these updates to the routing services that want them
2	Send Route Request		RabbitMQ	Domain-Specific Interface / Resource Access Proxy	Green Route Controller		From & To Locations (lat/lon) Mode of Transportation Optimization Criteria Returns: Route Geometry (Polyline) Route Properties (distance, travel time, air quality rating) Navigation Instructions	End point that receives route requests from applications and sends them to the correct routing service

6.3 Enabler Logic – Point of Interest Search

Component/service name	**Enabler Logic – Point of Interest Search**
URL of source codes	https://github.com/symbiote-h2020/EnablerLogicPoISearch
URL of Javadoc documentation	https://symbiote-h2020.github.io/EnablerLogicPoISearch/doxygen/
List of 3rd release features included	• Synchronous communication with Domain-Specific Interface • Processing of received request, and formatting HTTP request for fetching PoIs in a specified area • REST communication with OpenStreetMap-API (overpass-api) • Synchronous communication with EL-Interpolator

Table 17. Enabler Logic - Point of Interest Search interfaces

#	Interface	Name	Msg type	From	Msg Consumers	Address/Queue	Payload	Description
1	PoI search request		RabbitMQ	RAP/ DSI	EL-PoI		Latitude, longitude, radius, amenity	PoI request
2	Request PoIs in a specified area		HTTP-GET	EL-PoI	OpenStreetMap Overpass-API	Overpass-api URL	Amenity, Bounding box	Query for searched PoIs
3	Request interpolated data for each PoI		RabbitMQ	EL-PoI	EL-Interpolator		Map<Id, Point>	Air quality data for each PoI

6.4 Enabler Logic-Interpolator

Component/service name	Enabler Logic - Interpolator
URL of source codes	https://github.com/symbiote-h2020/EnablerLogicInterpolator
URL of Javadoc documentation	https://symbiote-h2020.github.io/EnablerLogicInterpolator/doxygen/
List of 3rd release features included	• Management of Regions (Street Segments) • Interpolation of measurement values with respect to street segments • (simplified) search for PoI's (simplified as no extra interpolation is done but the nearest street segment is used as reference)

Table 18. Enabler Logic - Interpolator interfaces

#	Interface	Name	Msg type	From	Msg Consumers	Address/Queue	Payload	Description
1	Register Regions		RabbitMQ	GRC	ELI		Street Segments Requested Properties RegionID Auxiliary Information	Make the street segment list and the requested ObservedProperties known to the ELI
2	Query sensors		RabbitMQ	ELI	Resource Manager		RegionID	Query Sensor readings for Region
3	Sensor Readings		RabbitMQ	Resource Manager	ELI		Observations	Sensor readings as requested in #1
2	Query Interpolated Values		RabbitMQ	GRC	ELI	StreetSegments with interpolated values		Get interpolated values for a region
2a	Push interpolate values		RabbitMQ	ELI	GRC	StreetSegments with interpolated values		Alternative to #2. Not requested but pushed as soon as available
3	Query PoI interpolate values		RabbitMQ	ELPoI	ELI	PoIs with interpolated values		Request interpolated values for a set of PoIs

Note: Due to the availability of numerical libraries needed for the interpolation algorithm, the interpolator is implemented in Python. The program is a command line oriented one that is invoked from the ELI with data exchange through files and command line arguments.

The program is not implemented as a microservice and has thus no RESTfull interface in the sense as requested here.

6.5 Implementation Summary

The current document reports the current implementation status of symbIoTe Enablers. The outcome of this work at the current stage is the final design of generic components for symbIoTe Enablers, design of 4 Enablers that will be used in symbIoTe use cases, as well as implementation details of both generic Enablers' components and specific components for SMEUR Enabler. The relevant source code and its documentation are published as open source under the BSD-3-Clause licence in the GitHub service organized in GitHub super-projects: generic Enabler components https://github.com/symbiote-h2020/SymbioteEnabler and SMEUR Enabler: https://github.com/symbiote-h2020/EnablerSMEUR.

There are 3 generic Enabler components implemented and 3 supporting projects with separate repositories. Other generic components are reused from SymbioteCloud super project. For the SMEUR Enabler there are 4 components implemented and 4 supporting projects.

The generic Enabler modules are: EnablerLogic, EnablerPlatformProxy, EnablerResourceManager. Reused modules from the SymbioteCloud: Interworking Interface, Monitoring, Registration Handler, Resource Access Proxy, and Security Handler (contained in SymbioteLibraries). The supporting projects are: EnablerConfigService, EurekaService and ZipkinService.

