Symbiosis of smart objects across IoT environments

688156 - symbIoTe - H2020-ICT-2015

Integrated Prototype and Developed Applications

The symbIoTe Consortium

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

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

Document Control

Number: D5.4

Title: Integrated Prototype and Developed Applications

Type: Public

Editor(s): Ilia Pietri, ICOM
E-mail: ilpiet@intracom-telecom.com

Author(s): Vasilis Glykantzis, ICOM, Gerhard Duennebeil, Karl Kreiner, Christoph Ruggenthaler, AIT, Petar Krivic, Ivana Podnar Žarko, Pavle Skocir, UNIZG-FER, Matteo Pardi, Luca Tomaselli, NXW, Luca De Santis, NAVIGO, Michael Jacoby, IOSB, Jose Antonio Sanchez Murillo, ATOS, Szymon Mueller, Mikolaj Dobski, Jakub Toczek, Roman Łapacz, PSNC, Matteo Di Fraia, UNIDATA, Reinhard Herzog, IOSB, Joao Garcia, UW, Antonio Paradell Bondia, Juan Belmonte Rodriguez, WLI, Raquel Ventura Miravet, S&C, Zvonimir Zelenika, VIP

Doc ID: D5_4-v09

Amendment History

Version	Date	Author	Description/Comments
V0.1	May 8th	I. Pietri	Initial ToC
V0.2	May 25th	I. Pietri, V. Glykantzis, G. Duennebeil	Assignment for contributions to partners
V0.3	May 29th	I. Pietri, V. Glykantzis, G. Duennebeil	Finalised ToC
V0.4	June 15th	I. Pietri, P. Krivic, M. Pardi, L. De Santis, P. Skocir, J. Garcia, M. Jacoby, J. A. Sanchez Murillo, S. Mueller, K. Kreiner, M. Di Fraia, R. Herzog, V. Glykantzis	Merged first round of contributions. Added input from partners
V0.5	July 5th	I. Pietri, V. Glykantzis, M. Dobski,J. Toczek, C. Ruggenthaler, J. Garcia	Merged second round of contributions. Added input from ICOM, AIT, PSNC and UW partners.
V0.6	July 6th	I. Pietri G. Duennebeil, R. Ventura Miravet, A. Paradell Bondia	Merged new round of contributions. Added input from AIT, S&C, WLI partners.
V0.7	July 17th	I. Pietri, , V. Glykantzis, M. Dobski, M. Pardi	Draft for internal review. Edits by ICOM and added missing input from PSNC and NXW partners.
V0.8	July 24th	I. Pietri, I. Podnar Žarko, R. Łapacz, J. Garcia, K. Kreiner	Draft with merged comments from internal reviews. Edits and comments by Corinna and Roman. Added missing inputs from UW and AIT partners.
V0.9	July 27th	I. Pietri, V. Glykantzis, R. Ventrura Miravet, L. De Santis, Z. Zelenika, M. Di Fraia, J. Garcia, R. Herzog, M. Jacoby, L. Tomaselli, K. Kreiner, J. Belmonte Rodriguez	Document to submit. Some final edits from involved partners to address the comments of reviewers, added enabler applications.

Table of Contents

1 Executive Summary 5
2 Introduction 7
2.1 symbIoTe 7
2.1.1 System Architecture 8
2.2 Purpose of the Document and Scope 10
2.3 Task T5.1 Objectives 10
2.4 Task T5.2 Objectives 11
2.5 Document Structure 11
3 Prototype Integration 12
3.1 Project Build and Deployment 12
3.2 Common Integration Information across Components 12
3.2.1 Programming Language 12
3.2.2 Generic Source Tree Information 13
3.2.3 Building Tool 13
3.2.4 External Tools 14
3.2.5 Continuous Integration 14
3.3 Software Releases 14
4 Integrated Prototype 16
4.1 Common Java Dependencies across Components 16
4.2 Common Components/Libraries 16
4.2.1 symbIoTeLibraries 16
4.2.2 symbIoTeSecurity 17
4.2.3 SymbIoTeSemantics 19
4.2.4 SemanticMapping 20
4.2.5 Authentication and Authorization Manager 21
4.3 Core Components 23
4.3.1 Administration 23
4.3.2 Cloud-core Interface 25
4.3.3 Core Interface 25
4.3.4 Core Resource Access Monitor 26
4.3.5 Core Resource Monitor 27
4.3.6 Registry 28
4.3.7 Search 29
4.3.8 Semantic Manager 31
4.3.9 Core Bartering and Trading Manager 32

4.3.10 Core Anomaly Detection 32
4.4 Platform Components 34
4.4.1 Federation Manager 34
4.4.2 Monitoring 34
4.4.3 Platform Registry 35
4.4.4 Registration Handler 36
4.4.5 Resource Access Proxy 37
4.4.6 Subscription Manager 38
4.4.7 Trust Manager 39
4.4.8 Bartering & Trading Manager 40
4.4.9 SLA Manager 40
4.5 Smart Space Components 41
4.5.1 Smart Space Middleware 41
4.5.2 SDEV SymbIoTe Agent 42
5 Developed Applications 44
5.1 Applications Developed for symbIoTe Use Cases 44
5.1.1 Smart Residence 44
5.1.2 Smart Mobility and Ecological Routing 55
5.1.3 EduCampus 59
5.1.4 Smart Stadium 63
5.1.5 Smart Yachting 74
5.2 Applications Developed for Demos 80
5.2.1 Demo Web App 80
5.3 Enabler-based Applications 82
5.3.1 Indoor Positioning based on Location Enabler 82
6 Conclusions 91
7 References 92
8 Acronyms 93

1 Executive Summary

The purpose of this deliverable is to document the final release of the symbIoTe software prototype. The symbIoTe project aims to provide a cooperation environment for Internet of Things (IoT) platforms to interoperate, collaborate and share resources by implementing IoT platform federations. It also has the objective to provide a uniform interface for the client applications to access different types of IoT platforms and to facilitate the implementation of cross-platform and cross-domain IoT applications.

To make this possible, applications as well as IoT platforms, need to be interoperable. symbIoTe builds an interoperability middleware and offers four interoperability modes for platforms to select the desired collaboration level with other platforms within the symbIoTe-enabled ecosystem. The four interoperability modes, referred to as “Compliance Levels” (L1 to L4), aim to enable an incremental deployment of functionalities across four architectural domains identified: 1) Application Domain, 2) Cloud Domain, 3) Smart Space Domain, and 4) Device Domain.

Among the main activities of Work Package 5 (WP5) “Use-case based Trials and Deployments” are the following: (i) the integration of the system components into a software prototype and (ii) the implementation of applications based on the use cases designed in T1.1 of WP1 to utilize symbIoTe APIs; these activities are the focus of this deliverable, D5.4. Our work relies on previous symbIoTe deliverables, especially D1.2 [1] and D5.2 [2] that describe in detail the system requirements and architecture, the components developed in earlier software releases, as well as their role, interactions and basic features incrementally deployed.

The first part of this deliverable documents the final release of the symbIoTe prototype including information about the build and deployment of the system, the structure of the project, the source tree and the maintained repositories, developed software components and their main features. A microservices architecture which allows for better scalability, performance and code maintenance in a highly-distributed environment was selected for the development of the project to design and provide distributed, performance-oriented IoT services.

The second part of the deliverable focuses on the implementation of applications designed to utilize the symbIoTe ecosystem (use case-related and demo applications) and the deployment of initial functional tests used to validate their features. Five use cases of symbIoTe that make use of different applications have been identified:

1) *Smart Residence* with Smart Healthy Indoor Air, Smart Area Controller, Home Comfort and Smart Health Mirror applications.
2) *Smart Mobility and Ecological Routing* with Mobile and Web applications offering optimized routing alternatives and point of interest search.
3) *EduCampus* with Searching for a Room application.
4) *Smart Stadium* with Visitor, Retailer and Promowall applications.
5) *Smart Yachting* with Portnet and Centrale Acquisti applications.

Most of the aforementioned use cases target L1 and L2 compliance levels. Smart Residence and Smart Yachting use cases also require extensions for L3 and L4 compliance. Different components are needed depending on the different domains and compliance levels desired. The IoT platforms need to integrate the required symbIoTe

components according to these compliance levels and the applications need to be designed and developed accordingly. The selected applications are used to test the usability of developed middleware in practice and for specific domains. They comprise an excellent example to demonstrate the development of new or adjustment of existing applications to provide services in the symbIoTe ecosystem and test the software prototype across existing platforms that interoperate and use a plethora of available resources (sensors, actuators, mobile devices, processing resources, etc.) within a symbIoTe-enabled cooperation environment.

In summary, this deliverable documents the work done in Tasks T5.1 and T5.2 for the system integration and implementation of the use case-related symbIoTe applications. Extensive trials and deployment of the symbIoTe platform and the developed applications with end users have been planned and will continue until month 33 and 36 in Tasks T5.4 and T5.5, respectively. The trials and final results will be presented in Deliverable D5.6.

2 Introduction

This section gives an overview of symbIoTe and summarizes the purpose of this document and the activities that correspond to the objectives mentioned in the Description of Action (DoA). Finally, the structure of the document follows.

