Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Report on symbIoTe Domain-Specific Enablers and Tools

The symbIoTe Consortium

Intracom SA Telecom Solutions, ICOM, Greece
Sveučilisteu ZagrebuFakultetelektrotehnikeiračunarstva, UNIZG-FER, Croatia
AIT Austrian Institute of Technology GmbH, AIT, Austria
NextworksSrl, NXW, Italy
ConsorzioNazionale 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
InstytutChemiiBioorganicznejPolskiejAkademiiNauk, PSNC, Poland

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): Pavle Skočir

E-mail: pavle.skocir@fer.hr

Author(s): Pavle Skočir (UniZG-FER), Mario Kušek (UniZG-FER), Vasilis Glykantzis (ICOM), Joao Garcia (UW), Gerhard Dünnebeil (AIT), Matteo Pardi (NXW), Tilemachos Pechlivanoglou (ICOM), Elena Garrido Ostermann (ATOS), Petar Krivić (UniZG-FER), Konstantinos Katsaros (ICOM), Ivana Podnar Žarko (UniZG-FER)

Doc ID: D2.3-v2.4

Amendment History

Version	Date	Author	Description/Comments
v0.1	5/12/2016	Pavle Skočir (UniZG-FER)	ToC
v0.2	19/12/2016	Mario Kušek (UniZG-FER), Vasileios Glykantzis (ICOM)	Initial responsibilities put into document, added content into Section 4
v0.3	5/1/2017	Pavle Skočir	Update of Sections 2, 3
v0.4	12/1/2017	Joao Garcia (UW), Gerhard Dünnebeil (AIT), Matteo Pardi (NXW)	Added content into Sections 6 and 7
v0.5	20/1/2017	Pavle Skočir, Vasileios Glykantzis, Tilemachos Pechlivanoglou (ICOM), Elena Garrido Ostermann (ATOS), Matteo Pardi, Petar Krivić (UniZG-FER), Mario Kušek	Update of Sections 2, 3, added content into Sections 5.1, 5.2
v1.0	23/1/2017	Pavle Skočir	Update of all sections
v1.1	26/1/2017	Joao Garcia, Matteo Pardi	Update of Sections 6, 7
v2.0	26/1/2017	Konstantinos Katsaros, Elena Garrido Ostermann, Mario Kušek,PavleSkočir	Update of all Sections after the internal review
v2.1	27/1/2017	Mario Kušek, Ivana Podnar Žarko (UniZG-FER)	Update of Sections 5, 6; Executive summary and Section 2
v2.2	31/1/2017	Konstantinos Katsaros, Elena Garrido Ostermann, Mario Kušek	Minor updates in sections: 1, 2, 3, 5, 6
v2.3	31/1/2017	Konstantinos Katsaros, Mario Kušek	Minor updates in section 3
v2.4	6/2/2017	Sergios Soursos	Submission ready version

Legal Notices
The information in this document is subject to change without notice.
The Members of the symbIoTe Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the symbIoTe Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Table of Contents

1 Executive Summary

The aim of Deliverable 2.3, entitled “Report on symbIoTe Domain-Specific Enablers and Tools”, is to document the generic architecture of symbIoTe Enablers. Enablers are envisionedas domain-specific back-end services,which are placed within the symbIoTe Application Domain (APP).They are designed to ease the process ofcross-platform and domain-specific application development, and even cross-domain application development (specifically for mobile and web applications).

The deliverable briefly presents the symbIoTe ecosystem and its layered architecture to explain how Domain-Specific Enablers fit into the symbIoTe ecosystem. Since symbIoTe applications access directly to resources from IoT platforms, applications need to be interacting with multiple platforms and perform data processing and aggregation on the application side. This may not be favorable for some domains where applications require e.g. higher-level aggregated and processed data, and would like to be freed of the burden of complex processing tasks.Enablers are envisioned as entities providing such domain-specific functionality to facilitate development of cross-platform and cross-domain applications. They can conceal the complexity of the symbIoTe system from applications and be used as a single access point to symbIoTe environment.

For example,an enabler for “air quality monitoring” could collect air quality readings fromappropriate sensors being managed by different platforms within the samecity, perform certain processing techniques so as to analyzethe collected data and provide the output in an as-a-Servicemanner to applications. This way, the application does not need to interact with multiple platforms and does not need to have domain-specific knowledge to process air quality data. In addition, cross-domain applications using multiple enablers can leverage and combine services offered by different domainenablers.Thus, application developers can easily create innovativeapplications by focusing only on the cross-domain logic,without having to care for the domain-specific details or direct interactions with multiple IoT platforms.

Enablers have a dual role in the symbIoTe architecture:1) they act as applications, since they connect to the CoreServices, search for resources and then access directly theinvolved IoT platforms; 2) they also appear as platformsfor third-party applications: applications use the CoreServices to search for enablers, and after getting appropriately authorized, can access directly the enabler services.

The document presents the System Requirements based on which Enablers are specified and their generic architecture. Enablers’architecture is composed of 1) generic components for communication with symbIoTe Core where it reuses the symbIoTe Cloud components which extend IoT platforms, and 2) domain-specific components providing added value on top of IoT platform resources to provide domain-specific services.This domain-specificfunctionality can be considered as the backend service of adomain-specific service: it performs all the necessary datatransformations and processing to create and offer therespective domain knowledge to third-party applications.

Within the symbIoTe project, Enablers will be designed in domains covered by symbIoTe-defined use cases to facilitate the development of use-case specific applications. We present a selected use case (Smart Mobility and Ecological Routing)and the corresponding Enabler in this deliverable, along with the corresponding functionalities planned to be implemented within an Enabler. Finally, we review the technologies that are planned to be used for Enabler implementation.

2 Introduction

In light of a highly fragmented IoT ecosystem faced with an increasing number of IoT platforms, their interoperability and collaboration is quite challenging to achieve. However, platform interoperability is an indispensable prerequisite for the emergence of novel cross-platform IoT applications and business opportunities.The symbIoTe project steps into this landscape by providing the means to create and manage virtual IoT environments across variousIoT platforms. Domain-specific Enablers, pointed out in Figure 1, leverage such virtual environments to offer specialized, value-added services. They can be regarded as "domain-specific virtual IoT platforms" since enablers do notmanage and offer services on top of actual physical resources but rather provide domain-specific functionality and services by abstracting and aggregatingresources belonging to different IoT platforms.

For example,an enabler on "air quality monitoring" could collect data fromappropriate sensors from different platforms within the samecity, perform certain processing techniques so as analyzethe collected data and provide the output in an as-a-Servicemanner to applications. This way, cross-domain applications can leverage and combine higher-level services offered by different domainenablers, while application developers can easily create innovativeapplications by focusing only on the cross-domain logic,without having to care for the domain-specific details.

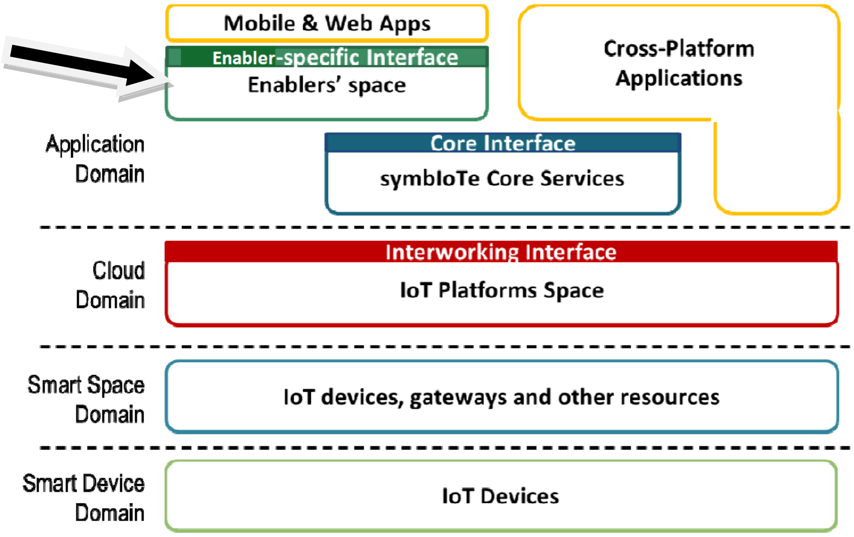

Figure 1 symbIoTe Architecture

Let us briefly reviewthe layered symbIoTe architecture to analyzethe placement of Enablers within this architecture. IoT platforms connect various devices (sensors, actuators, IoT gateways) within Smart Spaces with the Cloud. Nevertheless, the sharing of sensor data and interaction between devices from various platforms is difficult to achieve, even if devices aresituated in the same space and belongto the same domain (e.g., healthcarewearablesor home automation devices within the same apartment where their interaction could be used for development of new services). symbIoTe adds an open interface over such platforms (Interworking Interface shown in Figure 1), enabling different IoT platforms to decide which resources will be exposed and advertised to third parties through symbIoTe Core Services.This open interfacefacilitates application developers to access resources from different platforms in the same way to create cross-platform applications.

The symbIoTe architecture, shown in Figure 1, is built around the aforementioned layered IoT stack connecting various devices within Smart Spaces with the Cloud. The architecture comprises four layered domains: 1) Application Domain, 2) Cloud Domain, 3) Smart Space Domain, and 4) Smart Device Domain.

1. Application Domain (APP):enables platforms to register resources they want to advertise and make accessible via symbIoTe to third parties. symbIoTe provides Core Services as the means for discovery of those resources across platforms. Domain-Specific Enablers, the focus of this Deliverable, also reside within APP. Enablers are specific back-end services designed to ease the process of cross-platform and domain-specific application development.
2. Cloud Domain (CLD): provides a uniform and authenticated access to virtualized IoT devices exposed by platforms to third parties through an open Interworking Interface. In addition, it builds services for IoT Platform Federations enabling close platform collaboration, in accordance with platform-specific business rules.
3. Smart Space Domain (SSP): provides services for discovery and registration of new IoT devices in dynamic local Smart Spaces, dynamic configuration of devices in accordance with predefined policies in those environments, and the ability to connect user applications with Smart Space.These functionalities are a prerequisite for Smart Space-specific services (e.g. switching the lights on or off in the room user is currently in by using a smartphone application).
4. Smart Device Domain (SDEV): relates to smart devices and their roaming capabilities. We assume that devices have the capabilities to blend with a surrounding Smart Space while they are on the move. In other words, smart devices can interact with devices in a visited smart space, which are managed by a visited platform, in accordance with predefined access policies (e.g. an IoT device on yacht entering a marina can interact with the marina administration to find a desirable mooring).

symbIoTe allows for flexible interoperability mechanisms which can be achieved by introducing an incremental deployment of symbIoTe functionality across the listed architectural domains (APP, CLD, SSP and SDEV). Four Compliance Levels for IoT platforms are defined (Level1 – Level4):

- Level1 symbIoTe Compliant Platform indicates opening up platform interface to third parties in order to advertise and offer its resources through symbIoTe Core Services.
- Level2 symbIoTe Compliant Platform assumes that platforms federate, thus adding additional functionalities within their CLD.
- Level3 symbIoTe Compliant Platform assumes that platforms can share their IoT devices situated within the same space, i.e. within the symbIoTe-defined Smart Space.
- Level4 symbIoTe Compliant Platform offers support for device roaming, i.e. enabling interaction of smart objects within visited Smart Spaces.

Within this Deliverable, the focus is on Application Domain (APP) where Enablers are situated, and on the interaction of Enablers with symbIoTe Core components. Only symbIoTe Level 1 Compliance is considered.

2.1 Purpose of this document

This deliverable reports the features and design of Domain-Specific Enablers identified and specified in T1.3, and paves the way for simplified development of cross-platform IoT applications. It identifies the necessary technologies that can further simplify IoT application design and implementation.

2.2 Relation to other deliverables

System Requirements reported in this document based on which Enablers are defined are primarily derived from deliverable D1.2 “Initial Report on System Requirements and Architecture”. The progress in the specification of System Requirements, made after the submission of D1.2, is also captured. The generic Architecture for symbIoTe Enablers is built upon the symbIoTe Core Services presented in the D1.2 since symbIoTe Enablers reuse components from the symbIoTe Core Services, presented in deliverable D2.2 “symbIoTe Virtual IoT Environment Implementation.” symbIoTe will design and implement Domain-Specific Enablers 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.” The technologies planned to be used for the implementation of symbIoTe system are specified in D5.1: “Implementation Framework,” and will be considered in this document to be used for Enabler implementation.

This deliverable presents an initial specification related to symbIoTe Domain-Specific Enablers.The final report containing Domain-Specific features and implementation details will be reported in D2.6 “symbIoTe Domain-Specific Enablers and Tools.”

2.3 Document structure

Section 3 describes Enablers in more detail, introducing their main features and the benefits of their usage in an IoT ecosystem. System requirements relating to enablers are presented in Section 4. Even though symbIoTe Enablers are planned to be domain-specific, generic components existfor each of theseEnablers, which are proposed in Section 5. A Domain-Specific Enabler for aselected symbIoTe usecaseis presented in Section 6. Technologies for Enablers’ implementation are considered in Section 7.

3 Domain-Specific Enablers

In the telecommunications context, the term *enabler*isused to describe entities,which offer service interoperability across devices, geographies, service providers, operators, and networks, while allowing businesses to compete through innovation and differentiation [1]. Enablers are also used in the context of platforms providing easy to use,“menu” services to customers [2], for example, generic enablers are listed within the well-known FIWARE Catalogue[1]. In symbIoTe,we envision Enablers as domain-specific back-end services facilitating simplified development of third-party IoT applications within the symbIoTe ecosystem, thus creating an opportunity for a wider adoption of the ecosystem. Enablers,on the one hand, usethe symbIoTe system to access the actual IoT resources managed by various IoT platforms, while on the other hand add value on top of those resources to develop functionalities relevant to aspecific domain. Therefore, they are being referred to as Domain-Specific Enablers.

Let us first review how a third-party application can use resourcesoffered by symbIoTe-complaint platforms. Firstly, an application queries the symbIoTe Core Services to find the desired resources. Since symbIoTe Core Services storeonly the resource metadata, i.e. they onlystore resource description with information on how to access the resources, an application gets a response containing access points to “native” IoT platforms exposing the discovered resources. Secondly, the application sends requests to acquire sensor data or access actuator primitives directly on the IoT platform side, and handles the responses on its own.

In accordance with the previous scenario, Enablers, as shown in Figure 2,also act as third-party applications when using resources from different IoT platforms and handle all communication with symbIoTe Core Services and the underlying IoT platforms. However, they add value on top of these resourcesoffering enabler-specific services to applications.With Enabler support, an application thus gains access to processed data from various IoT platformsthrougha single interface, the Enabler-specificInterface exposed by that Enabler.If it were not for Enablers, each applicationwouldneed tohandle access to multiple IoT platforms and acquired data on its own.

[1] https://catalogue.fiware.org/enablers

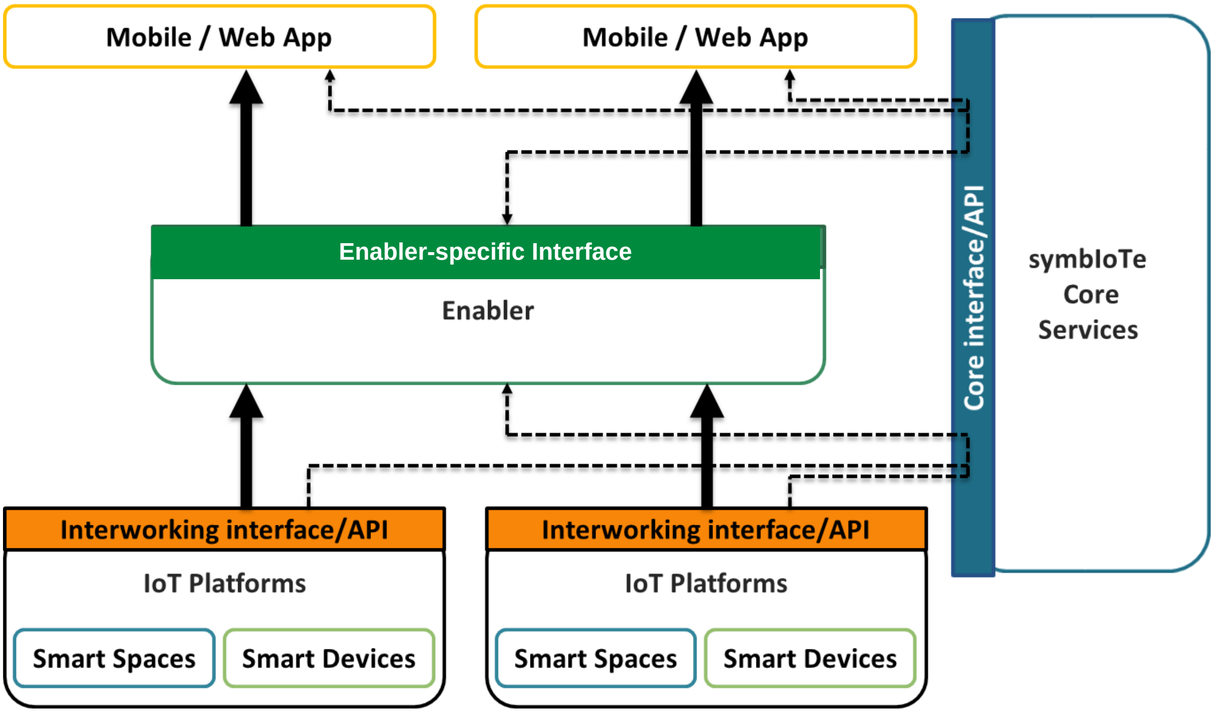

Figure 2 Cross-platform application development using Enablers

In addition to acting as third-party applications, Enablers can store data and provide data analytics on top of the data set acquired from various sources and administrative domains. They offer value-added services on top of IoT services, which are managed and offered by “native” IoT platforms. In principle, an Enabler is a software component that can be regarded as a virtual IoT platform since it does not possess the actual hardware, but rather offers value-added services on top of the IoT services and devices being accessed through the Interworking Interface. For symbIoTe Core Services an enabler thus plays a dual role: 1) it is an application that uses symbIoTe Core Services to find adequate IoT resources, and 2) it acts as another IoT platform offering domain-specific IoT resources to applications.