The SMEUR Enabler modules are: DomainSpecificInterfaceSMEUR, EnablerLogicGreenRouteController, EnablerLogicInterpolator, EnablerLogicPoISearch. Reused generic enabler modules are: EnablerLogic, EnablerPlatformProxy, EnablerResourceManager. Reused modules from SymbioteCloud are: Interworking Interface, Monitoring, Registration Handler, Resource Access Proxy, and Security Handler (contained in SymbioteLibraries). The supporting projects are: SMEURLibraries, EnablerConfigService, EurekaService and ZipkinService.

7 Conclusion

The document reports the generic architecture of symbIoTe enablers including a design of generic Enabler components which are developed to serve as a framework for design and implementation of future symbIoTe domain-specific enablers. The entire process of Enabler implementation is showcased on a Domain-Specific Enabler for SMEUR: The SMEUR Enabler is designed, mapped to the generic architecture, and all enabler-specific components are implemented. This demonstrates that generic Enabler components are indeed usable in practice. In addition, four Enablers relevant to symbIoTe use cases have been defined and their design is reported in this deliverable.

As future work, the designed Enablers need to be implemented following the SMEUR Enabler example to be used by applications which are implemented in Task 5.2. During this implementation process, future improvements or bug fixes of generic Enabler components will be performed as well as further refinement of user (development) documentation.

8 References

[1] Open Mobile Alliance (OMA): *OMA and Machine-to-Machine (M2M) Communication,* annual report, 2011, url: http://openmobilealliance.org/static/oma-annual-reports/documents/oma%20collateral%20m2m%205-11.pdf

[2] Tom Rebbeck, Analysys Mason: *Telecoms Operators' Approaches To M2M and IoT*, whitepaper, 2015, url: http://www.analysysmason.com/Research/Content/Reports/M2M-IoT-operators-approaches-May2015/

[3] The symbIoTe Consortium, "D1.2 – Initial Report on System Requirements and Architecture", 2016.

[4] The symbIoTe Consortium, "D1.4 – Final Report on System Requirements and Architecture", 2017.

[5] symbIoTe GitHub account with code sources: https://github.com/symbiote-h2020

[6] The symbIoTe Consortium, "D2.3 – Report on symbIoTe Domain-Specific Enablers and Tools", 2017.

[7] The symbIoTe Consortium, "D2.5 – Final symbIoTe Virtual IoT Environment Implementation", 2017.

[8] The symbIoTe Consortium, "D5.2 – Symbiosis of smart objects across IoT environments", 2017.

[9] The symbIoTe Consortium, "D2.2 – symbIoTe Virtual IoT Environment Implementation", 2017.

[10] The symbIoTe Consortium, "D1.3 – Final Specification of Use Cases and Initial Report on Business Models", 2017.

[11] The symbIoTe Consortium, "D5.1 – Implementation Framework", 2017.

9 Glossary

Application developers	build IoT applications based on the IoT services exposed by various IoT platforms (reside at the symbIoTe APP domain).
Core Services	services in symbIoTe Application Domain enabling applications and Enablers to find desired resources from underlying IoT platforms; and enabling IoT platforms to offer their resources to applications and Enablers
Enabler developers	build domain-specific functionalities within symbIoTe-provided Enablers to facilitate cross-platform application development
Enabler resources	resources offered by Enablers to Application developers. They are created by processing Underlying resources from IoT platforms according to domain-specific functionalities
External services	services outside IoT platforms that can be used within Enablers to add value to symbIoTe provided data
L1 Compliant Platform	an IoT platform that registers its resources to Core Services, and opens its Interworking Interface to enable applications and Enablers to access those resources
Resource	a uniquely addressable entity in symbIoTe architecture and, as a generic term, may refer to IoT devices, virtual entities, network equipment, computational resources and associated server-side functions (e.g., data stream processing). This definition is on purpose highly generic and abstract to allow its unified, recursive use across all layers of the envisioned symbIoTe stack.
Smart Space	physical environments (e.g. residence, campus, vessel, etc.) with deployed things where one or more IoT platforms provide IoT services.
Underlying resources	resources from underlying IoT platforms used by Enablers. Enabler Logic functions process these resources to create Enabler resources and offer them to Application developers

10 Abbreviations

ABAC	Attribute-Based Access Control
APP	symbIoTe Application Domain
CLD	symbIoTe Cloud Domain
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
MoBaaS	Mobility Backend as a Service
OGC	Open Geospatial Consortium
OSM	OpenStreetMap
OWL	Web Ontology Language
PoI	Point of Interest
RAP	Resource Access Proxy
RDF	Resource Description Framework
REST	Representational state transfer
RS	Routing Service
SMEUR	Smart Mobility and Ecological Urban Routing