2.1 symbIoTe

The use of smart objects in different domains of our life is rapidly increasing with actuators and sensors monitoring from personal activities to environmental and traffic data. As a result, the need for transparent and secure access to and usage of the available resources across various IoT domains to provide daily life services and satisfy the needs of an increasingly connected society emerges. Currently, IoT is evolving around a plethora of vertical solutions specifically suited to given scenarios. Although such solutions integrating connected objects within local environments like home and office coexist, they cannot cooperate to enable cross domain applications as they often adopt non-standard, sometimes fully proprietary protocols to control the variety of sensors and actuators. Application developers and providers are locked in with a platform and need to adjust their solutions to each new platform and underlying infrastructure, while infrastructure providers cannot offer their resources to multiple IoT service providers.

The symbIoTe project aims to address the challenging task of remedying this fragmented environment of isolated IoT ecosystems offering an abstraction layer for a unified view on various platforms and their sensing/actuating resources in a way that resources are transparent to application designers and developers. It creates a cooperation environment for IoT platforms to securely interoperate, collaborate and share resources for the mutual benefit (IoT platform federations). Last but not least, it enables the implementation of dynamic smart spaces where smart objects can seamlessly migrate and roam between various IoT domains and platforms. The example, Figure 1 is used to illustrate the symbIoTe's vision to provide a cooperation environment for various IoT domains: Smart Home environments (Platform 1), Smart office/Smart Campus environments (Platform 2) and public spaces solutions (Platform 3).

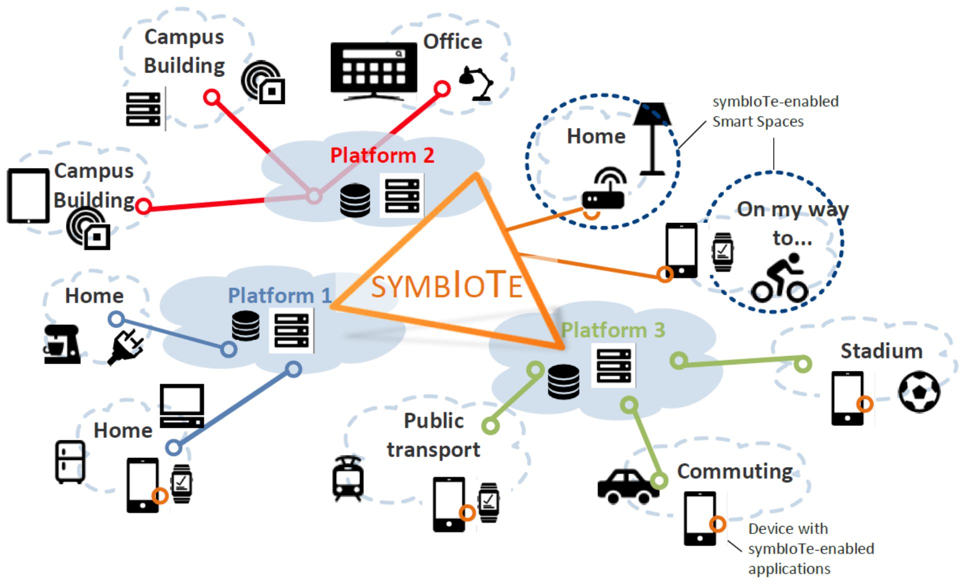


Figure 1: SymbIoTe ecosystem

2.1.1 System Architecture

The symbIoTe approach is built around a layered IoT stack connecting various devices (sensors, actuators and IoT gateways) within Smart Spaces (local environments with connected objects) with the Cloud. In this section we give an overview of the symbIoTe architecture. More details can be found in deliverables D1.2 [1] and D1.4 [3], the reports on the initial and final system architecture and requirements, respectively.

The symbIoTe architecture consists of four layered domains (shown in Figure 2):

1) *Application Domain (APP)*, which offers a high-level API to provide a unified view on the symbIoTe's IoT environments and support cross-platform discovery and management of resources.
2) *Cloud Domain (CLD)* that hosts the cloud-adjusted building blocks of specific platforms to enable platform collaboration and sharing of resources in accordance with platform-specific business rules.
3) *Smart Space Domain (SSP)* which consists of smart objects, IoT gateways and local and storage computing resources and enables dynamic sensor discovery and configuration within local smart spaces.
4) *Smart Device Domain (SD)* that spans over heterogeneous smart devices and their roaming capability to dynamically blend with a surrounding smart environment and get discovered to interact with devices in the visited smart space according to predefined access policies.

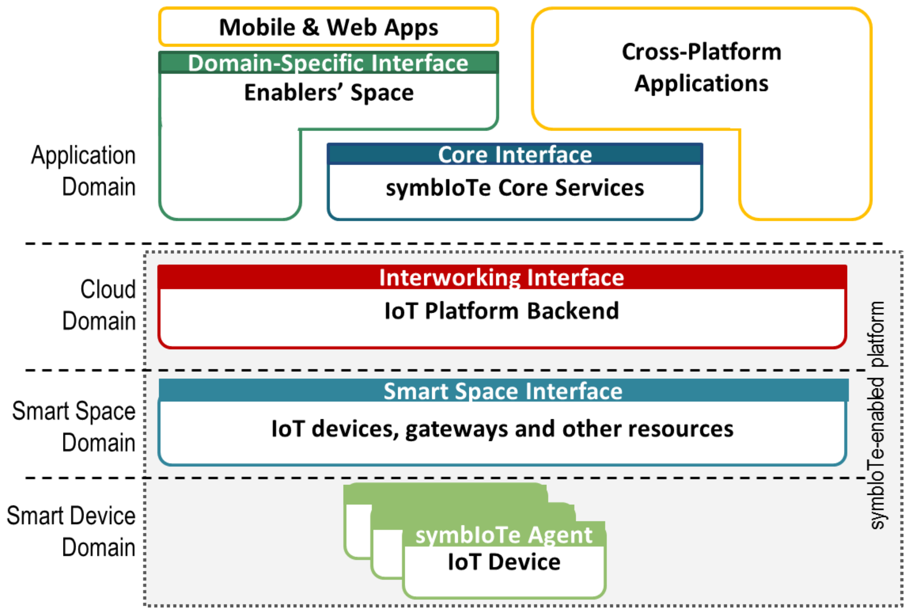


Figure 2: The symbIoTe architecture

Also, the symbIoTe approach offers four interoperability mechanisms (compliance levels), as depicted in Figure 3, to enable an incremental deployment of functionalities across the architectural domains (namely, APP, CLD, SSP and SD) and allow IoT platforms to define the appropriate level of integration of symbIoTe-specific services in order to achieve the desired level of cooperation within a symbIoTe-enabled ecosystem.

1) *Level 1 compliance (L1)* offers an open symbIoTe-defined platform interface within the Cloud Domain so that platform resources and IoT services are searchable within the symbIoTe Core Services.
2) *Level 2 compliance (L2)* implements functionality needed for platform federations and direct platform to platform interworking for resource bartering and trading.
3) *Level 3 compliance (L3)* supports dynamic smart spaces.
4) *Level 4 compliance (L4)* supports device roaming in visited domains so that a smart device can use services in a visited smart space.

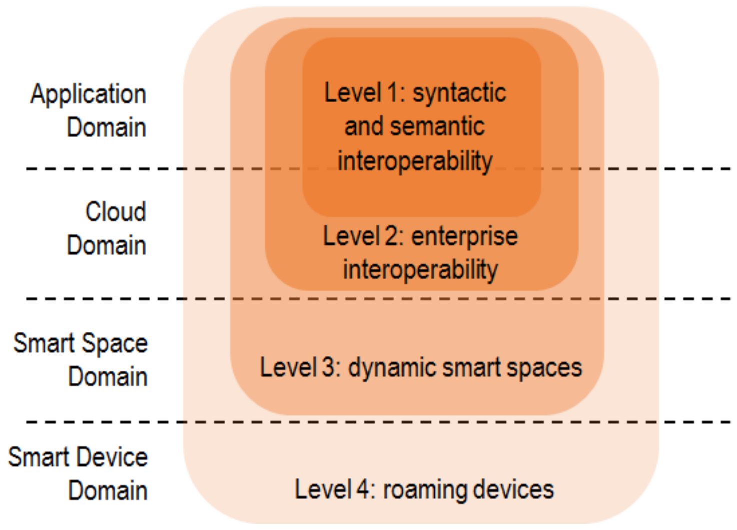

Application Domain
Cloud Domain
Smart Space Domain
Smart Device Domain
Level 1: syntactic and semantic interoperability
Level 2: enterprise interoperability
Level 3: dynamic smart spaces
Level 4: roaming devices

Figure 3: symbIoTe compliance levels

Different components are required to be installed in different domains according to the desirable compliance level. The project defines the components based on the domain in which they are placed, rather than which set of features or compliance level they provide.

The symbIoTe project with all the developed components is available in GitHub[1]. In the microservice architecture used the components are bundled into super-repositories that make use of the git submodules according to the domain; the symbIoTe *Core*[2] repository contains all the components belonging to the core, while components needed in the respective platform side can be found in the symbIoTe *Cloud*[3] repository. Finally, each environment considered as smart space needs to deploy the components in the symbIoTe *Smart Space Middleware*[4] repository.