In their role as applications, they access resources from actual IoT platforms and transform, combine and store them (e.g. air sensor data from European capitals). In their role as platforms, Enablers provide virtual resources to third-party applications from initial data (e.g. air quality indexes based on for example wind, temperature and traffic data). Additionally, as platforms, Enablers also register their services with the symbIoTe Core Services, so that they can be discovered by other applications wanting to make use of them to find and access data from underlying IoT platforms. The search functionality is depicted by dashed arrows in Figure 2.

Applications can also use different Domain-Specific Enablers to create cross-domain applications. This way, a wider range of value-added services can be developed. Without Enablers, applications would need to send multiple queries to symbIoTe Core Services to identifyneeded resources in different domains, and then would need to handle access to those resources residing in multiple IoT platforms.

The main features of the Enablers are the following:

- They facilitate the usage of symbIoTe system to third-party applications.

When Enablers are available to symbIoTe applications, applications can simply interface them, specifying the resources they would like to access. An Enabler returns the corresponding data. This is enabled by internal Enabler processes and components, which are responsible for searching and accessing the underlying IoT platform resources necessary for the application. In this process, an enabler can firstly contact symbIoTe Core to find specified resources, then access the discovered resources through the Interworking Interface of one or more IoT platforms, process the retrieved data and store it locally. Enablers thus offer the possibility to use symbIoTe Core Services and IoT platforms through only one interface, Enabler-specific Interface. Application developers do not even have to know the existence of the underlying IoT platforms.

It is important to emphasize that Enablers act as a single access point for applications during their execution time. In order to find the Enabler (if the location and functionalities of the Enabler are not known beforehand), an application developer can execute a query by using symbIoTe Core Services, in the same way as he/she would search for underlying resources from "native" IoT platforms.

- They can store intermediate data.

  Data storage is an important functionality for use-cases where historic data needs to be taken into account, where statistical operations need to be performed over gathered data or where historic data needs to be presented in the applications. Without Enablers, applications would be responsible for storing data gathered from multiple IoT platforms. As previously commented, symbIoTe Core Services do not store values of registered resources, they only store the location where the resources can be acquired. Handling data from IoT platforms, and performing further operations on this data would cause much additional work to application developers, and by having this functionality, Enablers expedite application development.

- They offer out-of-the-box logic functionality.

  Out-of-the-box logic includes specific services, such as data aggregation, and more complex data analysis tasks (e.g. outlier detection) that can be performed within Enablers after acquiring data from the desired resources residing at multiple IoT platforms, thus relieving the application developers of the additional work of implementing such functionalities that are considered important in a certain domain.

- They enable usage of external services to create added-value IoT services

  External services are services that can originally be thought of as outside the IoT domain, but able to be used with symbIoTe-provided data. Enablers offer integration of those services with symbIoTe-provided data. Examples are emergency response service, which can be triggered based on data from health sensors, or routing service that can route the users based on data from air quality sensors. By integrating these services with symbIoTe-provided data, Enablers can create new added-value IoT services.Enabler developers should manage these external services integrated with symbIoTe.

4 System Requirements for Enablers

The following table lists the set of symbIoTe system requirements applicable in the Enablers' domain. These include requirements presented in deliverable D1.2 but also additional ones specified past the submission of this deliverable. The requirements were specified within T1.3 and act as the starting point for defining main functionalities of the Enablers and to identify the required components in Enablers' Architecture.

Table 1 System Requirements for Enablers

Index	Domain	Type	Category	Importance	Description	Use Cases
6	Application, Cloud	Functional	Monitoring	SHOULD	The system SHOULD monitor the load on the registered IoT services. Related information can be directly retrieved by IoT platforms (if supported). Additionally, the system can keep track of the IoT services assigned to applications/enablers during the mediation process e.g., when an application developer has identified, requested and has been granted access to IoT services for the intended application. The retrieved information can be used to estimate service load, service popularity (useful for ranking).	2,3,4
20	Application	Functional	Management	MUST	The system MUST offer domain-specific enablers that hide from application developers the existence of multiple IoT platforms and resources targeted to a specific domain. The system must manage all the underlying resources, include the required logic, ensure the required quality, performance, etc (see Requirements 2, 5-12)	1,2,3,4,5
21	Application	Functional	Management	SHOULD	The system SHOULD allow application developers to create their own enablers (focusing on a single domain or be cross-domain), defining their own logic, etc. These "user-owned enablers" should be available at least to their creators.	1,2,3,4,5
22	Application	Functional	Management	MAY	The system MAY allow application developers to share their custom enablers with other application developers. Trading mechanisms may be in place to govern the use of custom enablers.	1,2,3,4
29	Application, Cloud	Non-Functional	Performance	SHOULD	Number of IoT platform instances/enablers: the system SHOULD scale in the order of thousands of instances.	2,3,4
30	Application, Cloud	Non-Functional	Performance	SHOULD	Number of applications/enablers: it SHOULD scale in the order of thousands of instances.	2,3,4
34	Application, Cloud	Functional	Management	SHOULD	The system SHOULD enable IoT platforms to control whether their IoT services appear in search results, subject to the access rights of the query issued to these services i.e., whether the application developer or enabler is registered with the respective IoT platform.	1,2,3,4,5
35	Application, Cloud	Functional	Management, Interface	MAY	The system MAY enable IoT platforms to define access rules to their IoT services during the registration process. Such access	

					rules refer to the intended availability of the IoT services to applications/enablers e.g., maximum 10 times per day, only from 7p.m. to 7a.m..	
37	Application, Cloud	Functional	Management	MAY	The system MAY support the registration of applications/enablers to underlying IoT platforms. This requirement pertains to cases where the search results contain IoT services that the query issuer does not currently have access rights for. An example mechanism for the intended symbIoTe support, is the redirection to the IoT platform registration interface.	1,4
39	Application, Cloud	Functional	Management	SHOULD	The system SHOULD support registration updates i.e., IoT platform operators/enablers should be able to update their registered IoT serviceswith symbIoTe. For example, updating provided information upon sensor/actuator upgrades.	1,2,3,4,5
45	Application, Cloud	Functional	Interface	SHOULD	Enablers SHOULD be regarded as high-level IoT platforms that can register their domain-specific services to the system, similar to native IoT platforms.	1,2,3,4,5
67	Smart Space	Functional	Management / Interface	SHOULD	An app/enabler SHOULD be able to receive a notification whenever an L4-Compliant resource it is using changes Smart Space association	1,2,3
81	Application	Functional	Monitoring	MUST	The system MUST monitor the quality of the offered services so as to make sure that the advertised quality of service is met e.g., number of aggregated sensors, accuracy of reported values, etc.	1,2,3,4,5
82	Application	Functional	Management	MUST	The system MUST manage all the underlying resources so as to ensure the required quality, performance. For instance, an enabler that aggregates sensor readings throughout a country can search for new/alternative sensors in a certain area, if it experiences failures with already aggregated sensors there.	1,2,3,4,5
83	Application	Functional	Management	MUST	The system MUST present a minimum application domain logic. In the simplest case this corresponds to the mere aggregation of IoT resources from multiple IoT services. More advances processing can be applied for the support of added value services.	1,2,3,4,5

5 Generic Architecture for symbIoTe Enablers

The main purpose of Enablers is to provide functionalities needed in specific domains. Examples of such functionalities are data aggregation, forwarding data based on predefined thresholds, forwarding data in certain periods of time, analyzing data, forwarding certain outliers etc. However, each of the Domain-Specific Enablers with the aforementioned functionalities have generic functionalities that enable interaction with symbIoTe components from other domains in the symbIoTe ecosystem. This generic architecture, with the necessary components, is shown in Figure 3.

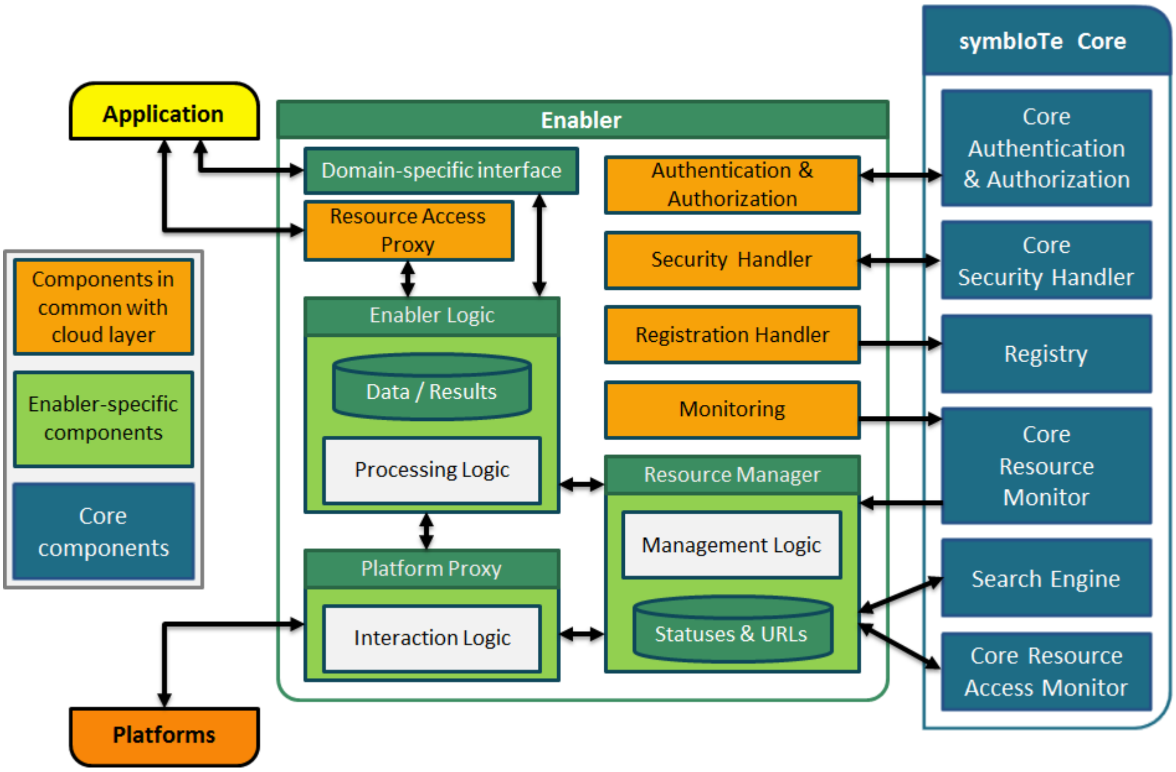

Figure 3 Enablers' Generic Architecture

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

Components in common with Cloud Domain are generic components for each Enabler. These are: Resource Access Proxy (RAP), Authentication & Authorization (AAM), Security Handler (SH), Registration Handler (RH), and Monitoring. RAP serves as access point for application to acquire resources exposed by Enabler. AAM is responsible for authenticating and authorizing applications and Enabler components. SH provides a set of libraries implementing security functionalities, RH registers resources exposed by Enabler to symbIoTe Core, while Monitoring monitors the availability and usage of the exposed resources.

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

Resource Manager is a component with the main goal tofind through symbIoTe Core the required resources that will be processed and made available by theEnabler. Domain-specific functionality of this component could be handling logic for finding Underlying resources through symbIoTe Core (e.g. when a resource is not available for a certain amount of time, a replacing resource will be searched for).

Platform proxyhas a main functionality to access Underlying resources found by Resource Manager from IoT platforms. Domain-specific functionality of this component could be interaction logic for retrieving resources from underlying IoT Platforms (e.g. every 10 seconds).

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

The following parts of this Section describe Enabler architecture in more detail. Section 5.1 presents Enabler-specific Interface exposed to applications. The Enabler components are described in Section 5.2. Sequence diagrams showing the interaction between these components, as well as interaction with components from other domains, are presented in Section 5.3.

5.1 Enabler-specific Interface

For communication with Applications, Enabler-specific Interface is specified, composed of RAP and a Domain-Specific Interface. RAP has the same functionality as the component with the same name in the CLD domain, it allows applications to access the wanted resources. When accessing Enabler resources through RAP, the communication goes through symbIoTe Core (as in L1 Compliance). In that way, symbIoTe Core is informed of the usage of Enabler resources. Domain-specific Interface enables additional domain-dependent functionalities, mainly for accessing Enabler Logic and data store. When using Domain-specific Interface, communication between application and Enabler is direct.Enabler-specificInterface should be exposed to application developers so that they can use it within their applications. Accessing Enablers in these two ways, by using RAP and by using Domain-specific Interface will be presented in Sequence diagrams in Sections 5.3.9, 5.3.10, 5.3.11, and 5.3.12.

5.2 Enablers’ Components

Components in common with Cloud Domain are presented first. These are:

- Registration Handler (RH),
- Authentication and Authorization Manager (AAM),
- Resource Access Proxy (RAP),

- Monitoring, and
- Security Handler (SH).