2.2 Purpose of the Document and Scope

The purpose of Deliverable D5.4 “Integrated Prototype and Developed Applications” is to document the final symbIoTe prototype and the developed use case-related applications running on top of it based on the work done in Tasks T5.1 and T5.2.

2.3 Task T5.1 Objectives

Task T5.1 defines an implementation framework for the symbIoTe architecture. It serves as a guide for all the implementation tasks (T2.2, T2.3, T3.3, T4.1, T4.2 and T4.3) in the technical WPs by setting common methodologies, tools and workflows for the development of the symbIoTe components. A second set of activities concerns the integration of the system components developed in previous WPs (WP2, WP3 and WP4) into a working software prototype.

[1] https://github.com/symbiote-h2020

[2] https://github.com/symbiote-h2020/SymbioteCore

[3] https://github.com/symbiote-h2020/SymbioteCloud

[4] https://github.com/symbiote-h2020/SymbioteSmartSpace

2.4 Task T5.2 Objectives

Task T5.2 implements the symbIoTe applications based on the use cases defined in Tasks T1.1 and T1.3. The implementation phase of the symbIoTe use case-related applications, including scenarios, tools and application workflows, is followed by initial functional tests to validate their features. T5.2 also interacts with the 2nd symbIoTe Open Call partners (WP6) who base their applications on the defined domain-specific symbIoTe APIs. To this end, it provides an empirical set of guidelines for the implementation of symbIoTe applications.

2.5 Document Structure

The rest of the deliverable is organized as follows. Section 3 provides information about the prototype integration (the build and installation process, etc.), while Section 4 presents the symbIoTe prototype and the integration of individual components. Section 5 describes the design and implementation of use case driven applications, followed by functional tests. Section 6 concludes the deliverable.

3 Prototype Integration

This section provides information about the system integration, covering the structure of the developed software, build and deployment requirements, the software releases, the functionalities of the individual components and dependencies specific to them.

3.1 Project Build and Deployment

All source codes are available in Github:

- `https://github.com/symbiote-h2020/SymbioteCore` for all core components.
- `https://github.com/symbiote-h2020/SymbioteCloud` for platform-side components.
- `https://github.com/symbiote-h2020/SymbioteSmartSpace` for environments considered as smart spaces.

The project also maintains the following repositories/libraries:

- *symbIoTeLibraries*[5] repository, which is a set of common models, definitions and methods used by components at all layers.
- *symbIoTeSecurity*[6] repository, which implements the symbIoTe-specific security solution.
- *symbIoTeSemantics*[7] repository, which contains the necessary ontology files for the symbIoTe framework and its use cases.
- *semanticMapping*[8] repository, which implements the semantic mapping solution of the symbIoTe framework.
- *Ontologies*[9] repository, which contains the information models and ontologies created.

Information on how to install and build all components can be found at the Wikipage of each repository (e.g., `https://github.com/symbiote-h2020/SymbioteCore/wiki`).

3.2 Common Integration Information across Components

3.2.1 Programming Language

For developing the symbIoTe framework, the project chose the Java programming language, since it is a very popular, easy to write, compile and debug, while it offers

[5]https://github.com/symbiote-h2020/symbIoTeLibraries

[6]https://github.com/symbiote-h2020/ symbIoTeSecurity

[7]https://github.com/symbiote-h2020/symbIoTeSemantics

[8]https://github.com/symbiote-h2020/symbIoTeMapping

[9]https://github.com/symbiote-h2020/Ontologies

numerous frameworks (e.g., Spring[10]). Furthermore, the choice of Java is highly compatible with the microservices approach chosen, since there are many available frameworks that facilitate the development of microservices (e.g., Spring Boot[11], Spring Cloud[12]).

Even though symbIoTe is entirely developed in Java, the project provides standard communication mechanisms to all the symbIoTe components (e.g., https and message queues). Therefore, the platform-specific plugins required to enable the integration of an IoT platform to the symbIoTe framework are language independent and must not necessarily be developed in Java.

3.2.2 Generic Source Tree Information

The symbIoTe developers follow the Standard Directory Layout for Java and all the symbIoTe components include the following subfolders:

- `src/main/java/eu/h2020/symbiote`: Application/Library sources.
- `src/main/resources`: Application/Library resources (e.g., bootstrap.properties configuration file of Spring Boot).
- `src/test/java/eu/h2020/symbiote`: Test sources.
- `src/test/resources`: Test resources.

3.2.3 Building Tool

Gradle[13] was the final choice for a building tool, due to its simplicity of creating and maintaining building scripts, its extensive documentation and good performance. In order to facilitate the simplicity and quality of the development procedure, the project used the following Gradle plugins listed in Table 1.

Table 1: Gradle plugins

Plugin	Version	Description
java	default	Plugin necessary for java
org.springframework.boot	1.5.14.RELEASE	Plugin necessary for Spring Boot
io.spring.dependency-management	1.0.0.RELEASE	A Gradle plugin that provides Maven-like dependency management functionality
jacoco	default	Gradle plugin that generates Jacoco reports from a Gradle Project.
org.owasp.dependencycheck	3.0.2	A software composition analysis plugin that identifies known vulnerability dependencies used by the project.
eclipse	default	Plugin for Eclipse

[10]https://spring.io/

[11] https://projects.spring.io/spring-boot/

[12]http://projects.spring.io/spring-cloud/

[13]https://gradle.org/

`idea`	`default`	Plugin for Intellij IDEA
`com.cinnober.gradle.semver-git`	`2.2.2`	Gradle plugin that combines git tags and semantic versioning, and sets the gradle version property accordingly.

3.2.4 External Tools

symbIoTe has further dependencies on external tools, so that certain aspects of the projects can be easily realized:

- *RabbitMQ*[14] (version 3.6.+): message queue server for internal messaging between same domain components. RabbitMQ's use is granted under a "Mozilla Public License".
- *MongoDB*[15] (version 3.6+): database used by symbIoTe components. The related license is the "GNU AFFERO GENERAL PUBLIC LICENSE".
- *Icinga 2*[16]: for monitoring registered resources. Icinga is licensed under the terms of the GNU General Public License Version 2.
- *Nginx*[17] (version 1.12.+): for enabling access of platform components with the external world (i.e., applications, enablers, symbIoTe core). Nginx is released under the terms of a BSD-like license.

3.2.5 Continuous Integration

During implementation, the project team used the branching model as described in Deliverable D5.1 [4] along with the continuous integration server *Travis*[18]. The test reports are also automatically pushed to *codecov*[19], a reporting tool used to group, merge, archive and compare coverage reports. Specific testing information per component is provided in the next section (Section 4).

3.3 Software Releases

The project has performed five major releases of the symbIoTe software (R1-R5) summarized in the following table. The date of release and the respective GitHub links are provided in the following table.

Table 2: Software releases

Release	Date	GitHub Link
0.1.0 (R1)	21/02/2017	Core: `https://github.com/symbiote-h2020/SymbioteCore/releases/tag/0.1.0` Cloud: `https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/0.1.0`

[14]https://www.rabbitmq.com/

[15] https://www.mongodb.com/

[16]https://www.icinga.com/products/icinga-2/

[17] https://nginx.org/en/

[18] https://travis-ci.org/

[19]https://codecov.io/github/symbiote-h2020

0.2.0 (R2)	22/05/2017	Core: https://github.com/symbiote-h2020/SymbioteCore/releases/tag/0.2.0 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/0.2.0
0.2.1	20/07/2017	Core:https://github.com/symbiote-h2020/SymbioteCore/releases/tag/0.2.1 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/0.2.1
1.0.0 (R3)	17/10/2017	Core:https://github.com/symbiote-h2020/SymbioteCore/releases/tag/1.0.0 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/1.0.0
1.1.0	16/11/2017	Core:https://github.com/symbiote-h2020/SymbioteCore/releases/tag/1.1.0 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/1.1.0
1.2.0	10/04/2018	Core: https://github.com/symbiote-h2020/SymbioteCore/releases/tag/1.2.0 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/1.2.0
2.0.0 (R4)	16/05/2018	Core: https://github.com/symbiote-h2020/SymbioteCore/releases/tag/2.0.0 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/2.0.0
3.0.0 (R5)	17/08/2018 (expected)	Core: https://github.com/symbiote-h2020/SymbioteCore/releases/tag/3.0.0 Cloud: https://github.com/symbiote-h2020/SymbioteCloud/releases/tag/3.0.0

4 Integrated Prototype

4.1 Common Java Dependencies across Components

Towards simplifying and accelerating the implementation procedure, the project team also made use of the Spring framework and specifically of the Spring Boot (1.5.11.RELEASE) and Spring Cloud (Dalston.RELEASE) projects. All the common Java dependencies across projects are listed in Table 3.

Table 3: Java dependencies

Group Id	Artifact Id	Version	Type
org.springframework.cloud	spring-cloud-starter-config	Dalston.SR5	compile
org.springframework.cloud	spring-cloud-starter-eureka	Dalston.SR5	compile
org.springframework.cloud	spring-cloud-starter-zipkin	Dalston.SR5	compile
org.springframework.boot	spring-boot-starter-amqp	1.5.14.RELEASE	compile
com.github.symbiote-h2020	SymbIoTeLibraries	5.+	compile
junit	junit	4.+	testcompile

4.2 Common Components/Libraries

4.2.1 symbIoTeLibraries

4.2.1.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/SymbIoTeLibraries
URL of javadoc	https://symbiote-h2020.github.io/SymbIoTeLibraries/doxygen
URL of code coverage reports	https://codecov.io/github/symbiote-h2020/SymbIoTeLibraries
Code coverage snapshot	26%

4.2.1.2 Feature History

Release	Main Features
5.0.0	• Support for Composite Access Policies
5.1.0	• Adding observed property by iri to the core query request
5.4.0	• Added necessary classes for Platform Registry
5.7.0	• Added property UOM iri implementation
5.11.0	• Adding input parameters and capabilities to query response class
5.14.0	• Return resource urls to Enabler Logic

5.15.0	• Added Trust values to resources
5.16.0	• Finalized Smart Space classes
5.17.0	• Finalized Federation classes
5.18.0	• Added GDPR-compliant SymbIoTeSecurity dependency

4.2.1.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`joda-time`	`joda-time`	`2.9.9`
`javax.validation`	`validation-api`	`2.0.0.Final`
`io.swagger`	`swagger-annotations`	`swaggerAnnotationsVersion`
`com.querydsl`	`querydsl-mongodb`	`4.1.4`

4.2.1.4 Component Source Tree Information

The SymbIoTeLibraries contain the following packages:

- `/src/main/java/eu/h2020/symbiote/client`: offers a set of clients for communicating with symbIoTe components.
- `/src/main/java/eu/h2020/symbiote/cloud`: classes used in the Cloud Domain.
- `/src/main/java/eu/h2020/symbiote/core/cci`: classes used for communication with between the platform and Core components.
- `/src/main/java/eu/h2020/symbiote/core/ci`: classes used for communication with between applications and Core components.
- `/src/main/java/eu/h2020/symbiote/core/internal`: classes used for internal communication of the symbIoTe Core components.
- `/src/main/java/eu/h2020/symbiote/enabler`: a set of classes facilitating the communication of the Enabler components.
- `/src/main/java/eu/h2020/symbiote/model/cim`: a set of classes describing the symbIoTe information Core Information Model (CIM).
- `/src/main/java/eu/h2020/symbiote/model/mim`: a set of classes describing the symbIoTe Meta Information Model (MIM).
- `/src/main/java/eu/h2020/symbiote/util`: general helper classes.

4.2.2 symbIoTeSecurity

4.2.2.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/SymbIoTeSecurity`
URL of javadoc	`https://symbiote-h2020.github.io/SymbIoTeSecurity/doxygen`
URL of code coverage reports	`https://codecov.io/gh/symbiote-h2020/SymbIoTeSecurity`
Code coverage snapshot	40%

4.2.2.2 Feature History

Release	Main Features
21.4.0	• constants, errors and enums used throughout the system • single token access policies that can be used out of the box • AAM client implementation • Thin java security handler and component security handler clients with corresponding factory • Group of helper classes: generating keystores, validating security responses, converting certificates to PEMs etc.
23.0.0	• Readme containing basic instructions added • Component Home Token Access Policy added • Better error handling in AAMClient • Payloads hardening • Security Handlers locality added
23.2.0	• Logging support added • ability to work in platforms disconnected from SymbIoTe Core
24.2.0	• SSP payloads added • extended IComponentSecurityHandler access policies resolver to optionally provide external cache of validated credentials • Composite Access Policies introduction • Keystore certificate mismatch with registered one check in Component Security Handler • Public method checking trust chain of the certificates
25.0.0	• Certificate Key Store Factory generalized to support Platform, Enabler and SmartSpace AAMs
25.4.0	• Improved verbosity of the service response validator
25.7.0	• Anomaly Detection Module payloads and client added • SingleFederatedTokenAccessPolicy changed
26.0.0	• Release missing due to jitpack bug
27.0.0	• GDPR compliance release introduced: ○ user service terms agreement consent ○ research and marketing consent ○ user statuses which can cause blocked access to symbiote services without required service terms agreement • updated dependencies
27.1.0	• Bartering And Trading support added

4.2.2.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
io.jsonwebtoken	jjwt	0.9.1
org.bouncycastle	bcprov-jdk15on	1.60
org.bouncycastle	bcpkix-jdk15on	1.60
io.github.openfeign	feign-jackson	9.7.0
commons-logging	commons-logging	1.2

javax.xml.bind	jaxb-api	2.3.0

4.2.2.4 Component Source Tree Information

The SymbIoTeSecurity library includes 6 main packages:

- `src/main/java/eu/h2020/symbiote/security/accesspolicies`: offers a set of default access policies that can be used out of the box.
- `src/main/java/eu/h2020/symbiote/security/clients`: offers some factories creating clients for the essential modules.
- `src/main/java/eu/h2020/symbiote/security/commons`: contains constants, errors and enums used throughout the system.
- `src/main/java/eu/h2020/symbiote/security/communication`: contains communication interfaces, used payloads and clients to modules responsible for security, such as AnomalyDetectionModule, AuthenticationAuthorizationManager and BarteringTradingModule.
- `src/main/java/eu/h2020/symbiote/security/handler`: thin java clients used throughout different components and different layers. It contains methods that allow the clients to acquire authorization credentials, service to evaluate the received credentials in terms of both authorizing operations and authenticating the clients and finally the clients to verify the authenticity of service they interact with.
- `src/main/java/eu/h2020/symbiote/security/helpers`: helper classes containing methods used for security purposes.

4.2.3 SymbIoTeSemantics

4.2.3.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/SymbIoTeSemantics`
URL of javadoc	`https://symbiote-h2020.github.io/SymbIoTeSemantics/doxygen`
URL of code coverage reports	`https://codecov.io/gh/symbiote-h2020/SymbIoTeSemantics`
Code coverage snapshot	45%

4.2.3.2 Feature History

Release	Main Features
1.0.0	• Helper functions for handling semantic data • Java classes representing information models v2.1.0
2.2.0	• Updated to information model v2.2.0
2.3.0	• Updated to information model v2.3.0
2.3.1	• Added more helper methods for SPARQL execution

4.2.3.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.apache.jena	jena-core	3.4.0
org.apache.jena	jena-querybuilder	3.4.0
org.apache.jena	jena-cmds	3.4.0

4.2.3.4 Component Source Tree Information

The SymbIoTeSemantic component includes three main folders:

- `src/main/java/eu/h2020/symbiote/semantics`: the helper classes this library provides.
- `src/main/java/eu/h2020/symbiote/semantics/ontology`: auto-generated classes representing all information models provided by symbIoTe (CIM, BIM, MIM, internal).
- `src/main/java/eu/h2020/symbiote/semantics/util`: utility classes used for implementation.

4.2.4 SemanticMapping

4.2.4.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/SemanticMapping
URL of Javadoc	https://symbiote-h2020.github.io/SemanticMapping/doxygen
URL of code coverage reports	https://codecov.io/gh/symbiote-h2020/SemanticMapping
Code coverage snapshot	47%

4.2.4.2 Feature History

Release	Main Features
1.0.0	• Initial commit, prototype status • parser & printer for mapping DSL • SPARQL query re-writing • RDF data transformation

4.2.4.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.apache.jena	jena-core	3.4.0
org.apache.jena	jena-querybuilder	3.4.0
org.apache.jena	jena-cmds	3.4.0
org.apache.jena	jena-arq	3.4.0
org.reflections	reflections	0.9.11

4.2.4.4 Component Source Tree Information

The SemanticMapping component includes six main folder:

- `src/main/java/eu/h2020/symbiote/semantics/mapping/model`: the main and model classes for semantic mapping
- `src/main/java/eu/h2020/symbiote/semantics/mapping/data`: classes related to RDF data transformation.
- `src/main/java/eu/h2020/symbiote/semantics/mapping/sparql`: classes related to SPARQL query re-writing.
- `src/main/java/eu/h2020/symbiote/semantics/mapping/utils`: utility classes.
- `src/main/java/eu/h2020/symbiote/semantics/mapping/parser`: the parser for the mapping language.
- `src/main/jjtree`: input data for automatic parser generation.