Afterwards, Enabler-specific components are described:

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

Table 2 Registration Handler

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 IoT resources published by Enablers with other application developers by registering them at symbIoTe Core. Trading mechanisms may be in place to govern the use of the Enablers. Furthermore, the component enables registration updates.
Provided functionalities	• Registers resources to the symbIoTe Core using the symbIoTe Core Information Model • Updates resource status and unregisters resources • Registers pricing of a resource • Registers security info • Handles configuration of the exposed resource • Synchronizes the information with symbIoTe Core
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 on the Resource Access Proxy, along with the access policy to access it
Related use cases	ALL
Related requirements	22, 34, 39, 45

Table 3 Authentication and Authorization Manager

Component	**Authentication and Authorization Manager**
Description	This component provides similar functionalities to the platform-side Authentication and Authorization Manager component (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 tokens revocation in place of other 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 tokens revocation in place of other enabler's components (e.g., Registration Handler, Resource Access Proxy) Platform Authentication & Authorization Manager (CLD): To check home tokens revocation in place of other enabler's components (e.g., Registration Handler, Resource Access Proxy)
Related use cases	ALL
Related requirements	35, 37

Table 4 Resource Access Proxy

Component	**Resource Access Proxy**
Description	This component provides similar functionalities to the platform-side Resource Access Proxy component (CLD). The component acts as a mediator between Enabler Logic and the application. The presence of this component is necessary to make the Enabler symbIoTe-compliant, by allowing the applications to access the resources of the Enablers and platforms uniformly.
Provided functionalities	• Access to the resource or service exposed by the Enabler • Registers when an application starts or stops using resources
Relation to other components	symbIoTe-enabled Application: It accesses Resource Access Proxy to acquire resources Registration Handler (Enabler): Informs Resource Access Proxy of the registered resources that it should provide access to Security Handler (Enabler): Verifies tokens Monitoring (Enabler): Resource Access Proxy notifies Monitoring when applications start or stop using resources Core Resource Monitor: Contacts Resource Access Proxy to check the availability/status of resources

	Core Resource Access Monitor: Resource Access Proxy emits resource usage and notifies the Core Access Resource Monitor when the resource is released
Related use cases	ALL
Related requirements	

Table 5 Monitoring

Component	**Monitoring**
Description	Similar to the platform-side Monitoring component (Cloud Domain). It monitors the load on the IoT services offered by Enabler and the usage of the registered services by the applications. The retrieved 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 e.g., number of aggregated sensors, accuracy of reported values, etc.
Provided functionalities	• Checks load/availability of the resources registered by Enabler • Record of start and end of the access to a resource
Relation to other components	Security Handler (Enabler): To request core token Resource Manager (Enabler): To notify about the status/availability/performance of the resources Core Resource Access Monitor: To send usage report
Related use cases	ALL
Related requirements	6, 81

Table 6 Security Handler

Component	**Security Handler**
Description	This component combines functionality from the platform-sideSecurity Handler(CLD) and the Application Security Handler (APP).
Provided functionalities	• Authenticates with the core AAM or foreign AAM on behalf of the entity that uses its functionalities. • Manages core tokens and foreign tokens assigned to the entity that uses its functionalities. • Performs the “validate access tokens” procedure when one or more core tokens are provided to the entity that uses its functionalities. • Performs the “check revocation procedure” with the Core AAM when one or more core tokens are provided to the entity that uses its functionalities. • Initiates the “Challenge-Response Procedure” to verify that the component or application using the core tokens is effectively the component or application for which they have been released by the Core AAM, in case one or more core tokens are provided to the entity that uses its functionalities. • Manages cryptography operations on behalf of a component using its

	functionalities, when such component or application provides a set of tokens to a component in a given IoT platform federated with symbIoTe. • Performs the “Check Access Policy” procedure to verify that the tokens supplied by the applications satisfy the access policies of the resources when the Search Engine is used by an application. • Performs the “Validate Certificate” procedure on behalf of the component or application that uses its functionalities.
Relation to other components	Registration Handler (Enabler): The Registration Handler requests a core token or wants to validate a certificate Resource Access Proxy (Enabler): The Resource Access Proxy requests a core token or wants to validate a certificate Authentication and Authorization Manager (Enabler): To retrieve home tokens Core Authentication and Authorization Manager: To retrieve core tokens for other components (e.g. Registration Handler, Resource Access Proxy) Core Security Handler: To validate tokens
Related use cases	ALL
Related requirements	

Table 7 Resource Manager

Component	**Resource Manager**
Description	This component manages underlying IoT resources used by Enabler. It receives input from Monitoring component about the status/availability/performance of the resources offered by the Enabler. If the advertised quality of service is not met (e.g., some sensors go offline), this component is responsible for automatically discovering and registering to new resources. The component queries symbIoTe Core Search Engine to discover new resources which match certain criteria related to the Enabler's services. Furthermore, it ranks the results returned by the Search Engine according to the enabler's domain-specific requirements and registers to new resources in order to meet the advertised quality of service. Additionally, the component keeps track of the IoT resources assigned to Enablers during the mediation process e.g., when an enabler has identified, requested and has been granted access to IoT resources for the intended application. 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 newfound 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 registration to new resources • Periodically checks for more suitable resources in the symbIoTe Core Registry

Relation to other components	Enabler's Logic: The Enabler's Logic provides information about the required resources to the Resource Manager Monitoring: The monitoring notifies the Resource Manager about the status/availability/performance of the resources
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 8 Enabler Logic

Component	**Enabler Logic**
Description	This component presents a minimum application domain logic. In the simplest case this corresponds to the mere aggregation of IoT resources from multiple IoT services. More advanced processing can be applied for the support of 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. It is responsible for notifying the Resource Manager about the required resources as well as for accessing and storing the resources data. Furthermore, it groups the platform resources in virtual resources (if necessary), processes the data and offer more specialized services (e.g. weather forecast, providing historical data, statistical results, etc...). Finally, it notifies the Enabler's Registration Handler about which resources should be registered in symbIoTe Core, and initiates the update of registered resources.
Provided functionalities	• Defines the kind and amount of resources • Groups the platforms resources in virtual resources • Provides data storage • Processes the data and offers the possibility to develop specialized services
Relation to other components	Resource Manager (Enabler): The Resource Manager gets the description of the required resources Registration Handler (Enabler): The Enabler Logic specifies what kind of resources should be registered in symbIoTe Core Resource Access Proxy (IoT Platforms): Requirements by applications are forwarded through Resource Access Proxy
Related use cases	ALL
Related requirements	20, 21, 39, 45, 83

Example 2:

An Enabler claims to provide temperature sensors in all the European capitals. In the application, a user wants to access average temperature in Zagreb for each of the last three days. Enabler logic should gather data from selected temperature sensors in Zagreb (found by Resource Manager), calculate the average, and send the response to application.Table 9 Platform Proxy

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 data is initiated by Enabler Logic. Data can be acquired by using a push or pull mechanism. Gathered data is forwarded to Enabler Logic where it can be stored.
Provided functionalities	• Accesses IoT Platforms to acquire resources • Forwards acquired resources 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 contains the description of the required resources Enabler Logic: specifies when and how should resources from IoT platform be acquired
Related use cases	ALL
Related requirements	20, 21

Example 3:
An Enabler claims to provide temperature sensors in all the European capitals. In the application, a user wants to receive updated values every minute from two nearest sensors. Platform Proxy should acquire these measurements from sensors specified by Enabler Logic, and found by Resource Manager.

5.3 Sequence diagrams

The functionalities defined for symbIoTe Enablers are the following:

- Enabler registration
- Enabler resource registration
- Enabler resource unregistration
- Enabler resource update
- Enabler resource availability reporting
- Availability Reporting for Underlying resources
- Scheduled Enabler resource monitoring
- Search
- Enabler resource access by using RAP
- Enabler resource access by using Domain-specific Interface

- Using Underlying resource from an application using Enabler RAP
- Using Underlying resource from an application using Domain-specific Interface
- Enabler resource usage monitoring

Hereafter all functionalities are presented in the form of UML sequence diagrams, with detailed description of the exchanged messages. Figure 4 shows the legend for messages used in the diagrams. The arrow showing a “mandatory interaction”is in description of diagrams referred to as “Procedure”because it presents a sequence of messages being exchanged between components. These “Procedures” are mainly related to security mechanisms.

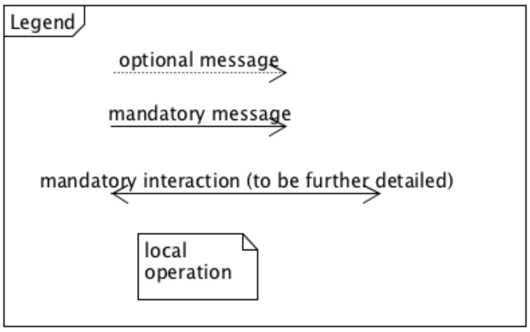

Figure 4Legend – messages used in the following diagrams

Components in the sequence diagrams are color-coded so that each color signifies a block in symbIoTe architecture:

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

5.3.1 Enabler registration

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

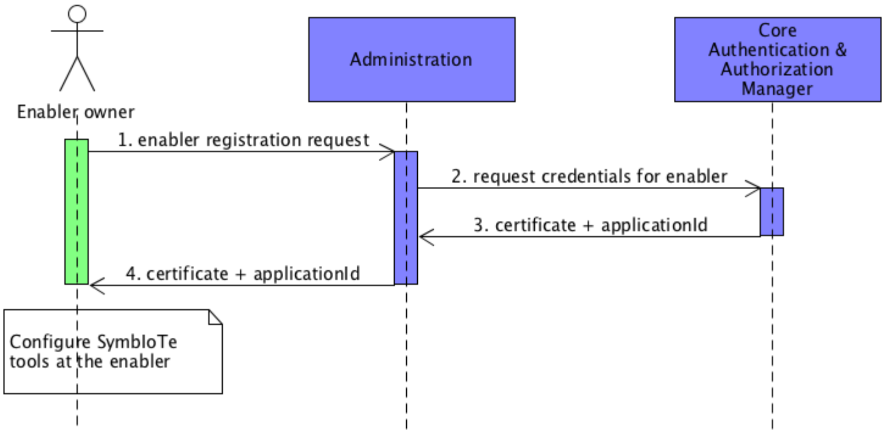
Enabler owner
Administration
Core Authentication & Authorization Manager
1. enabler registration request
2. request credentials for enabler
3. certificate + applicationId
4. certificate + applicationId
Configure SymbIoTe tools at the enabler

Figure 5 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 applicationIdto Enabler owner. The Enabler owner can subsequently configure the Enabler to become L1 Compliant.

5.3.2 Enabler resource registration

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

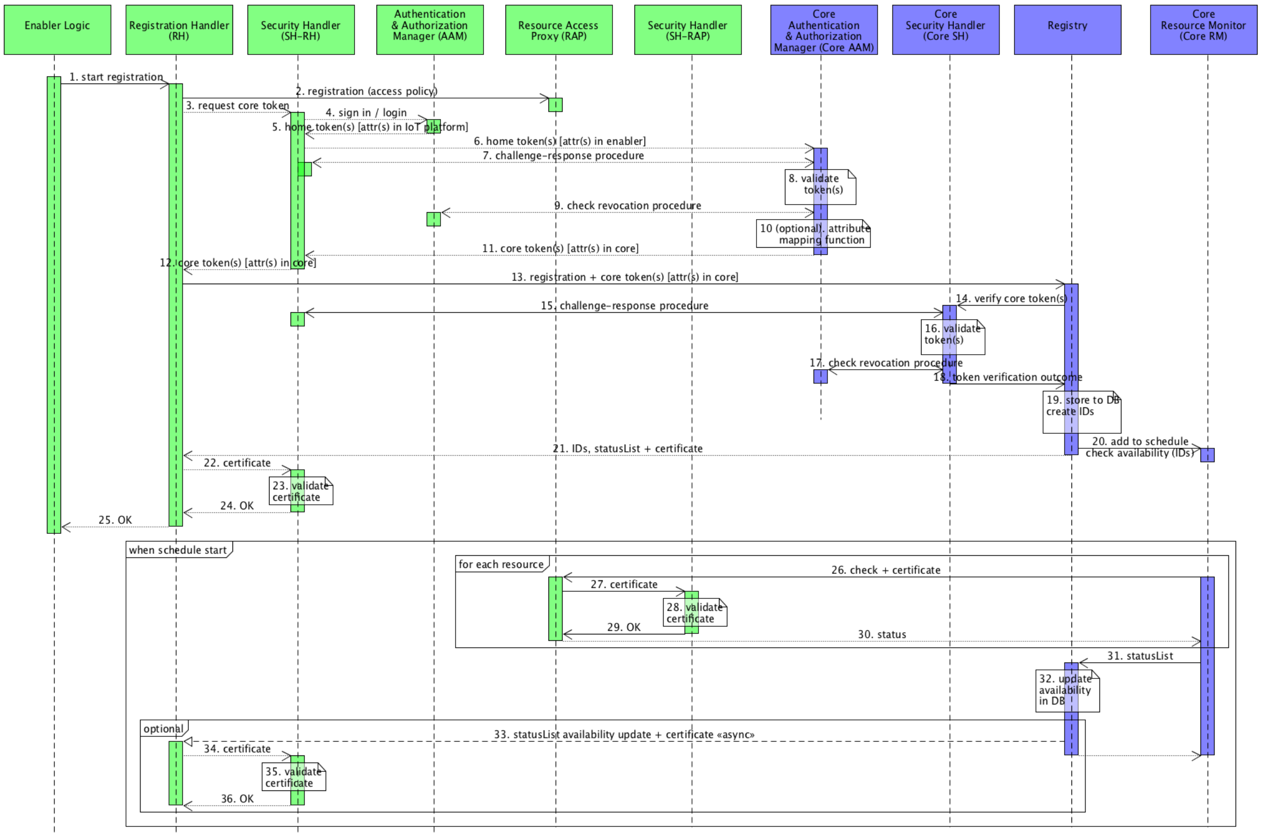
Enabler Logic
Registration Handler (RH)
Security Handler (SH-RH)
Authentication & Authorization Manager (AAM)
Resource Access Proxy (RAP)
Security Handler (SH-RAP)
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Registry
Core Resource Monitor (Core RM)
1. start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

Figure 6 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 the resource on the Resource Access Proxy, along with the access policy to access it;
- 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 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) (PlatformAAInterface): generated by the Security Handler and sent to the Core AAM in the core layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.

- Procedure 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 8 (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 9 (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 10 (optional): procedure that, in case it is needed, translates attributes that the Registration Handler has in the home Enabler in a new set of attributes that it has in the core layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 11 (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 12 (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 13 (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.
- Message 14: 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 15 (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 16: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 17: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 18: generated by the Security Handler in the core layer and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 19: stores registrations to database and generates ID for that resource
- Message 20: Registry sends a message to Core Resource Monitor to add a schedule task for checking availability of registered resources. Core Resource

Monitor will in the future check availability (messages 26-30) and asynchronously inform Resource Handler about availability with updated status list (message 33).

- Message 21: 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 22: Registration Handler forwards the certificate to Security Handler for validation
- Procedure 23: Security Handler validates the certificate
- Message 24: Security Handler returns status of validation
- Message 25: Registration Handler forwards status of validation to Enabler Logic.
- Messages 26-30 is checking of availability of each resource
- Message 26 (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 27: Resource Access Proxy sends the certificate to Security Handler for validation
- Procedure 28: Security Handler validates the certificate
- Message 29: Security Handler returns status of validation
- Message 30: Resource Access Proxy returns availability status
- Message 31: Core Resource Monitor collects all availability status, makes a status list and sends it to Registry
- Procedure 32: Updates availability in database
- Message 33 (optional): Registry sends asynchronous message with availability list and the certificate to Registration Handler
- Message 34 (triggered by 33) (RegistrationHandlerInterface): Registration Handler forwards the certificate to Security Handler for validation
- Procedure 35: Security Handler validates certificate
- Message 36: Security Handler returns status of validation

5.3.3 Enabler resource unregistration

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

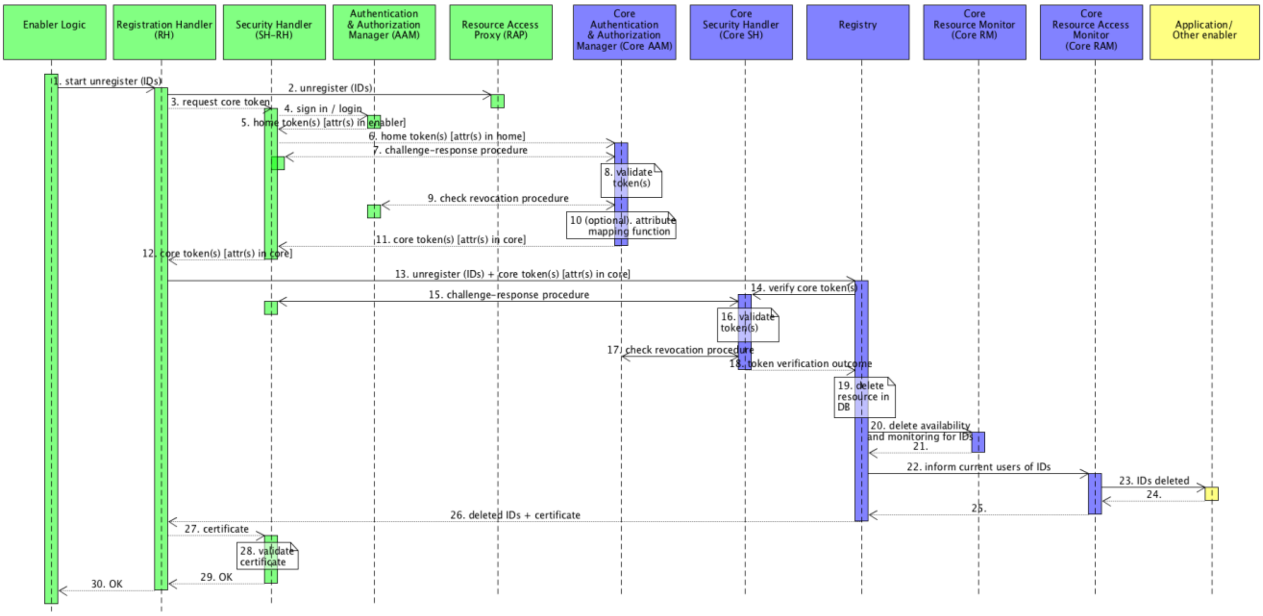
Enabler Logic
Registration Handler (RH)
Security Handler (SH-RH)
Authentication & Authorization Manager (AAM)
Resource Access Proxy (RAP)
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Registry
Core Resource Monitor (Core RM)
Core Resource Access Monitor (Core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

Figure 7 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 layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 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 8 (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 9 (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 10 (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 layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 11(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 12 (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 13 (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 14: generated by the Registry and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 15 (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 16: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 17: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 18: 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 19: Registry deletes resource in database
- Message 20: Registry sends message to Core Resource Monitor to delete availability and cancel scheduled monitoring tasks
- Message 21: returns call
- Message 22: Registry informs Core Resource Access Monitor that specific source is unregistered and that the users of that resource need to be informed
- Message 23 (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 24 returns call
- Message 25 returns call
- Message 26: 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 27: Registration Handler forwards certificate to Security Handler for validation
- Procedure 28: Security Handler validates certificate
- Message 29: Security Handler returns status of validation
- Message 30: Registration Handler forwards status of validation to Enabler Logic.

5.3.4 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.

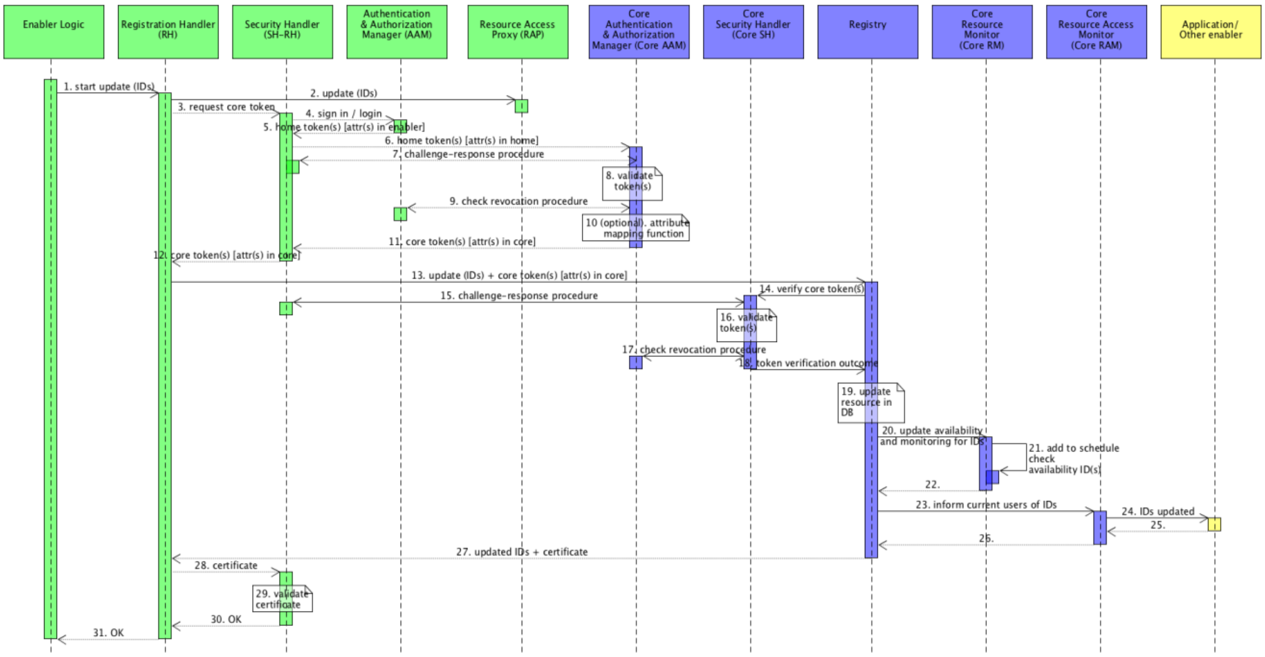
Enabler Logic
Registration Handler (RH)
Security Handler (SH-RH)
Authentication & Authorization Manager (AAM)
Resource Access Proxy (RAP)
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Registry
Core Resource Monitor (Core RM)
Core Resource Access Monitor (Core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

Figure 8 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 layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- 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 8 (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 9 (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 10 (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 layer. If attributes are the same or the Registration Handler already has valid core token(s), it is not necessary.
- Message 11 (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 12 (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 13 (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 14: generated by the Registry and sent to the Security Handler in the core layer. It is used to ask to the security handler to verify the complete validity of the token.
- Message 15 (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 16: verification of the time validity, authenticity and integrity of the provided token(s).

- Procedure 17: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 18: generated by the Security Handler in the core layer and sent to the Registry. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Procedure 19: Registry updates resource in database
- Message 20: Registry ends request to Core Resource Monitor to update availability and to schedule availability check
- Message 21: Core Resource Monitor schedules task for checking availability for specified resources
- Message 22: returns call
- Message 23: Registry sends message to Core Access Resource Access Monitor to inform current user of updated resources
- Message 24 (optional) (ApplicationInterface): Core Resource Access Monitor informs each reachable (open connection or registered endpoint) Application/Enabler that uses specific resource about resource update
- Message 25 returns call
- Message 26 returns call
- Message 27: Registry returns updated IDs including a certificate
- Message 28: Registration Handler forwards certificate to Security Handler for validation
- Procedure 29: Security Handler validates certificate
- Message 30: Security Handler returns status of validation
- Message 31: Registration Handler forwards status of validation to Enabler Logic.

5.3.5 Enabler resource availability reporting

Enabler registers Enabler resources to Core Resource Monitor. Availability check will be initiated by Core Resource Monitor (described in 5.3.7Scheduled Enabler resource monitoring).

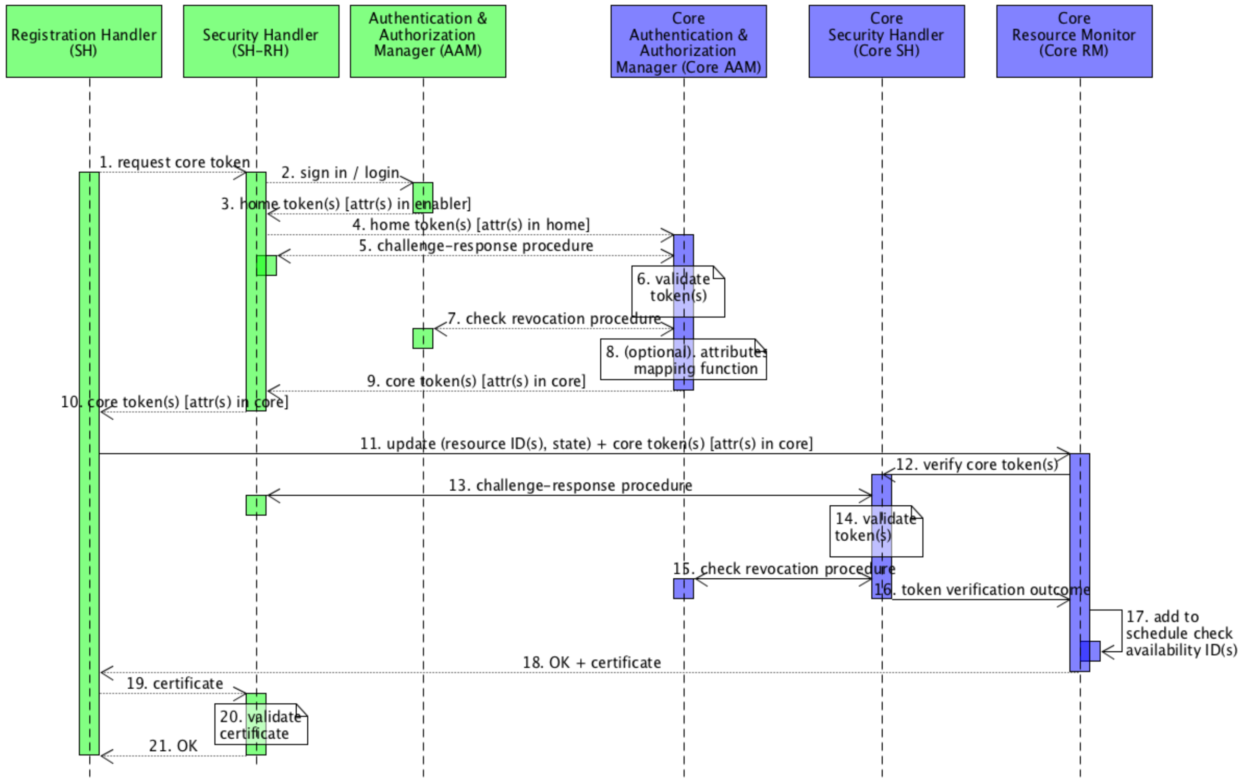
Registration Handler (SH)
Security Handler (SH-RH)
Authentication & Authorization Manager (AAM)
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Core Resource Monitor (Core RM)
1. request core token
2. sign in / login
3. home token(s) [attr(s) in enabler]
4. home token(s) [attr(s) in home]
5. challenge-response procedure
6. validate token(s)
7. check revocation procedure
8. (optional). attributes mapping function
9. core token(s) [attr(s) in core]
10. core token(s) [attr(s) in core]
11. update (resource ID(s), state) + core token(s) [attr(s) in core]
12. verify core token(s)
13. challenge-response procedure
14. validate token(s)
15. check revocation procedure
16. token verification outcome
17. add to schedule check availability ID(s)
18. OK + certificate
19. certificate
20. validate certificate
21. OK

Figure 9 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 layer. It is used to trigger the operations for obtaining the core token(s). If the Registration Handler already has valid core token(s), it is not necessary.
- Message 5 (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 6 (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 7 (optional) (AAInterface): verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Registration Handler already has valid core token(s), it is not necessary.
- Procedure 8 (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 layer. 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 (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 11: 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 12 (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 13: 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 Core Resource Monitor. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Messages 16: returns core token to Registration Handler
- Message 17: schedules task for checking availability of specified resources (IDs)
- Message 18: 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 19: Registration Handler sends certificate to Security Handler for validation
- Procedure 20: validate certificate
- Message 21: returns result of certificate validation

5.3.6 Availability reporting for Underlying resources

Resource Manager reports to Core RM about the availability of resources it uses for its operations.

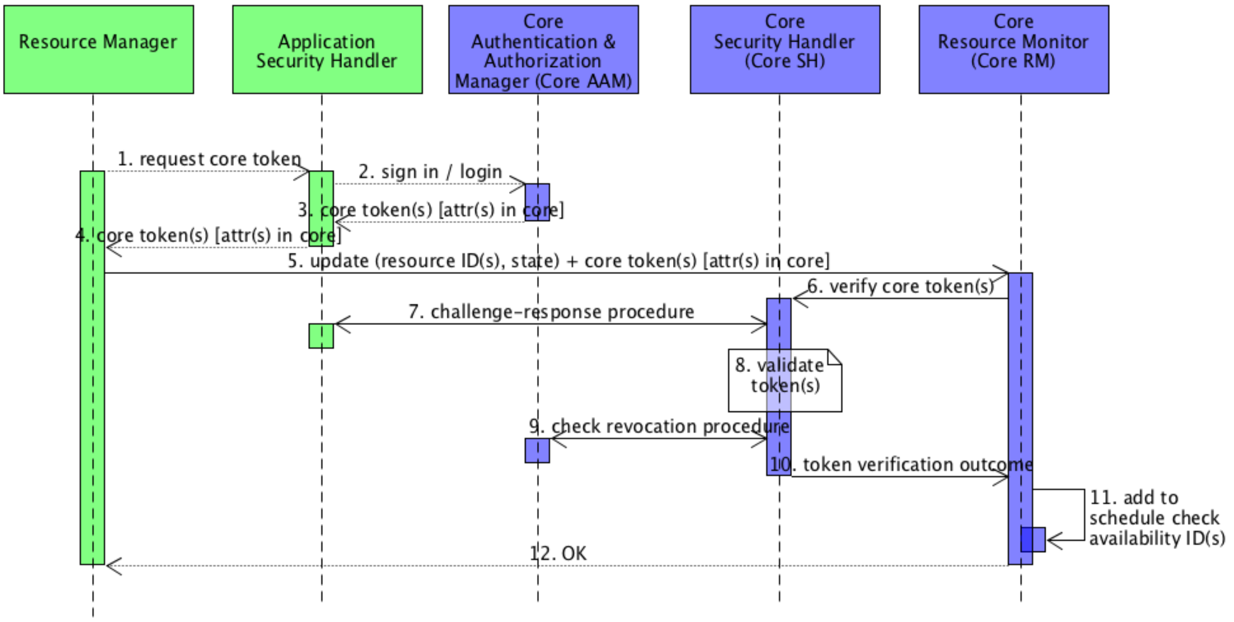
Resource Manager
Application Security Handler
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Core Resource Monitor (Core RM)
1. request core token
2. sign in / login
3. core token(s) [attr(s) in core]
4. core token(s) [attr(s) in core]
5. update (resource ID(s), state) + core token(s) [attr(s) in core]
6. verify core token(s)
7. challenge-response procedure
8. validate token(s)
9. check revocation procedure
10. token verification outcome
11. add to schedule check availability ID(s)
12. OK

Figure 10 Availability reporting for Underlying resources

Description:

- Message 1 (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 2 (optional) (EnablerAAInterface): generated by the Enabler Security Handler and sent to the Core (home) 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 3 (optional): generated by the Core (home) AAM in the Enabler 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 4 (optional): generated by the Enabler Security Handler and sent to the Resource Manager. It it is used to deliver the core token(s).
- Message 5 (ResAvailabilityInterface): generated by the Resource Manager and sent to the Core Resource Monitor. It is used to forward the update message and the core token(s) to the Core Resource Monitor.
- Message 6: generated by the Core Resource Monitor and sent to the Core Security Handler in the core layer. It is used to ask to the security handler to verify the complete validity of the token.

- Message 7 (EnablerSecurityInterface): procedure that allows the Enabler Security Handler that is acting on behalf of the Enabler to demonstrate that it is the real owner of the token(s).
- Procedure 8: verification of the time validity, authenticity and integrity of the provided token(s).
- Message 9: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 10: generated by the Core Security Handler in the core layer and sent to the Core Resource Monitor. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 11: Core Resource Monitor schedules task for checking availability of specifies IDs
- Message 12: returns status of scheduling task for checking availability

5.3.7 Scheduled Enabler resource monitoring

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

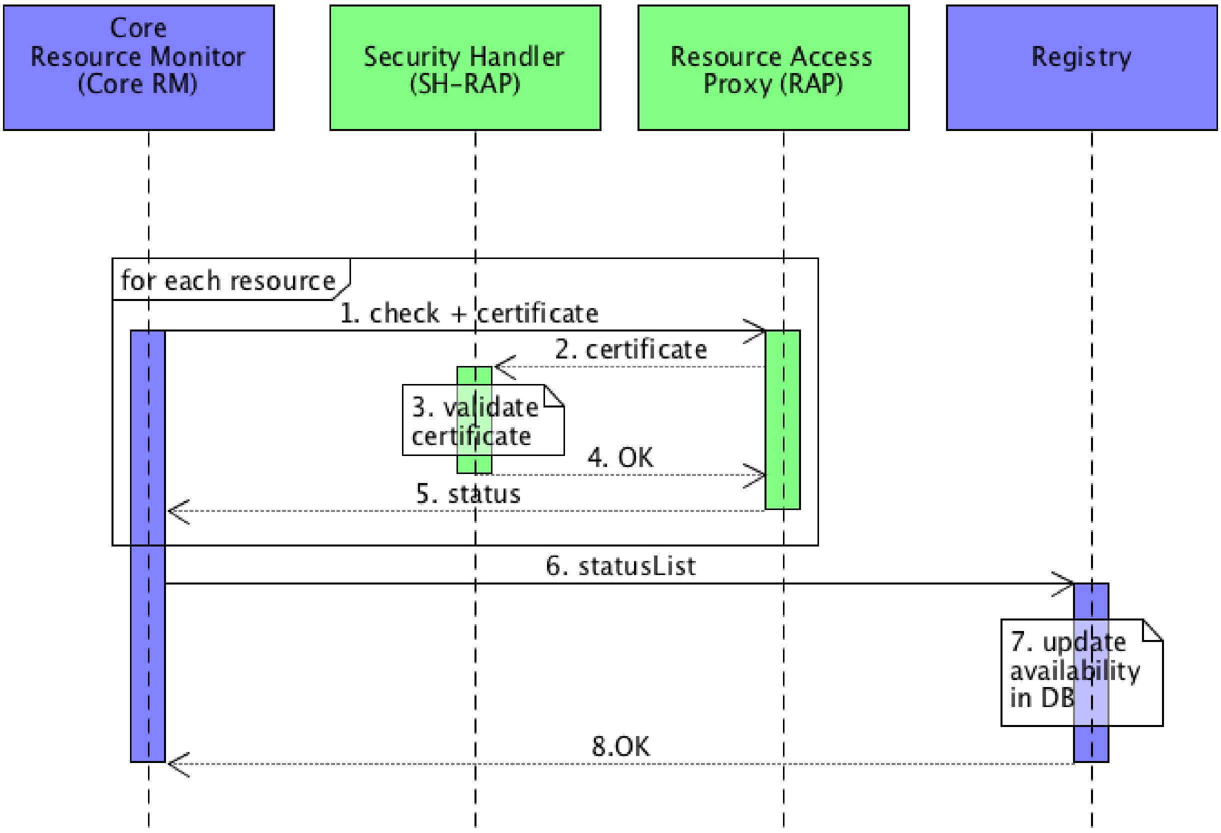

Figure 11 Scheduled Enabler resource monitoring

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 3: validates certificate
- Message 4: returns result of certificate validation
- Message 5: Resource Access Proxy returns result of availability to Core Resource Monitor
- Message 6: Core Resource Monitor collects all availability results, creates status list and send it to Registry
- Procedure 7: updates availability in database
- Message 8: returns call

5.3.8 Search

Enabler searches for resources needed for operations within Enabler logic.