4.2.5 Authentication and Authorization Manager

4.2.5.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/AuthenticationAuthorizationManager`
URL of javadoc	`https://symbiote-h2020.github.io/AuthenticationAuthorizationManager/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/AuthenticationAuthorizationManager`
Code coverage snapshot	76%

4.2.5.2 Feature History

Release	Main Features
0.1.0	• Initial release supporting o issuing of GUEST tokens o Platform management o User management
1.0.0	• L1 compliance release • Certificate issuing • Token acquisition (GUEST, HOME, FOREIGN) • User Details acquisition • Local attributes management • Platform Owners list acquisition • Owned platform details acquisition • Revocation service • Token and certificate validation features
1.1.0	• acquiring component certificate from DB • checking deployment type with certificate • strengthened validation • Home token acquisition using AMQP removed
1.2.0	• Caching added (available AAMs, getComponentCertificate, valid tokens) • Revocation check of remote tokens during validation added
1.3.2	• getAAMsInternally added • AMQP listeners reimplemented

	• Activation of the listener depending on the AAM role (Core, Platform)
2.0.0	• L2 support finalized • Federation Management • Federation attributes in tokens added
3.0.0	• L3/L4 support introduced • Smart Space handling added • Unused getPlatformOwners AMQP consumer removed • Core certificate checked during validation • Platform Agents management added
3.1.0	• Offline validation improved
4.0.0	• GDPR compliance introduced by managing users consents ○ service terms ○ marketing

4.2.5.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
io.github.swagger2markup	swagger2markup-spring-restdocs-ext	1.2.0
io.github.swagger2markup	swagger2markup-gradle-plugin	1.2.0
org.asciidoctor	asciidoctorj-pdf	1.5.0-alpha.10.1
io.spring.dependency-management	io.spring.dependency-management	1.0.4.RELEASE
org.springframework.boot	spring-boot-gradle-plugin	1.5.14.RELEASE
org.owasp	dependencycheck	3.2.0
com.cinnober.gradle	semver-git	2.3.1
org.springframework.cloud	spring-cloud-starter	Dalston.SR5
org.springframework.cloud	spring-cloud-starter-config	Dalston.SR5
org.springframework.cloud	spring-cloud-starter-eureka	Dalston.SR5
org.springframework.cloud	spring-cloud-starter-zipkin	Dalston.SR5
org.springframework.retry	spring-retry	Dalston.SR5
org.springframework.boot	spring-boot-starter-amqp	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-aop	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-cache	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-web	1.5.14.RELEASE
javax.xml.bind	jaxb-api	2.3.0
io.swagger	swagger-annotations	1.5.16

com.github.symbiote-h2020	SymbIoTeSecurity	27.0.0
com.github.symbiote-h2020	SymbIoTeLibraries	5.6.1

4.2.5.4 Component Source Tree Information

The Authentication and Authorization Manager component includes five main packages:

- `/src/main/java/eu/h2020/symbiote/security/commons`: containing enums used in the module.
- `/src/main/java/eu/h2020/symbiote/security/config`: configuration classes.
- `/src/main/java/eu/h2020/symbiote/security/listeners`: containing all the REST and AMQP interfaces and controllers/consumers.
- `/src/main/java/eu/h2020/symbiote/security/repositories`: repositories definitions with payloads.
- `/src/main/java/eu/h2020/symbiote/security/services`: all services responsible for the module logic.

4.3 Core Components

4.3.1 Administration

4.3.1.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/Administration
URL of Javadoc	https://symbiote-h2020.github.io/Administration/doxygen
URL of code coverage reports	https://codecov.io/github/symbiote-h2020/Administration
Code coverage snapshot	84%

4.3.1.2 Feature History

Release	Main Features
0.1.0	• Interface definition • User registration • Platform registration, modification, removal
0.2.0	• Interface definition (enhancements) • Visual improvements • App registration • Security credential passing to users
1.0.0	• Interface definition (enhancements) • List registered resources • Administrator user actions
1.1.0	• Clear platform resources as Administrator • Delete information model as Administrator • Getting platform configuration
1.2.0	• Integrated React.js • Platform update • Update of user email and password

	• Initial implementation of creating federations • Registration and deletion of SSPs
2.0.0	• Updated platform configuration feature for L2

4.3.1.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.springframework.boot	spring-boot-starter-data-rest	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-security	1.5.14.RELEASE
org.springframework.boot	spring-security-test	1.5.14.RELEASE
org.webjars	jquery	3.2.1
org.webjars	bootstrap	3.3.7-1
commons-validator	commons-validator	1.6

4.3.1.4 Component Source Tree Information

The Administration component includes the packages below:

- `src/main/java/eu/h2020/symbiote/administration/communication/rabbit`: contains classes responsible for RabbitMQ messaging.
- `src/main/java/eu/h2020/symbiote/administration/communication/controllers`: contains definitions of the REST endpoints.
- `src/main/java/eu/h2020/symbiote/administration/exceptions`: contains custom exceptions.
- `src/main/java/eu/h2020/symbiote/administration/model`: contains the model classes used in the component.
- `src/main/java/eu/h2020/symbiote/administration/repository`: contains classes which interact with the database.
- `src/main/java/eu/h2020/symbiote/administration/services`: contains the service classes which handle the requests.
- `src/main/java/eu/h2020/symbiote/administration/AppConfig.class`: provides the generic configuration of the Administration component.
- `src/main/java/eu/h2020/symbiote/administration/CustomAuthenticationProvider.class`: provides a custom AuthenticationProvider which communicates with Core AAM for authenticating and authorizing users.
- `src/main/java/eu/h2020/symbiote/administration/MvcConfig.class`: provides the MVC configuration of the Administration component.
- `src/main/java/eu/h2020/symbiote/administration/WebSecurityConfig.class`: provides the security configuration of the Administration component

4.3.2 Cloud-core Interface

4.3.2.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/CloudCoreInterface`
URL of Javadoc	`https://symbiote-h2020.github.io/CloudCoreInterface/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/CloudCoreInterface`
Code coverage snapshot	73.04%

4.3.2.2 Feature History

Release	Main Features
0.1.0	• Interface definition for resource registration (json and rdf), update and removal
0.2.0	• Adding security (X-Auth-Token header) to existing interfaces • Adding monitoring information interface
1.0.0	• Updating security to SecurityRequests • Adding resource access notification interface
1.1.0	• Adding clearData interface for synchronization between Cloud and Core
1.2.0	• Updates with respect to updated Libraries
2.0.0	• Updates with respect to updated Libraries

4.3.2.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`io.swagger`	`swagger-annotations`	`1.5.13`

4.3.2.4 Component Source Tree Information

The Cloud-Core Interface component includes two packages:

- `src/main/java/eu/h2020/symbiote/communication`: contains classes responsible for RabbitMQ messaging.
- `src/main/java/eu/h2020/symbiote/controllers`: contains definitions of the endpoints (REST services) to be used by the Cloud components.

4.3.3 Core Interface

4.3.3.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/CoreInterface`
URL of javadoc	`https://symbiote-h2020.github.io/CoreInterface/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/CoreInterface`
Code coverage snapshot	85.04%

4.3.3.2 Feature History

Release	Main Features
0.1.0	• Interface definition for parameterized search, sparql search and resource urls
0.2.0	• Adding security (X-Auth-Token header) to existing interfaces • Adding security related interfaces (login, getCaCert, getAvailableAAMs)
1.0.0	• Updating security to SecurityRequests • Adding more security related endpoints (getHome/Guest/ForeginToken, sign/revokeCertificate)
1.1.0	• Updates with respect to updated Libraries
1.2.0	• Updates with respect to updated Libraries
2.0.0	• Updates with respect to updated Libraries

4.3.3.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`io.swagger`	`swagger-annotations`	`1.5.13`

4.3.3.4 Component Source Tree Information

The Core Interface component includes two packages:

- `src/main/java/eu/h2020/symbiote/communication`: contains classes responsible for RabbitMQ messaging.
- `src/main/java/eu/h2020/symbiote/controllers`: contains definitions of the endpoints (REST services) to be used by the applications and Enablers.

4.3.4 Core Resource Access Monitor

4.3.4.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/CoreResourceAccessMonitor`
URL of Javadoc	`https://symbiote-h2020.github.io/CoreResourceAccessMonitor/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/CoreResourceAccessMonitor`
Code coverage snapshot	81%

4.3.4.2 Feature History

Release	Main Features
0.1.0	• Interface definition • User registration • Platform registration, modification, removal
0.2.0	• Interface definition (enhancements) • Visual improvements • App registration • Security credential passing to users

1.0.0	• Interface definition (enhancements) • List registered resources • Administrator user actions
1.1.0	• Clear platform resources as Administrator • Delete information model as Administrator • Getting platform configuration
1.2.0	• Integrated React.js • Platform update • Update of user email and password • Initial implementation of creating federations • Registration and deletion of SSPs
2.0.0	• Updated platform configuration feature for L2

4.3.4.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.springframework.boot	spring-boot-starter-data-rest	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE
org.springframework.boot	spring-amqp	2.0.0.M1
org.springframework.boot	spring-rabbit	2.0.0.M1

4.3.4.4 Component Source Tree Information

The Core Resource Access Monitor component includes the packages below:

- `src/main/java/eu/h2020/symbiote/cram/exceptions`: contains custom exceptions.
- `src/main/java/eu/h2020/symbiote/cram/managers/AuthenticationManager.class`: responsible for the authentication and authorization of the receiving requests.
- `src/main/java/eu/h2020/symbiote/cram/messaging`: contains classes responsible for RabbitMQ messaging.
- `src/main/java/eu/h2020/symbiote/cram/model`: contains the model classes used in the component.
- `src/main/java/eu/h2020/symbiote/cram/repository`: contains classes which interact with the database.
- `src/main/java/eu/h2020/symbiote/cram/utils`: contains custom utility classes.
- `src/main/java/eu/h2020/symbiote/cram/AppCon.fig.class`: provides the generic configuration of the CRAM component.