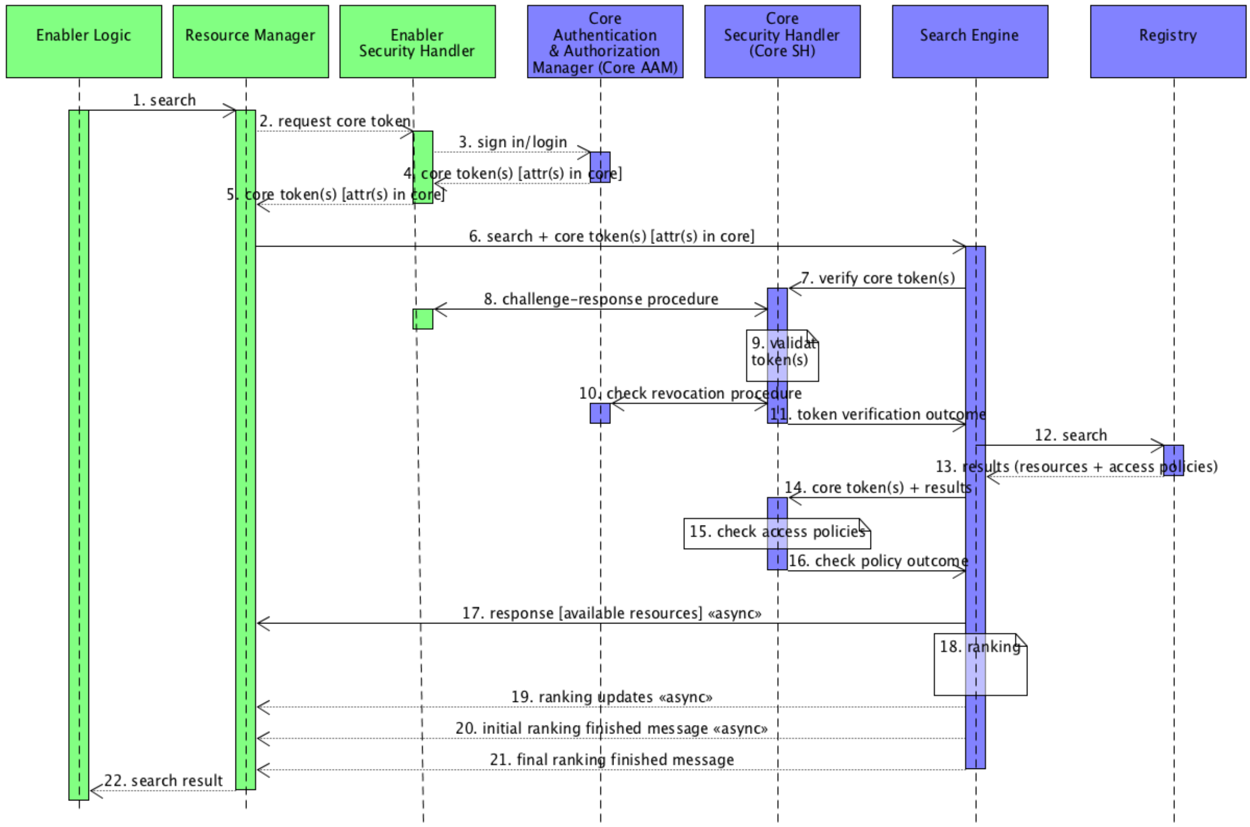
Enabler Logic
Resource Manager
Enabler Security Handler
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Search Engine
Registry
1. search
2. request core token
3. sign in/login
4. core token(s) [attr(s) in core]
5. core token(s) [attr(s) in core]
6. search + core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

Figure 12 Search

Description:

- Message 1: Enabler logic send request to Resource Manager to find resources needed for Enabler Logic
- 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 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 that is acting on behalf of the Resource Manager to demonstrate that it is the real owner of the token(s).
- Procedure 9: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 10: 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 11: generated by the Core Security Handler and sent to the Search Enginge. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 12: generated by the Search Engine and sent to the Registry. It is used to search available resources.
- Message 13: generated by the Registry and sent to the Search Engine. It is used to return the result of the search operation, containing resources and associated access policies.
- Message 14: 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 15: procedure that checks, for each resource, if the attributes contained in the core token(s) satisfy the access policy associated to that resource.
- Message 16: generated by the Core Security Handler and sent to the Search Engine. It is used to deliver the result of the previous procedure.

- Message 17: 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 18: executes ranking of food resources
- <u>Message 19 (optional)</u>: asynchronously sends ranking update to Resource Manager
- <u>Message 20 (optional)</u>: asynchronously sends message about the end of initial ranking
- <u>Message 21 (optional)</u>: synchronously sends message of the end of final ranking
- <u>Message 22 (optional)</u>: returns search results to Enabler Logic

5.3.9 Enabler resource accessby using RAP

A symbIoTe application uses resources offered by the Enabler. It accesses the Enabler through RAP. In this case, Core RAM in symbIoTe Core needs to be informed of this interaction. The application needs to be registered both with symbIoTe Core and with the Enabler.

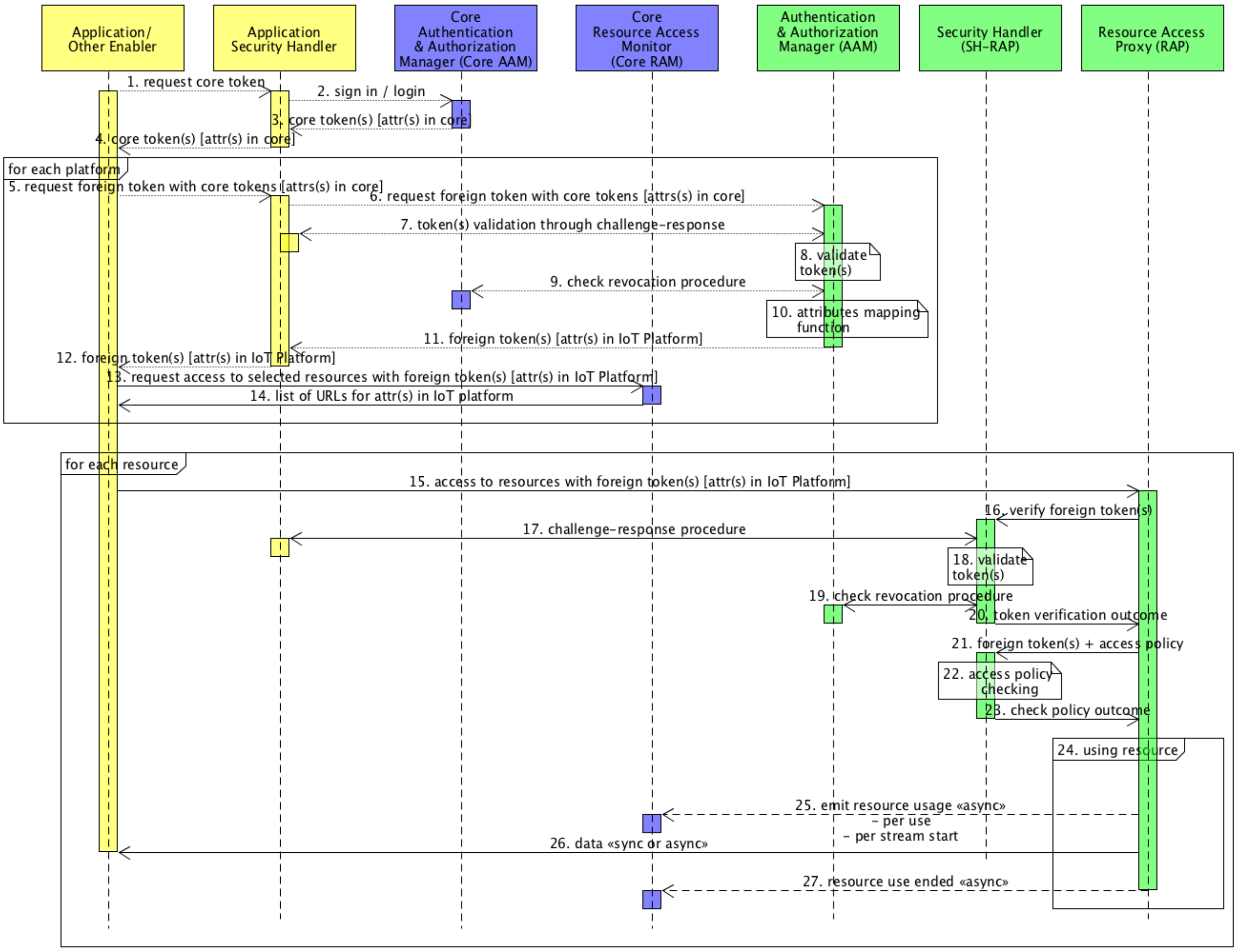
Application / Other Enabler
Application Security Handler
Core Authentication & Authorization Manager (Core AAM)
Core Resource Access Monitor (Core RAM)
Authentication & Authorization Manager (AAM)
Security Handler (SH-RAP)
Resource Access Proxy (RAP)
1. request core token
2. sign in / login
3. core token(s)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

Figure 13 Enabler resource access

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 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 8 (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 9 (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 10 (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 layer. If attributes are the same or the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Message 11 (optional): generated by the foreign AAM and sent to the Application Security Handler. It is used to deliver the foreign token(s) with the new attribute(s). If the Application/Other Enabler already has valid foreign token(s), it is not necessary.
- Message 12 (optional): generated by the Application Security Handler and sent to the Application/Other Enabler. It is used to forward the foreign token generated at the previous step.

- Message 13: Application/Other Enabler sends request access to selected resources to Core Resource Access Monitor. Message includes foreign token obtained in previous message
- Message 14: Core Recoure Access Monitor returns list of URLs for selected resources in IoT platform
- Message 15: 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 16: generated by the Resource Access Proxy and sent to the Security Handler in the foreign IoT platform. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 17: 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 18: verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 19: verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself).
- Message 20: generated by the Security Handler in the foreign IoT platform and sent to the Resource Access Proxy. It is used to communicate the outcome of the token validation procedures performed by the Foreign Security Handler.
- Message 21: generated by the Resource Access Proxy and sent to the Security Handler. It is used to deliver the core token(s) previously verified and the access policy of the requested resource to the Security Handler.
- Procedure 22: it is used to check if the attributes included in the core token(s) satisfy the access policy associated to the requested resource.
- Message 23: generated by the Security Handler and sent to the Resource Access Proxy. It is used to deliver the result of the operation executed at the previous step.
- Procedure 24: In this procedure the Enabler internally calculates results or fetch data from IoT platforms or services and generates messages 25-27. This procedure is defined in Using Resource diagram.
- Message 25: asynchronously emit resource usage per use/per stream start
- Message 26: this message can be synchronous, then Resource Access Proxy returns data. If it is asynchronously then it can emit async messages for some time
- Message 27: if previous message is asynchronous then this message informs Core Resource Access Monitor when the stream is ended

5.3.10 Enabler resource access by using Domain-specific Interface

A symbIoTe application uses resources offered by the Enabler. It accesses the Enabler through Domain-specific Interface. The application needs to be registered only with the Enabler.

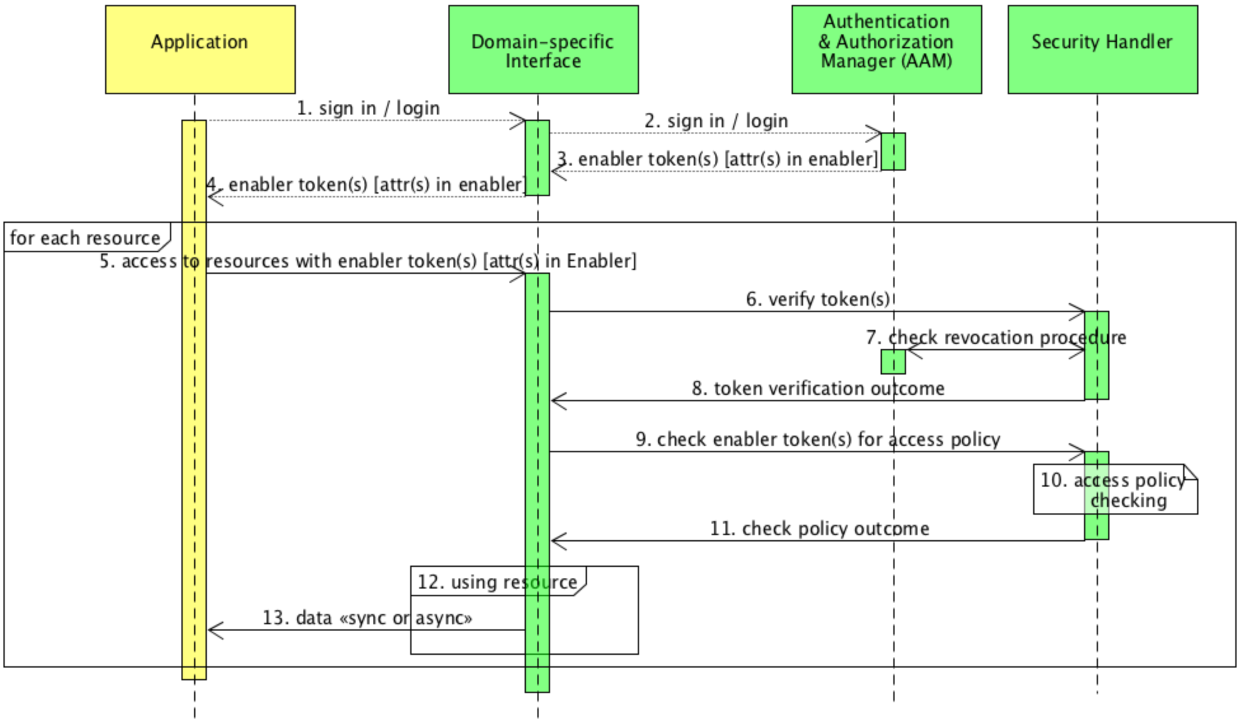
Application
Domain-specific Interface
Authentication & Authorization Manager (AAM)
Security Handler
1. sign in / login
2. sign in / login
3. enabler token(s) [attr(s) in enabler]
4. enabler token(s) [attr(s) in enabler]
for each resource
5. access to resources with enabler token(s) [attr(s) in Enabler]
6. verify token(s)
7. check revocation procedure
8. token verification outcome
9. check enabler token(s) for access policy
10. access policy checking
11. check policy outcome
12. using resource
13. data «sync or async»

Figure 14 Enabler resource access

Description:

- Message 1 (optional): generated by the application and sent to the Domain-specific Interface. It is used to get enabler token for accessing enabler. 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 enabler 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 enabler token(s).
- Message 5: generated by the application and sent to the Domain-specific Interface. It is used to access resources, while providing the enabler 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 to 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 enablers 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 enabler token(s) previously verified and the access policy of the requested resource to the Security Handler.
- Procedure 10: it is used to check if the attributes included in the enabler token(s) satisfy the access policy associated to the requested resource.
- Message 11: generated by the Security Handler and sent to the Domain-specific Interface. It is used to deliver the result of the operation executed at the previous step.
- Procedure 12: In this procedure the Enabler internally calculates results or fetch data from IoT platforms or services and generates message 13. This procedure is defined in Using Resource diagram from enabler application.
- Message 13: This message can be synchronous, then Domain-specific Interface returns data. If it is asynchronously then it can emit async messages for some time.

5.3.11 Using Underlying resource from an application using Enabler RAP

On request from an application, Enabler is accessing Underlying resources at IoT platforms. The application accesses Enabler through Enabler RAP.