4.3.5 Core Resource Monitor

4.3.5.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/CoreResourceMonitor
URL of javadoc	https://symbiote-h2020.github.io/CoreResourceMonitor/doxygen

URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/CoreResourceMonitor`
Code coverage snapshot	45.58%

4.3.5.2 Feature History

Release	Main Features
1.0.0	• Collecting monitoring information from platforms, regarding availability and load information for resources
2.0.0	• Added security headers' checks

4.3.5.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`org.springframework.boot`	`spring-boot-starter-data-rest`	`1.5.14.RELEASE`
`org.springframework.boot`	`spring-boot-starter-data-mongodb`	`1.5.14.RELEASE`
`org.springframework.boot`	`spring-amqp`	`2.0.0.M1`
`org.springframework.boot`	`spring-rabbit`	`2.0.0.M1`

4.3.5.4 Component Source Tree Information

The Core Resource Monitor component includes the packages below:

- `src/main/java/eu/h2020/symbiote/crm/exceptions`: contains custom exceptions.
- `src/main/java/eu/h2020/symbiote/crm/managers`: responsible for the authentication and authorization of the receiving requests.
- `src/main/java/eu/h2020/symbiote/crm/interfaces`: contains classes responsible for RabbitMQ messaging.
- `src/main/java/eu/h2020/symbiote/crm/repository`: contains classes which interact with the database.
- `src/main/java/eu/h2020/symbiote/crm/resources`: contains classes for static definitions and application configuration.

4.3.6 Registry

4.3.6.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/Registry`
URL of javadoc	`https://symbiote-h2020.github.io/Registry/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/Registry`
Code coverage snapshot	60.08%

4.3.6.2 Feature History

Release	Main Features
0.1.0	• Initial implementation of the resource and platform handling (registration/update/removal)
0.2.0	• Integration with first iteration of Security • Adding communication with SemanticManager for resource validation
1.0.0	• Implementation of the symbiote federation handling • Implementation of the PIM handling
1.1.0	• Implementation of the clearData functionality
1.2.0	• Updates with respect to updated Libraries
2.0.0	• Updates with respect to updated Libraries

4.3.6.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
commons-io	commons-io	2.5
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE

4.3.6.4 Component Source Tree Information

The Registry component includes four main packages:

- `src/main/java/eu/h2020/symbiote/managers`: definition of three managers used by the component: RabbitMQ communication, authorization and repository.
- `src/main/java/eu/h2020/symbiote/consumers`: consumers used to handle incoming requests to the Registry.
- `src/main/java/eu/h2020/symbiote/model:` contains definitions of the classes used internally by this component.
- `src/main/java/eu/h2020/symbiote/repository:` contains MongoDB repository definitions.
- `src/main/java/eu/h2020/symbiote/utils:` helper methods used by other classes.

4.3.7 Search

4.3.7.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/Search
URL of Javadoc	https://symbiote-h2020.github.io/Search/doxygen
URL of code coverage reports	https://codecov.io/github/symbiote-h2020/Search
Code coverage snapshot	69.16%

4.3.7.2 Feature History

Release	Main Features
0.1.0	• Initial implementation definition for resource and platform handling (registration/update/removal) • Initial implementation of the search functionality (parameterized)
0.2.0	• Implementation of the sparql search functionality • Moving some functionalities to SemanticManager
1.0.0	• Implementation of the ranking algorithm • Integration with popularity information messages
1.1.0	• Integration with availability information message and adding it to ranking algorithm • Adding handling of the public and private resources (filtering) based on the SecurityRequest
1.2.0	• Performance improvements in sparql searching
2.0.0	• Updates with respect to updated Libraries and semantic models

4.3.7.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
com.github.symbiote-h2020	SymbIoTeSemantics	1.+
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE
org.apache.jena	jena-core	3.4.0
	jena-querybuilder	3.4.0
	jena-permissions	3.4.0
	jena-spatial	3.4.0

4.3.7.4 Component Source Tree Information

The Search component includes following packages:

- `src/main/java/eu/h2020/symbiote/communication`: definition of classes used in RabbitMQ communication, including consumers.
- `src/main/java/eu/h2020/symbiote/filtering`: classes used in handling filtering of the private resources in Jena.
- `src/main/java/eu/h2020/symbiote/handlers`: contain feature specific implementations used by various functionalities.
- `src/main/java/eu/h2020/symbiote/ontology`: package containing Jena specific classes.
- `src/main/java/eu/h2020/symbiote/query`: contains classes responsible for query construction.
- `src/main/java/eu/h2020/symbiote/ranking`: contains classes responsible for ranking functionality.
- `src/main/java/eu/h2020/symbiote/search`: search engine.

4.3.8 Semantic Manager

4.3.8.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/SemanticManager`
URL of javadoc	`https://symbiote-h2020.github.io/SemanticManager/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/SemanticManager`
Code coverage snapshot	55.40%

4.3.8.2 Feature History

Release	Main Features
0.2.0	• Initial implementation of resource and platform RDF generation
1.0.0	• Adding PIM handling: validation and storage. • Support for local caching of ontologies.
1.1.0	• Updates with respect to updated Libraries and information models
1.2.0	• Updates with respect to updated Libraries and information models
2.0.0	• Updates with respect to updated Libraries and information models

4.3.8.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`org.apache.jena`	`jena-core`	`3.4.0`
	`jena-querybuilder`	`3.4.0`
	`jena-cmds`	`3.4.0`
`org.mongodb`	`bson`	`3.4.2`
`com.github.symbiote-h2020`	`SymbIoTeSemantics`	`1.+`

4.3.8.4 Component Source Tree Information

The SemanticManager component includes following packages:

- `src/main/java/eu/h2020/symbiote/messaging`: classes responsible for RabbitMQ communciation.
- `src/main/java/eu/h2020/symbiote/ontology/errors`: contains RDF validation/translation exceptions definitions.
- `src/main/java/eu/h2020/symbiote/ontology/utils`: contains utility classes for RDF translation.
- `src/main/java/eu/h2020/symbiote/ontology/validation`: contains classes for RDF validation.

4.3.9 Core Bartering and Trading Manager

4.3.9.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/BarteringAndTrading`
URL of javadoc	`https://symbiote-h2020.github.io/BarteringAndTrading/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/BarteringAndTrading`
Code coverage snapshot	72.92%

4.3.9.2 Feature History

Release	Main Features
1.0.0	• Overseeing and providing support for Bartering Operations • Provide Trust Manager with Bartering information for trust calculation

4.3.9.3 Component Source Tree Information

The Core Bartering & Trading component shares the repository with the Bartering & Trading Manager component, since they share a lot of models and functionalities. Their interfaces are defined by Spring profiles.

The Bartering & Trading Manager is composed of the following repositories:

- `communication`: Classes for REST communication with other components.
- `config`: Configuration classes.
- `listeners`: Classes used for RabbitMQ and REST communication specific for the BTM and Core B&T.
- `repositories`: Classes used to store information regarding Coupons.
- `services`: Classes that contain services provided by these components.

4.3.10 Core Anomaly Detection

4.3.10.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/AnomalyDetectionModule`
URL of javadoc	`https://symbiote-h2020.github.io/AnomalyDetectionModule/doxygen`
URL of code coverage reports	`https://codecov.io/gh/symbiote-h2020/AnomalyDetectionModule`
Code coverage snapshot	79%

4.3.10.2 Feature History

Release	Main Features
1.0.0	• Handling reports of failed authorization within the federation • Providing statistics about platform misdeeds

4.3.10.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.asciidoctor	convert	1.5.3
org.asciidoctor	asciidoctorj-pdf	1.5.0-alpha.10.1
io.github.swagger2markup	swagger2markup-spring-restdocs-ext	1.2.0
io.github.swagger2markup	swagger2markup-gradle-plugin	1.2.0
org.springframework.cloud	spring-cloud-dependencies	Dalston.SR5
org.springframework.cloud	spring-cloud-starter	Dalston.SR5
org.springframework.cloud	spring-cloud-starter-config	Dalston.SR5
org.springframework.cloud	spring-cloud-starter-eureka	Dalston.SR5
org.springframework.cloud	spring-cloud-starter-zipkin	Dalston.SR5
org.springframework.boot	spring-boot-starter-amqp	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-web	1.5.14.RELEASE
javax.xml.bind	jaxb-api	2.3.0
io.swagger	swagger-annotations	1.5.16
com.github.symbiote-h2020	SymbIoTeSecurity	25.7.3
com.github.symbiote-h2020	SymbIoTeLibraries	5.15.1

4.3.10.4 Component Source Tree Information

The Core Anomaly Detection component includes the following packages:

- `src/main/java/eu/h2020/symbiote/security/communication`: contains communication interfaces
- `src/main/java/eu/h2020/symbiote/security/config`: configuration classes
- `src/main/java/eu/h2020/symbiote/security/listeners`: containing all the REST and AMQP interfaces and controllers/consumers.
- `src/main/java/eu/h2020/symbiote/security/repositories`: repositories definitions with payloads.
- `src/main/java/eu/h2020/symbiote/security/services`: all services responsible for all the module logic.

4.4 Platform Components

4.4.1 Federation Manager

4.4.1.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/FederationManager`
URL of javadoc	`https://symbiote-h2020.github.io/FederationManager/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/FederationManager`
Code coverage snapshot	82.14%

4.4.1.2 Feature History

Release	Main Features
2.0.0	• Receive and process federation and QoS updates from Administration • Validate federation information and access (security & business validation) • Distribute federation updates to relevant components within platform
3.0.0	• Maintain and generate federation history events for all federation activities • Aggregate federation history per platform for Trust management

4.4.1.3 Specific Dependencies

No specific dependencies apart from the common libraries listed in Section 4.1 used.

4.4.1.4 Component Source Tree Information

The Federation Manager component include the following subfolders:

- `src/main/java/eu/h2020/symbiote/fm/interfaces:` REST Controllers and RabbitMQ Connector classes.
- `src/main/java/eu/h2020/symbiote/fm/model:` Internal DTO classes for data storage.
- `src/main/java/eu/h2020/symbiote/fm/repositories:` DAO classes and central backend service.
- `src/main/java/eu/h2020/symbiote/fm/services:` Business logic services.

4.4.2 Monitoring

4.4.2.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/Monitoring`
URL of javadoc	`https://symbiote-h2020.github.io/Monitoring/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/Monitoring`
Code coverage snapshot	80.88%

4.4.2.2 Feature History

Release	Main Features
1.0.0	• Basic compatibility with Icinga2
2.0.0	• New implementation without Icinga2 • Metric gathering • Metric querying • Metric aggregation • RAP integration

4.4.2.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`org.mongodb`	`mongodb-driver`	`3.6.0`
`org.apache.commons`	`commons-lang3`	`3.4`
	`commons-colection4`	`4.1`

4.4.2.4 Component Source Tree Information

The Monitoring component include the following subfolders:

- `src/main/java/eu/h2020/symbiote/monitoring/beans`: Java beans used by the component.
- `src/main/java/eu/h2020/symbiote/monitoring/compat`: Compatibility classes used as helpers for MongoDB.
- `src/main/java/eu/h2020/symbiote/monitoring/constants`: Constants used in different classes of the component.
- `src/main/java/eu/h2020/symbiote/monitoring/db`: Classes to interact with the database backend.
- `src/main/java/eu/h2020/symbiote/monitoring/service`: REST and RabbitMQ facades to the database services.
- `src/main/java/eu/h2020/symbiote/monitoring/utils`: Utility classes.

4.4.3 Platform Registry

4.4.3.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/PlatformRegistry`
URL of javadoc	`https://symbiote-h2020.github.io/PlatformRegistry/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/PlatformRegistry`
Code coverage snapshot	58%

4.4.3.2 Feature History

Release	Main Features

1.0.0	• Interface definition
2.0.0	• Registration, update and delete of home resource metadata exposed to the federations. • Share and unshare resources to federations. • Basic search functionalities of federated resources.
3.0.0	• Handling of enhanced search requests (ranking-filtering)

4.4.3.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
`com.querydsl`	`querydsl-mongodb`	`4.1.4`
`com.querydsl`	`queydsl-apt`	`4.1.4`
	`com.ewerk.gradle.plugins.querydsl`	`1.0.9`

4.4.3.4 Component Source Tree Information

The PlatformRegistry component includes four main subfolders:

- `src/main/java/eu/h2020/symbiote/pr/communication`: the rabbit listeners for the registration and rest controller of the search service.
- `src/main/java/eu/h2020/symbiote/pr/repositories`: the resource repository.
- `src/main/java/eu/h2020/symbiote/pr/services:` the registration and search services.
- `src/main/java/eu/h2020/symbiote/pr/helpers:` helper methods for the authentication service.

4.4.4 Registration Handler

4.4.4.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/RegistrationHandler`
URL of javadoc	`https://symbiote-h2020.github.io/RegistrationHandler/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/RegistrationHandler`
Code coverage snapshot	65.07%

4.4.4.2 Feature History

Release	Main Features
1.0.0	• Basic L1 functionality
1.2.0	• Complete L1 functionality
2.0.0	• Complete L2 functionality

4.4.4.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.apache.commons	commons-collections	4.1

4.4.4.4 Component Source Tree Information

The Registration Handler component include the following subfolders:

- `src/main/java/eu/h2020/symbiote/rh/constants`: Constants used in different classes of the component.
- `src/main/java/eu/h2020/symbiote/rh/db`: Sprint data repositories for MongoDB.
- `src/main/java/eu/h2020/symbiote/rh/exceptions`: Custom exceptions for the REST interface.
- `src/main/java/eu/h2020/symbiote/rh/inforeader`: Plug-Ins to read resource metadata from different sources when the component loads.
- `src/main/java/eu/h2020/symbiote/rh/messaging`: RabbitMQ utilities to communicate changes to the rest of the components.
- `src/main/java/eu/h2020/symbiote/rh/service`: REST interface for resource metadata management.
- `src/main/java/eu/h2020/symbiote/rh/util`: Miscellaneous utilities used in the component.

4.4.5 Resource Access Proxy

4.4.5.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/ResourceAccessProxy
URL of javadoc	https://symbiote-h2020.github.io/ResourceAccessProxy/doxygen
URL of code coverage reports	https://codecov.io/github/symbiote-h2020/ResourceAccessProxy
Code coverage snapshot	18.51%

4.4.5.2 Feature History

Release	Main Features
1.0.0	• Support for REST and OData access to resources (for sensing and actuation) • Support for push mechanism via WebSockets • Support for custom Platform Information Models (described with *.owl* files) • Support for a subset of OData features (such as filters) • Mapping between symbIoTe global and platform internal IDs of resources • Security features using Security Handler component from Symbiote Libraries
2.0.0	• Support for multiple platform plugins • Support for accessing L2 resources • Checking federation access policies
3.0.0	• Support for accessing bartered L2 resources

4.4.5.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.springframework.boot	spring-boot-starter-data-rest	1.5.14.RELEASE
org.springframework.boot	spring-boot-starter-data-mongodb	1.5.14.RELEASE
org.springframework.boot	spring-amqp	2.0.0.M1
org.springframework.boot	spring-rabbit	2.0.0.M1
org.apache.olingo	odata-commons/odata-server	4.3.0
org.springframework.boot	spring-boot-starter-websocket	1.5.14.RELEASE
net.sourceforge.owlapi	owlapi-contract	5.1.1
net.sourceforge.owlapi	owlapi-util	3.3

4.4.5.4 Component Source Tree Information

The Resource Access Proxy component includes the packages below:

- `src/main/java/eu/h2020/symbiote/rap/exceptions`: contains custom exceptions
- `src/main/java/eu/h2020/symbiote/rap/managers`: responsible for the authentication and authorization of the receiving requests.
- `src/main/java/eu/h2020/symbiote/rap/interfaces`: contains classes responsible for implementing the external interfaces but OData (for plugin registration, for notifications, for interacting with CRAM and RH, for accessing to resources).
- `src/main/java/eu/h2020/symbiote/rap/bim`: contains classes for reading platform information model files (.owl).
- `src/main/java/eu/h2020/symbiote/rap/resources`: contains classes for static definitions, application configuration, mongo db and filters.
- `src/main/java/eu/h2020/symbiote/rap/messages`: contains classes for data models of the json messages exchanged with external modules.
- `src/main/java/eu/h2020/symbiote/rap/plugin`: contains classes for template RAP plugin.
- `src/main/java/eu/h2020/symbiote/rap/service`: contains classes for OData and WebSocket interface implementation.

4.4.6 Subscription Manager

4.4.6.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/SubscriptionManager
URL of javadoc	https://symbiote-h2020.github.io/SubscriptionManager/doxygen
URL of code coverage reports	https://codecov.io/github/symbiote-h2020/SubscriptionManager
Code coverage snapshot	72%

4.4.6.2 Feature History

Release	Main Features
1.0.0	• Interface definition
2.0.0	• Broadcast of shared (and unshared) resources to all federated platforms
3.0.0	• Enabled subscription definition to the platform owner • Broadcast of platform subscription to all federated platforms • Forwarding of shared (or unshared) resources to federated platforms depending on their subscription definitions

4.4.6.3 Component Source Tree Information

The SubscriptionManager component includes three main subfolders:

- `src/main/java/eu/h2020/symbiote/subman/controller`: REST interface and authentication helper classes.
- `src/main/java/eu/h2020/symbiote/subman/messaging`: Configuration of RabbitMQ and implementation of listeners for communication with PlatformRegistry and FederationManager components.
- `src/main/java/eu/h2020/symbiote/subman/repositories`: Mongo resource repositories.

4.4.7 Trust Manager

4.4.7.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/TrustManager`
URL of javadoc	`https://symbiote-h2020.github.io/TrustManager/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/TrustManager`
Code coverage snapshot	62.38%

4.4.7.2 Feature History

Release	Main Features
3.0.0	• Calculate (own) resource trust for offered resources with federated platforms • Calculate (foreign) platform reputation for federated platforms • Calculate (foreign) adaptive resource trust for shared resources from foreign platforms within the federation

4.4.7.3 Specific Dependencies

No specific dependencies apart from the common libraries listed in Section 4.1 used.