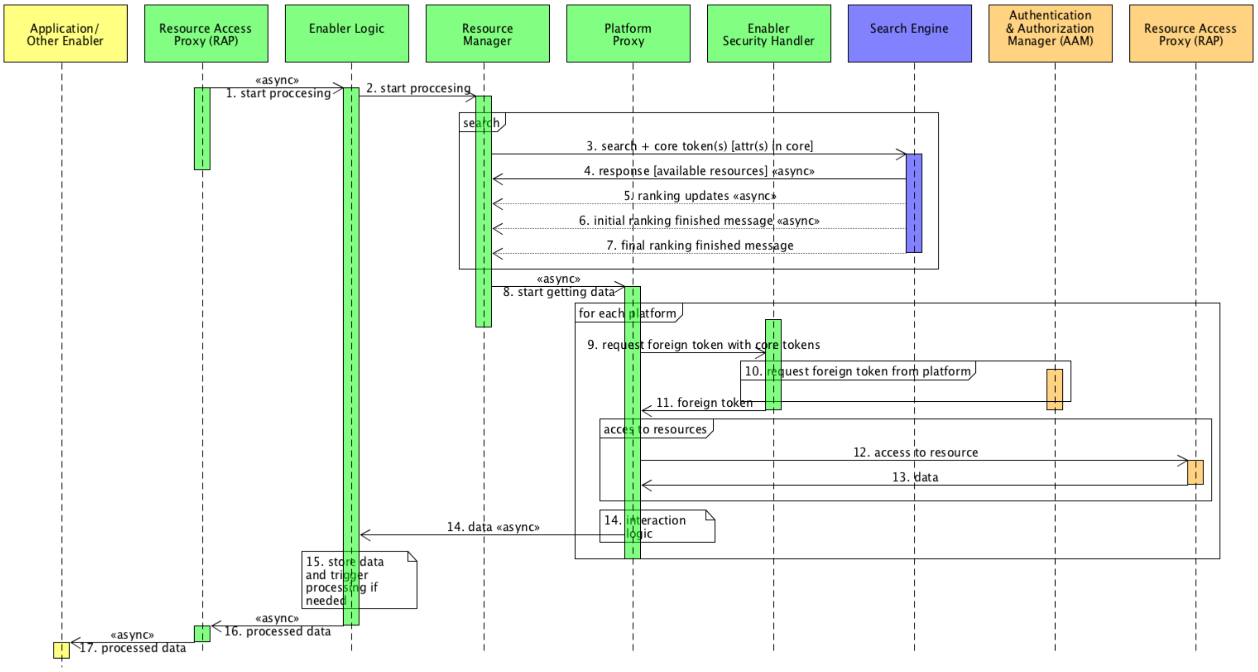
Application / Other Enabler
Resource Access Proxy (RAP)
Enabler Logic
Resource Manager
Platform Proxy
Enabler Security Handler
Search Engine
Authentication & Authorization Manager (AAM)
Resource Access Proxy (RAP)
«async»
1. start proccesing
2. start proccesing
search
3. search + core token(s) [attr(s) in core]
4. response [available resources] «async»
5. ranking updates «async»
6. initial ranking finished message «async»
7. final ranking finished message
«async»
8. start getting data
for each platform
9. request foreign token with core tokens
10. request foreign token from platform
11. foreign token
access to resources
12. access to resource
13. data
14. interaction logic
14. data «async»
15. store data and trigger processing if needed
«async»
16. processed data
«async»
17. processed data

Figure 15 Using Underlying resources

Description:

- Message 1: When Resource Access Proxy in the Enablergets request for accessing resource and when it checks security and policy (Resource Access Diagram after message 23) it sends start processing to Processing Logic.
- Message 2: If Enabler Logic component has all data in local database and can give processed data to Resource Access Proxy (message 16) it skips messages 2-15. If it needs data then it sends this message to Resource Manager.
- Messages 3-7: This part is like in Search diagram. Messages between Resource Manager and Search Engine have been copied; details are in Search diagram.
- Message 8: Starts getting data that isneeded for Enabler Logic by sending async message to Platform Proxy
- Message 9: Generated by the Platform Proxy and sent to Enabler Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform. If the Enabler already has valid foreign token(s), it is not necessary.
- Procedure 10: This procedure does token validation and attribute mapping.
- Message 11: Result from procedure 10 is returning foreign token and attributes for accessing resource.
- Messages 12-13: is accessing resource. In this access, the platform side is validating token and checking policy.
- Message 12: Platform Proxy is sending request for accessing resource to Resource Access Proxy in IoT platform. This request contains foreign token and attributes for accessing resource.
- Message 13: After token validation and policy checking if everything is ok the Resource Access Proxy in IoT platform returns data to Platform Logic
- Procedure 14: Platform Proxy executes procedures if interaction logic (e.g. scheduled next request to resource).
- Message 14: Platform Proxy sends date to the Enabler Logic
- Procedure 15: Enabler Logiccan store data and process it.
- Message: 16: After Enabler Logic has processed data it sends data to Resource Access Proxy.
- Message: 17: Resource Access Proxy sends processed data to the Application/Other Enabler.

5.3.12 Using Underlying resource from an application using Domain-specific Interface

On request from an application, Enabler is accessing Underlying resources at IoT platforms. The application accesses Enabler through Domain-specific Interface.

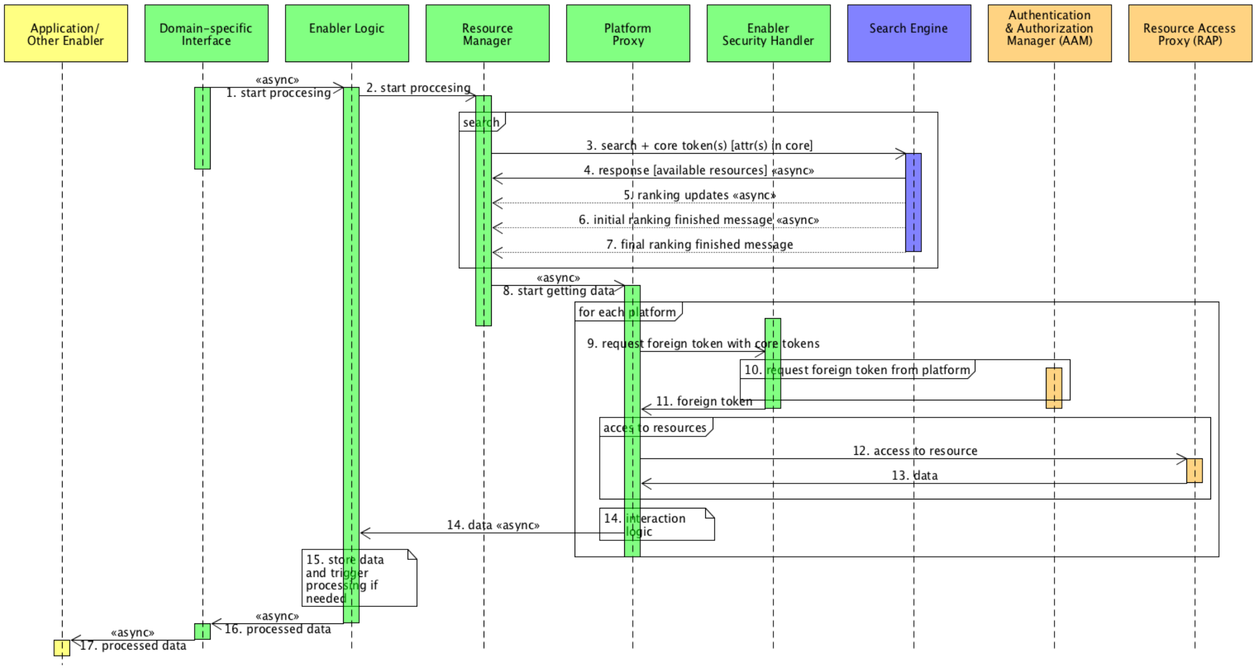
Application/ Other Enabler
Domain-specific Interface
Enabler Logic
Resource Manager
Platform Proxy
Enabler Security Handler
Search Engine
Authentication & Authorization Manager (AAM)
Resource Access Proxy (RAP)
«async» 1. start proccesing
2. start proccesing
search
3. search + core token(s) [attr(s) in core]
4. response [available resources] «async»
5. ranking updates «async»
6. initial ranking finished message «async»
7. final ranking finished message
«async» 8. start getting data
for each platform
9. request foreign token with core tokens
10. request foreign token from platform
11. foreign token
access to resources
12. access to resource
13. data
14. interaction logic
14. data «async»
15. store data and trigger processing if needed
«async» 16. processed data
«async» 17. processed data

Figure 16 Using underlying resources

Description:

- Message 1: When Domain-specific Interface in the Enablergets request for accessing resource and when it checks security and policy (Resource Access Diagram from enabler application in procedure 12) it sends start processing to Processing Logic.
- Message 2: If Enabler Logic has all data in local database and can give processed data to Domain-specific Interface (message 16) it skips messages 2-15. If it needs data, then it sends this message to Resource Manager.
- Messages 3-7: This part is like in Search diagram. Here are shown just messages between Resource Manager and Search Engine. Details are in Search diagram.
- Message 8: Starts getting data that isneeded for Enabler Logic by sending async message to Platform Proxy
- Message 9: Generated by the Platform Proxy and sent to Enabler Security Handler. It is used to trigger the operations for obtaining the foreign token(s) from IoT platform. If the Enabler already has valid foreign token(s), it is not necessary.
- Procedure 10: This procedure does token validation and attribute mapping.
- Message 11: Result from procedure 10 is returning foreign token and attributes for accessing resource.
- Messages 12-13: is accessing resource. In this access, the platform side is validating token and checking policy.
- Message 12: Platform Proxy is sending request for accessing resource to Resource Access Proxy in IoT platform. This request contains foreign token and attributes for accesing resource.

- Message 13: After token validation and policy checking if everything is ok the Resource Access Proxy in IoT platform returns data to Platform Logic
- Procedure 14: Platform Proxy executes procedures if interaction logic (e.g. scheduled next request to resource).
- Message 14: Platform Proxy sends date to the Enabler Logic
- Procedure 15: Enabler Logic can store data and process it.
- Message: 16: After Enabler Logic has processed data it sends data to Domain-specific Interface.
- Message: 17: Domain-specific Interface sends processed data to the application.

5.3.13 Enabler resource usage monitoring

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

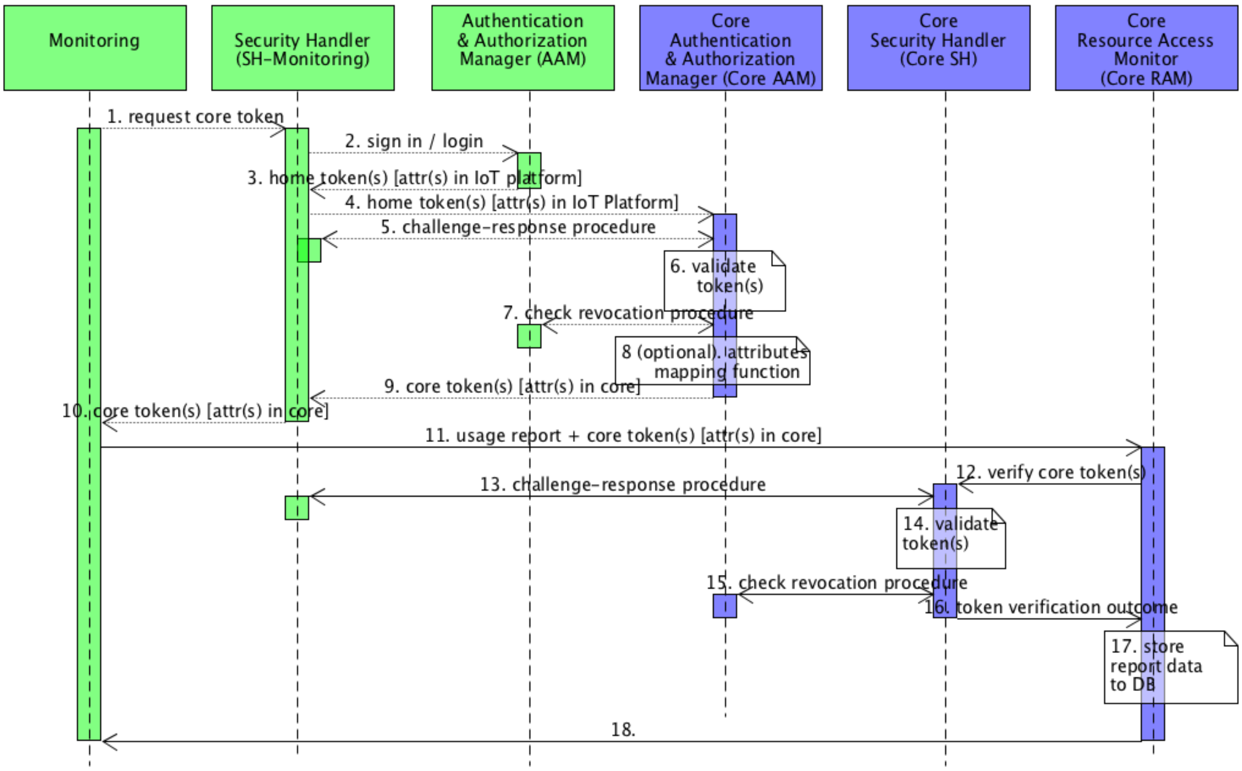
Monitoring
Security Handler (SH-Monitoring)
Authentication & Authorization Manager (AAM)
Core Authentication & Authorization Manager (Core AAM)
Core Security Handler (Core SH)
Core Resource Access Monitor (Core RAM)
1. request core token
2. sign in / login
3. home token(s) [attr(s) in IoT platform]
4.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

Figure 17 Enabler resource usage monitoring

Description:

- Message 1 (optional): Generated by the Monitoring and sent to the Security Handler. It is used to trigger the recovery of the core token(s). If the Monitoring is already logged in, it is not necessary.
- Message 2 (optional): Generated by the Security Handler and sent to the home (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 layer. It is used to trigger the operations for obtaining the core token(s). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 5 (optional): Procedure that allows the Security Handler that is acting on behalf of the Monitoring to demonstrate that it is the real owner of the token(s). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 6 (optional): Verification of the time validity, authenticity and integrity of the provided token(s). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 7 (optional): Verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the home AAM before the expiration time indicated within the token itself). If the Monitoring already has valid core token(s), it is not necessary.
- Procedure 8 (optional): Procedure that, in case it is needed, translates attributes that the Monitoring has in the home IoT platform in a new set of attributes that it has in the core layer. If attributes are the same or the Monitoring already has valid core token(s), it is not necessary.
- Message 9 (optional): Generated by the Core AAM and sent to the Security Handler. It is used to deliver the core token(s) with the new attribute(s). If the Monitoring already has valid core token(s), it is not necessary.
- Message 10(optional): Generated by the Security Handler and sent to the Monitoring. It is used to forward the core token generated at the previous step.
- Message 11: Monitoring generates usage report and sent it to the Core Resource Access Monitoring.
- Message 12: Generated by the Core Resource Access Monitoring and sent to the Core Security Handler. It is used to ask to the security handler to verify the complete validity of the token.
- Procedure 13: Procedure that allows the Security Handler that is acting on behalf of the Monitoring to demonstrate that it is the real owner of the token(s).
- Procedure 14: Verification of the time validity, authenticity and integrity of the provided token(s).
- Procedure 15: Verification of any asynchronous revocation of the token(s) (i.e., if any token(s) have been revoked by the Core AAM before the expiration time indicated within the token itself).
- Message 16: Generated by the Security Handler in the core layer and sent to the Core Resource Access Monitoring. It is used to communicate the outcome of the token validation procedures performed by the Core Security Handler.
- Message 17: Stores report data to database
- Message 18: returns call

6 symbIoTe Use Case Enablers

This Section focuses on generic Enabler components (Resource Manager, Platform Proxy, and Enabler Logic) which are domain-specificand relevant to symbIoTe use cases. It is envisioned that symbIoTe use caseswill use Enablers(if and when appropriate) to facilitate application development.Enablers performall processes toidentify, access and analyze the data originatedfromthe underlying IoT resources.

An example of Enabler is designed for the Smart Mobility & Ecological Routing (SMER) use, and here we present its domain-specific components. Workflows of the SMER use case are mapped to the Enabler Architecture, and Enabler components are identified which are responsible for executing processes within the use case.Domain-specific Enablers related to other use cases defined within symbIoTe will be considered in the following phases of the project.

6.1 Smart Mobility & Ecological Routing Use Case

The Smart Mobility and Ecological Routing (SMER) 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 will empower this usecase by providing platform interoperability, allowing for developers to easily access and handle data from different platforms and domains in the same manner.

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

Enablers facilitate the implementation of SMER 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.Example of such application could be monitoring air quality in the city to alarm citizens in the event of pollution with potential peril for human health.

6.2 SMER workflows and Enabler Architecture

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 5, Enabler-specific components (Enabler Logic, Platform Proxy and Resource Manager) need to be created and implementedaccording to domain-specific requirements. In SMER domain, Enabler-specific components are defined based on the aforementioned workflows.

Data Acquisition workflow is planned to be 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 Routing Service (RS), and Point of Interest (PoI) Search Service. Routes and PoIs are found by combining best routes and PoIs with air-quality data. Those external services can also be situated within IoT platforms (as RS within MoBaas). Even this 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.

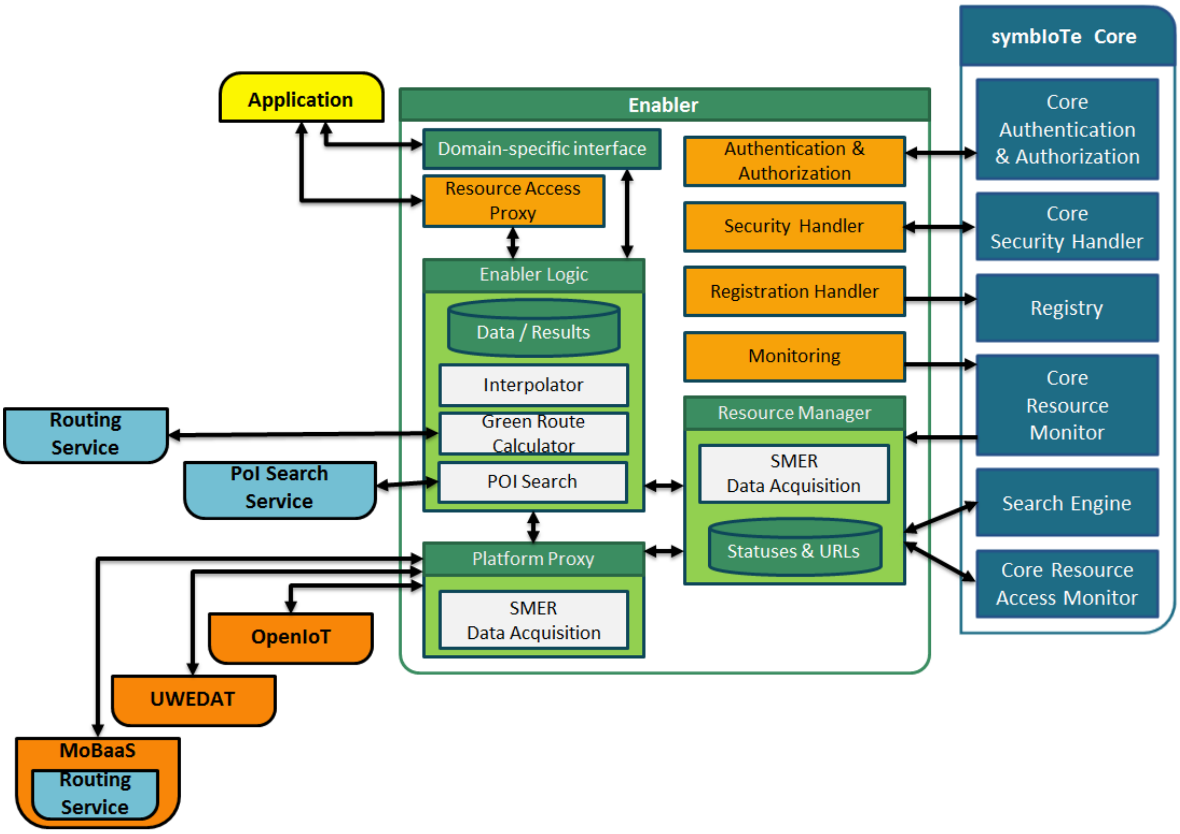

Figure 18 Architecture of Smart Mobility & Ecological Routing Enabler

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

6.3 Data Acquisition

The workflow Data acquisition is responsible for findingresources and acquiring sensor data from underlying IoT platforms: mobile sensors from UniZG-FER's OpenIoT platform, fixed stations from AIT's UWEDAT System and sensors fromUbiwhere'sMoBaaS platform. Resource Manager is responsible for finding the resources, and Platform Proxy for acquiring data from resources. 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 will be handled by components within Enabler Logic, and offered to applications.

6.4 Data Interpolation

The workflow Data Interpolation finds relevant resources by using Resource Manager and takes sensor readings as an input. Ithas 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 – EnablerLogic.

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

6.4.1 Interface for measurement data access

The Interpolator component obtains its data from the underlying IoT platforms.Platform Proxy is responsible for acquiring the needed data.The selection of the resources tobe used within Interpolator is done through Resource Manager which contacts the symbIoTe Search Engine in Core Services.

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.
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.

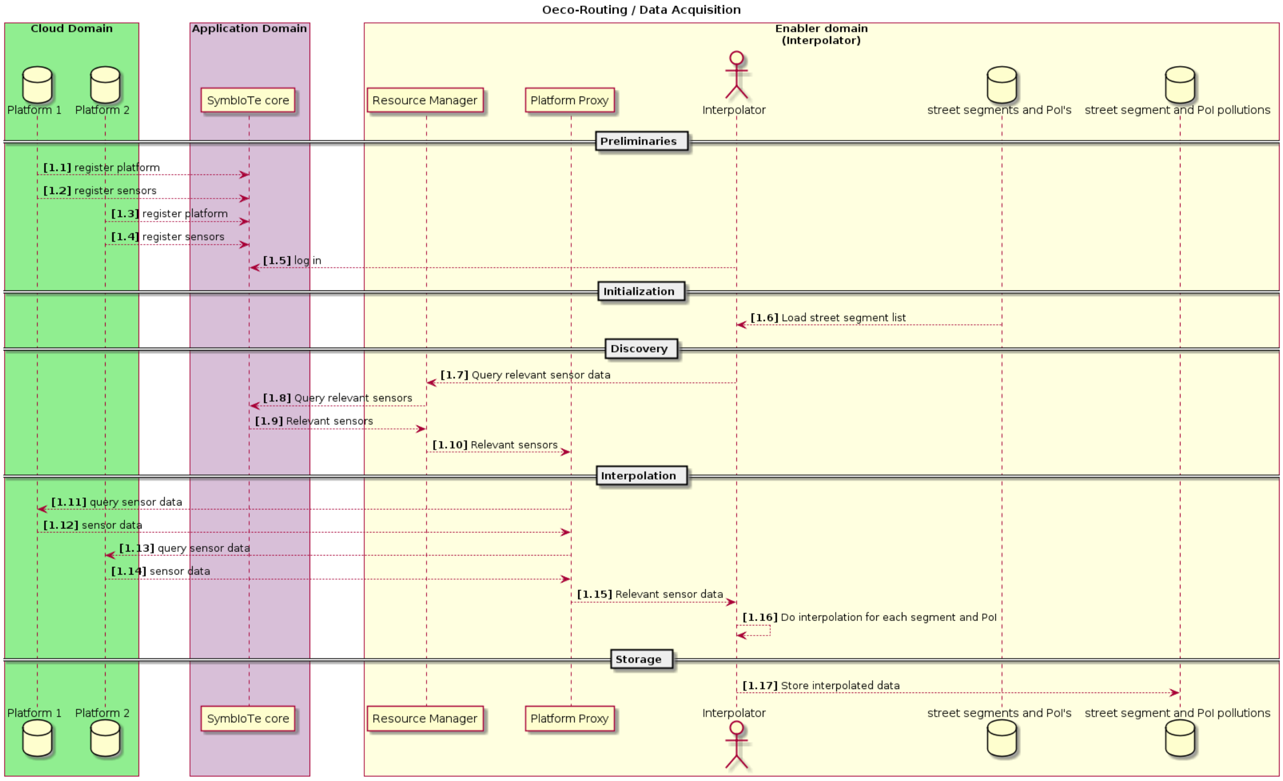

Figure 19 Data Interpolation

6.4.2 Interface for providing Interpolated Data

The Interpolator and the RS should 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. Thus, we need an interface to convey network data from the RS to the Interpolator. This interaction will be handled by the Green Route Calculator, and is explained in more detail in the Section 6.5.

Once an interpolation is available, theGreen Route Calculator 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 Calculator to obtain data in bulk. It also allows the Green Route Calculator to register for various artificial “sensors” events and get (only) relevant updates within near real-time. This data can then be sent to the corresponding Routing Services.

6.4.3 Implementation architecture and environment

A convenient set of libraries for the interpolation task is available for the Python (CPython) environment. Especially Java does not provide a similar convenient functionality. Thus the implementation will be in Python.

6.5 Calculation of Green Route

The workflow Calculation of Green Route is planned to be implemented in Enabler-specific component Green Route Calculator. 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). These data will be 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 Calculator will be 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.

It is important to note that, the difference between external Routing Services and platform Routing Services is that the platform service must communicate with the Enabler through the Platform Proxy while the external service communicates with the Green Route Calculator directly.

6.5.1 Obtaining Data

The RS consumes the street air quality data provided by the Interpolator to be able to provide routes through areas with low pollution. The main concern with this process is that, whenever a route request is made, it is not 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, only updates from that data (and not the whole data) are obtainedin 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 will be made 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).

There are two distinct flows designed to implement the functionalities mentioned above. The first one is represented in Figure 20, where the Green Route Calculator register a RS. In return, the RS send its preferences, stating what data it wishes 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 Calculator will store the platform’s preferences and, additionally, send them to the Interpolator. This way, the Interpolator knows from which sensors it needs to obtain data. It then requests from the Interpolator the air quality data. This data is returned to the platform to be stored and later used for the calculations of routes.

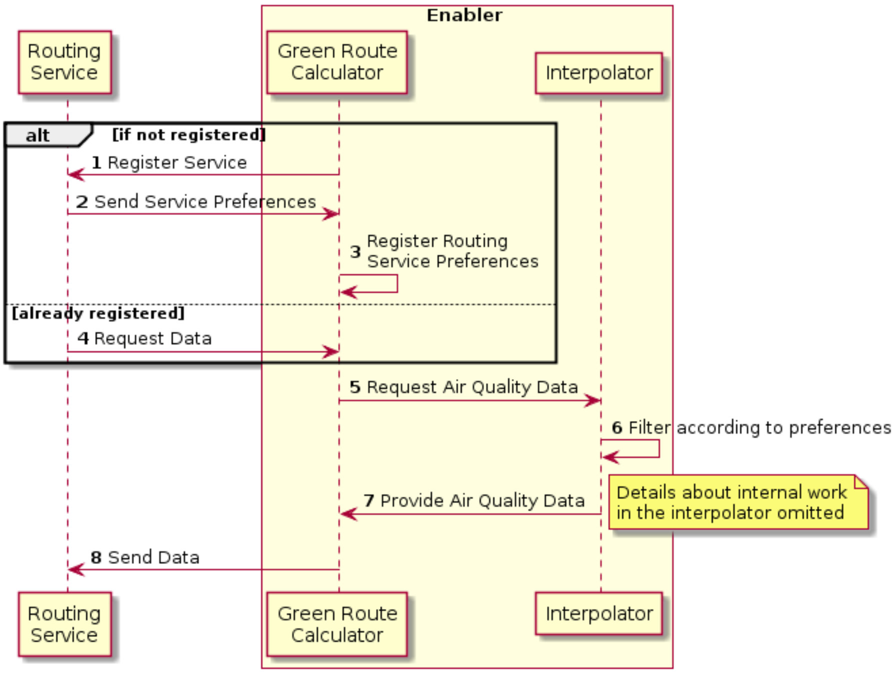
Enabler
Routing Service
Green Route Calculator
Interpolator
alt [if not registered]
1 Register Service
2 Send Service Preferences
3 Register Routing Service Preferences
[already registered]
4 Request Data
5 Request Air Quality Data
6 Filter according to preferences
7 Provide Air Quality Data
Details about internal work in the interpolator omitted
8 Send Data
Routing Service
Green Route Calculator
Interpolator

Figure 20 Registration and Bulk Data Request – external RS

The Figure 21 shows the same case as in Figure 20 but this time the RS is in the platform and Green Routing Calculator need to communicate by using Platform Proxy and RAP.

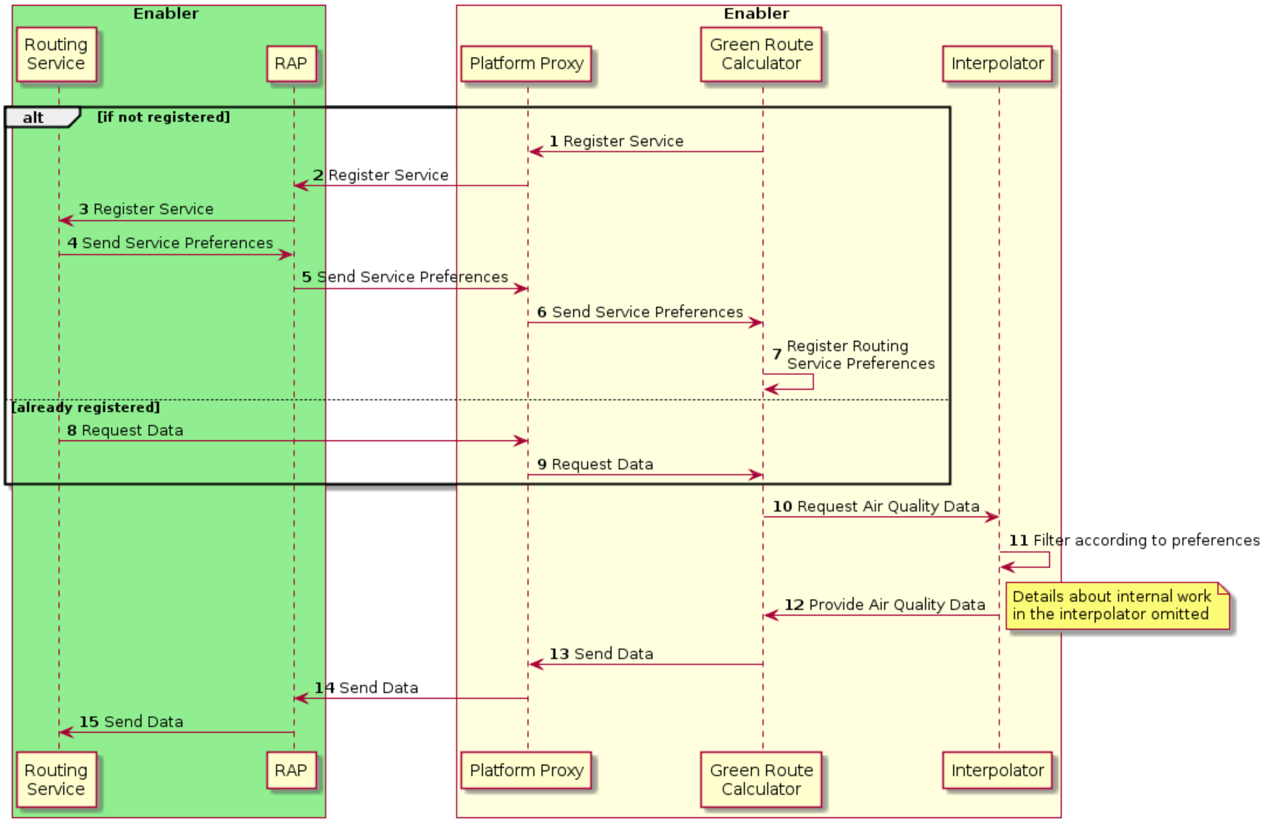

Figure 21 Registration and Bulk Data Request – RAP

The second flow relates to obtaining data updates from the Interpolator. Only changes to the state of the air quality in the streets 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 already has all the information it needs. As can be seen in Figure 22, the Interpolator sends its data updates to the Green Route Calculator, which sends the relevant information to RS.

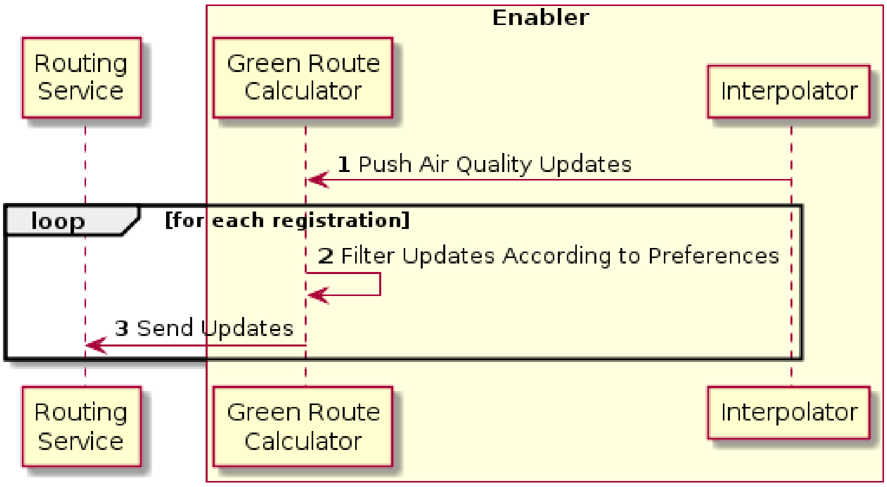

Figure 22 Air Quality Data Updates – external RS

The Figure 23 shows the same case as in Figure 22 but this time the RS is in the platform and Green Routing Calculator needs to communicate by using Platform Proxy and RAP.

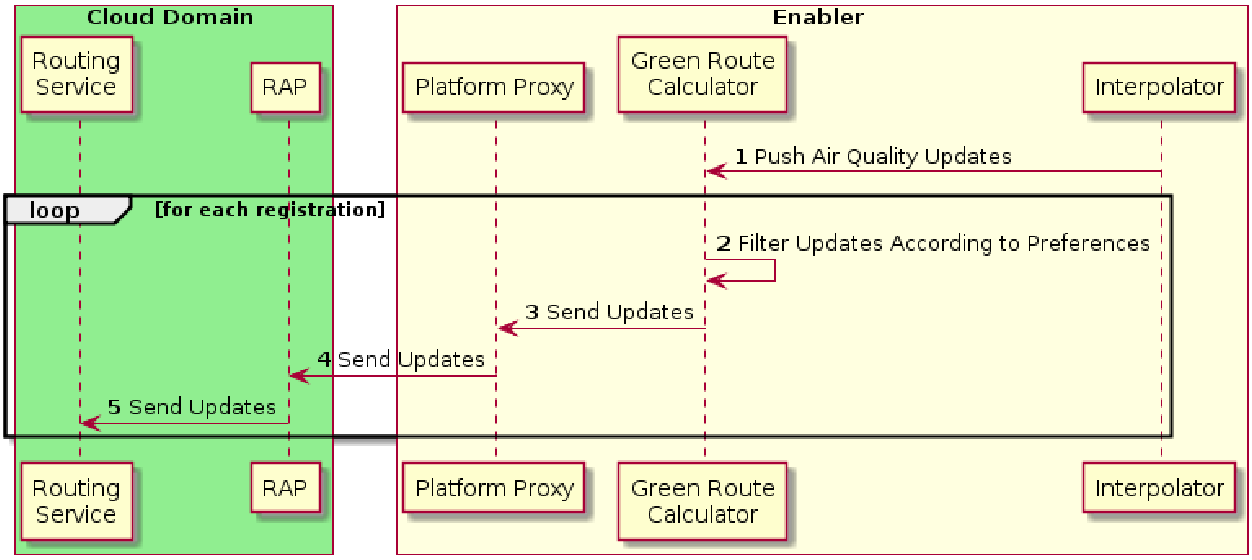

Figure 23 Air Quality Data Updates – RAP

Following these diagrams, the RS will be kept updated regarding the quality of the air of the relevant streets.

6.5.2 Obtaining Route

As can be seen in Figure 24, the process of obtaining an ecological route will be more direct due to most of the complex interactions between services being done in the previous section. A registered application makes a request to the Green Route Calculator within theEnabler, who directs it to thecorrect RS. The RS uses the collected air quality data, plus any other data that can help (e.g. traffic data) and returns the result. For this particular use case, the used RSwill be provided by AIT and by Ubiwhere'sMoBaaS platform.

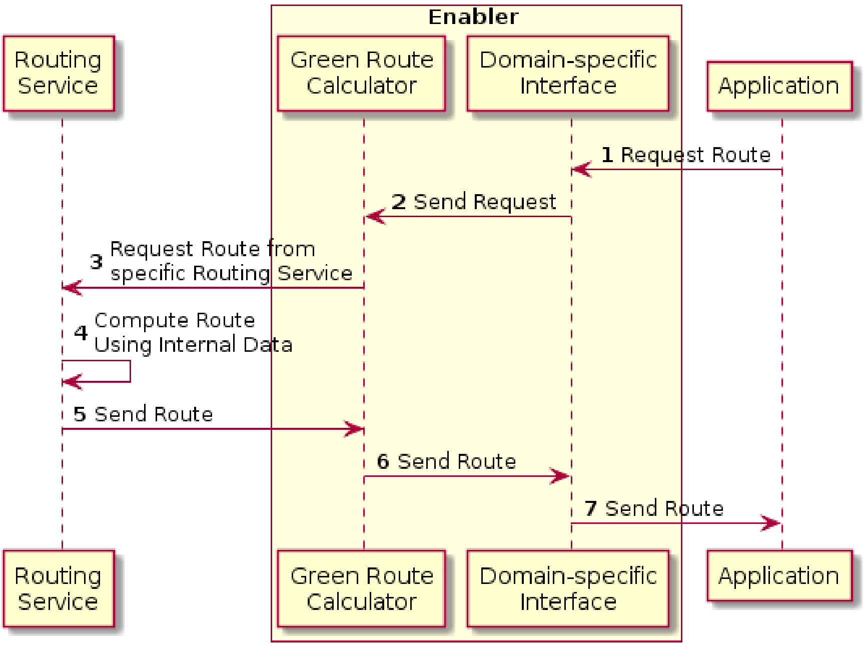

Figure 24 Obtaining Ecological Route – external RS

The Figure 25 is the same as Figure 24 but this time the RS is in the platform and Green Routing Calculator need to communicate by using Platform Proxy and RAP.

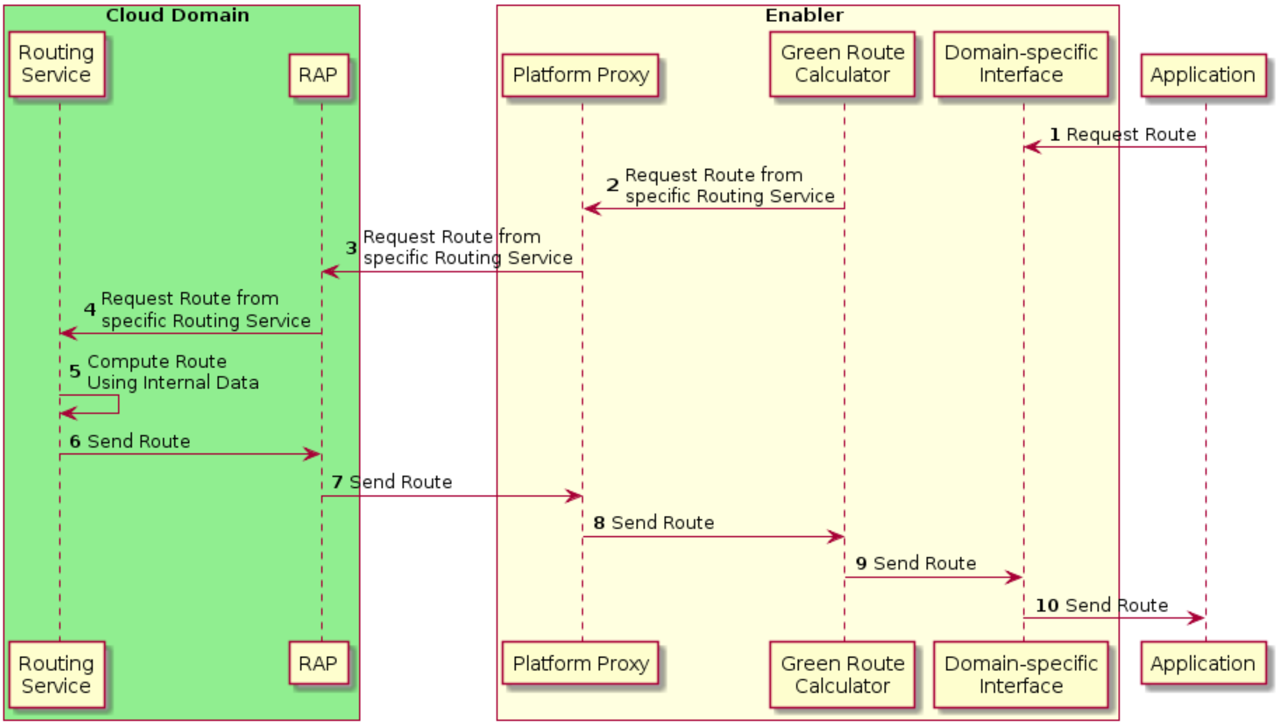

Figure 25 Obtaining Ecological Route – RAP

6.6 Point of Interest Search

The workflow Point of Interest (PoI) Search is planned to be implemented in an Enabler-specific component PoI Search. The workflow allows users to search for PoIs, such as restaurants or bars. Furthermore, it allows filtering of the results through stated preferences, such as noise levels or available parking. These data will be obtained through symbIoTe-compliant platforms. It can then be possible for an application to use PoI Search component to obtain a destination and use the Green Route Calculator component to find a way to reach it.

Ascan be observed in Figure 26, an application makes a request to the Enabler, asking for PoIs near a certain place and stating their preferences regarding the conditions of the PoI. The PoI Search component will request from an externalPoISearch Service the PoIs near that place. The PoI Search component will abstract this PoI Search Service for the application. For this use case, the OpenStreetMap (OSM) service will be used, but others, such as Foursquare, can eventually be integrated and used. After obtaining the PoIs, the PoI Search component, for each item, will search for relevant resources availablethroughCore Services. The Resource Manager will handle the resource search and the Platform Proxy will take care of obtaining the data from the sensors near requested point. If the acquired resources show that the PoI does not comply with the user preferences, that PoI is discarded. The PoIs that do comply are then returned to the application.

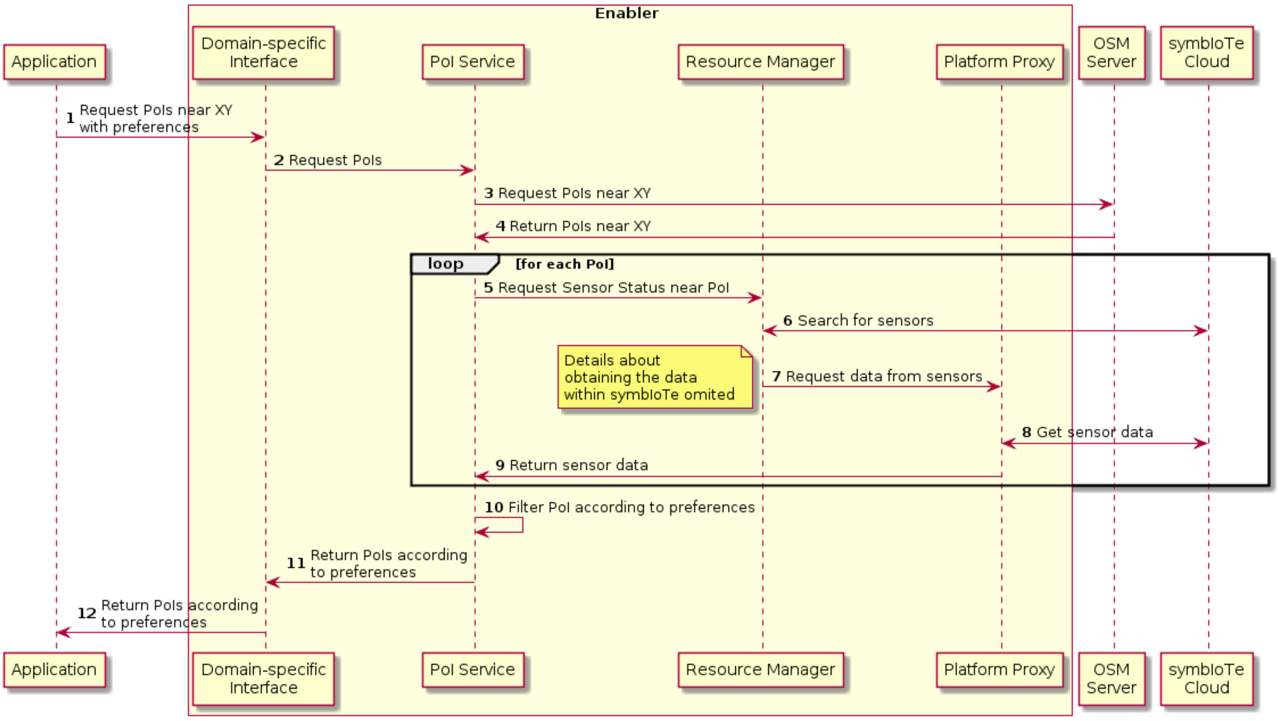
Enabler
Application
Domain-specific Interface
PoI Service
Resource Manager
Platform Proxy
OSM Server
symbIoTe Cloud
1 Request PoIs near XY with preferences
2 Request PoIs
3 Request PoIs near XY
4 Return PoIs near XY
loop [for each PoI]
5 Request Sensor Status near PoI
6 Search for sensors
Details about obtaining the data within symbIoTe omited
7 Request data from sensors
8 Get sensor data
9 Return sensor data
10 Filter PoI according to preferences
11 Return PoIs according to preferences
12 Return PoIs according to preferences
Application
Domain-specific Interface
PoI Service
Resource Manager
Platform Proxy
OSM Server
symbIoTe Cloud

Figure 26 Point of Interest Search

7 Technologies for Enabler Implementation

This Section lists selected technologies considered to be used for Enabler implementation. It has been derived from a complete collection of technologies relevant to the development of the symbIoTe project that has been reported in Deliverable "D5.1: Implementation Framework".

7.1 Spring Cloud

Spring Cloud is a complete solution that offers seamless integration with Framework and Boot tools. The easy use of Java, combined with the wide and active communities that contribute the Spring project have been the main reason for choosing this tool.
In particular:

- the Spring Framework provides powerful mechanisms to help a faster development of applications
- Spring Boot allows developers to create Spring-based applications easily

7.2 OData

OData is a REST-based communication protocol built on the HTTP defining a standard for data-centric API. It provides standards and best practices for performing Create-Read-Update-Delete (CRUD) operations on REST-based resources as well as querying collections of resources.
These are the main reasons why OData have been selected for the resource access and discovery, for symbIoTe-enabled platforms.

7.3 SensorThings API

The OGC SensorThings API is a standard HTTP/REST-based communication API and data model for transferring sensor data and metadata. Since it is based on the OData protocol the symbIoTe data model have been defined following this standard implementation.

7.4 Apache Jena

Apache Jena is an open source Java framework for building Semantic Web and Linked Data applications. It offers tools for storing and manipulating RDF data as well as an HTTP interface or triple store, and it provides support for the Web Ontology Language (OWL).
In the context of symbIoTe the Apache Jena framework is used to store, process and query data in RDF format.

7.5 JSON Web Tokens

JSON Web Token (JWT) represents a compact instrument for transferring trusted information between two parties, where information (*claim*) is encoded as a JSON object. JWT includes a Message Authentication Code or a Signature that allows verifying its authenticity and integrity as well.

Within the symbIoTe project, JWT is used for carrying trusted access permissions (i.e., attributes), useful to support ABAC techniques.

8 Conclusion and Future Steps

This document presents Domain-Specific Enablers within the symbIoTe ecosystem, designed according to system requirements, and reports their main functionalities and generic architecture. Enablers are components placed in the symbIoTe Application Domain that facilitate third-party application development by offering mechanisms to find resources registered by L1 Compliant IoT Platforms, access the recourses directly on the platform side, add value to those resources, and offer value-added services to applications. Furthermore, the resources acquired and processed by Enablers and offered to third-party applications are registered in symbIoTe Core, thus enabling other application providers to find and to use them. Enablers have a generic architecture, but with components that can be tailored according the needs of aspecific domain. These Enabler-specific components handle the application logic, data queries and data access.

We have proposed a Domain-Specific Enabler for a selected symbIoTe use case – Smart Mobility and Ecological Routing. Enabler-specific components are described for this Enabler, as well as its main functionalities.The document also presents a list of technologies considered to be used for Enabler implementation. During the next phases of the project, the process of implementing generic Enabler components will start, as well as the definition of Enabler-specific components for other symbIoTe use cases (where applicable).

9 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/

10 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 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

11 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
SMER	Smart Mobility and Ecological Routing