4.4.7.4 Component Source Tree Information

The Trust Manager component include the following subfolders:

- `src/main/java/eu/h2020/symbiote/tm/interfaces`: REST and RabbitMQ Connector classes.

- `src/main/java/eu/h2020/symbiote/tm/repositories`: DAO class to access MongoDB repository.
- `src/main/java/eu/h2020/symbiote/tm/services`: Business logic services.
- `src/main/java/eu/h2020/symbiote/tm/cron`: Scheduled tasks for trust/reputation update.

4.4.8 Bartering & Trading Manager

4.4.8.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/BarteringAndTrading`
URL of javadoc	`https://symbiote-h2020.github.io/BarteringAndTrading/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/BarteringAndTrading`
Code coverage snapshot	72.92%

4.4.8.2 Feature History

Release	Main Features
1.0.0	• Providing Bartering mechanisms for bartering of resources between federated platforms

4.4.8.3 Component Source Tree Information

The Bartering & Trading Manager component shares the repository with the Core Bartering & Trading component, since they share a lot of models and functionalities. Their interfaces are defined by Spring profiles.

The Bartering & Trading Manager is composed by the following repositories:

- `communication`: Classes for REST communication with other components.
- `config`: Configuration classes.
- `listeners`: Classes used for RabbitMQ and REST communication specific for the BTM and Core B&T.
- `repositories`: Classes used to store information regarding Coupons.
- `services`: Classes that contain services provided by these components.

4.4.9 SLA Manager

4.4.9.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/SLAManager`
URL of javadoc	`https://symbiote-h2020.github.io/SLAManager/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/SLAManager`
Code coverage snapshot	`https://codecov.io/github/symbiote-h2020/SLAManager`

4.4.9.2 Feature History

Release	Main Features
1.0.0	• L2 functionality

4.4.9.3 Specific Dependencies

Contrary to the rest of the components, SLA Manager is not a Spring Boot application but a Spring one and the build system is based on Maven instead of Gradle. As such, we don't provide here the difference in dependencies since most of them are different to the rest and it would mean to list all of the project's dependencies.

4.4.9.4 Component Source Tree Information

SLA Manager is composed of several sub components:

- `sla-common`: Common classes used by the rest of the sub components.
- `sla-enforcement`: Sub-component dedicated to the evaluation and enforcement of SLAs.
- `sla-personalization`: Specific classes related to SymbIoTe which include adapters from JSON to WS-Agreement and RabbitMQ communication.
- `sla-repository`: JPA classes and utilities to persist SLAs into MySQL.
- `sla-service`: Web Services interface based on WS-Agreement.
- `sla-tools`: Miscellaneous utilities and tools used in the rest of the sub-components.
- `sla-swag-model`: Beans based on the WS-Agreement data model.

4.5 Smart Space Components

4.5.1 Smart Space Middleware

4.5.1.1 Generic Information

URL of git repository	`https://github.com/symbiote-h2020/symbioteSmartSpace`
URL of javadoc	`https://symbiote-h2020.github.io/symbioteSmartSpace/doxygen`
URL of code coverage reports	`https://codecov.io/github/symbiote-h2020/symbioteSmartSpace`
Code Coverage Snapshot	9.6 %

4.5.1.2 Feature History

Release	Main Features
1.0.0	• Registration/Unregistration of L3 resources • Access to L3 resources • Keep alive messages
2.0.0	• Added lightweight security protocol • Added interfaces • Added Core communication for L3/L4 resources

4.5.1.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
org.mongodb	mongodb-driver	3.4.2
org.apache.commons	commons-lang3	3.3.2

4.5.1.4 Component Source Tree Information

The Smart Space Middleware application includes a set of packages:

- `src/main/java/eu/h2020/symbiote/innkeper`: Innkeeper component related classes (for REST controller, data models, services).
- `src/main/java/eu/h2020/symbiote/rap`: SSP RAP component related classes (for REST / OData controllers, data models, push service.
- `src/main/java/eu/h2020/symbiote/lwsp`: LightWeight Security Protocol implementation classes.
- `src/main/java/eu/h2020/symbiote/resources`: data models and service classes for MongoDB.

4.5.2 SDEV SymbIoTe Agent

4.5.2.1 Generic Information

URL of git repository	https://github.com/symbiote-h2020/SymbioteSmartSpace/tree/sym-agent/sym-agent
URL of javadoc	https://symbiote-h2020.github.io/symbioteSmartSpace/doxygen
URL of code coverage reports	https://codecov.io/github/symbiote-h2020/symbioteSmartSpace
Code coverage snapshot	--

4.5.2.2 Feature History

Release	Main Features
1.0.0	• Basic interfaces (registration, unregistration, keep alive, etc.) without security
2.0.0	• Added lightweight security protocol • Added interfaces • Added semantic class • Added NTP server synch for timestamp value reading

4.5.2.3 Specific Dependencies

GroupId	Artifact/Plugin	Version
ArduinoJson	bblanchon/ArduinoJson	v5.11.1-1-g729bf0a
RestClient	DaKaZ/esp8266-restclient	3.3.2
Cypto	intrbiz/arduino-crypto	commit: *7943d
Hash	--	2.4.0

sha1	bbx10/Cryptosuite	commit: *48669
base64	Densaugeo/base64_arduino	1.1.0
ESP8266WebServer	--	2.4.0
WiFiUdp	--	2.4.0
NTPClient	arduino-libraries/NTPClient	3.1.0

4.5.2.4 Component Source Tree Information

SDEV agent is a C++ library for the ESP8266 Arduino platform. It is composed of the following sub components:

- `libraries/` contains:
  - `lsp/` the lightweight security protocol library that is automatically included from the agent library.
  - `semantic_resources/` that contains the semantic handling class for resource mapping into the symbIoTe ontology.
- `src/` contains the agent library and a helper header file (symbiote_resources.h) from which you have a list of validated symbIoTe resource type.
- `test-sketch/` contains a list of example sketches[20] divided by platform specific type (e.g. commercial available board like Huzzah, Wemos, etc.) and library test class.

[20]Arduino C++ code is called sketch.

5 Developed Applications

5.1 Applications Developed for symbIoTe Use Cases

SymbIoTe uses a set of use cases to validate the implemented interoperability concepts. Each use case uses and accomplishes chains of information flows starting from the sensors through different stages like enablers up to the applications. Detailed descriptions of each use case can be found in Deliverable 1.3 [5].

The use cases were chosen to drive the development of symbIoTe and thus are now testing as many different aspects of the symbIoTe framework as possible. All use cases interact with users and needed Human-Machine-Interfaces, usually a GUI in the form of an application, to interact with the underlying logic. In the following sections these applications are described in more detail.

5.1.1 Smart Residence

The following applications that offer comfort, automation, security, energy efficiency and healthcare services have been implemented within the Smart Residence use case:

- Smart Healthy Indoor Air: This application is based on the indoor/outdoor air quality monitoring with the aim to improve indoor air quality by giving recommendations and alerts.
- Smart Area Controller: This application is related to the Dynamic Interface Adaptation scenario, where the users can control different devices according to the controllable room in range.
- Home Comfort: this application demonstrates the Energy Saving scenario, which shows how to automatically control home devices, in order to keep environmental parameters (e.g. light, temperature, humidity, etc.) at some predefined comfort values.
- Smart Health Mirror: It is an Ambient Assisted Living (AAL) application to help people, in particular elderly people, to live independently for longer.

5.1.1.1 Smart Healthy Indoor Air

This application is based on the indoor/outdoor air quality monitoring and pursues to improve indoor air quality. Indoor air quality (IAQ) refers to the quality of the air inside buildings as represented by concentrations of pollutants and thermal (temperature and relative humidity) conditions that affect the health, comfort and performance of occupants. It is important to ensure that the air inside the building people inhabit on a daily basis is of a good quality. Outdoor generated air pollution is relevant for indoor air quality and health. Exposure to indoor air pollution has been linked to the development of different diseases from infections to asthma or to poor sleep. It can also cause less serious side effects such as headaches, dry eyes and nasal congestion[21].

[21] Quantifying the Performance of Natural Ventilation Windcatchers. Jones, B; (2010) Quantifying the Performance of Natural Ventilation Windcatchers. Doctoral thesis , Brunel University

Sensing and Control Systems SL partner's (S&C) roadmap aims to create a smart home/office connected with the city. The current S&C's platform, nAssist[22], monitors and controls a number of direct parameters related to indoor air quality, such as CO_2 levels, humidity and temperature. In addition, this platform monitors and controls other factors that are important for indoor environmental quality considerations such as light and noise since they also affect occupants.

The idea is to improve this framework to understand how indoor and outdoor sources of pollution, heat and humidity, together with the ventilation and air conditioning systems, affect the indoor air quality in buildings. It also begins to address methods of controlling those factors in order to improve quality of the indoor air for occupants' health, comfort and performance. To achieve this goal, the smart home retrieves outdoor air quality data received from other platforms. Such data can include air pollution levels. The smart home reacts to changes in indoor parameters such as temperature, humidity, CO_2 levels and noise to maintain a healthy and safe indoor environment by recommending actions to the user such as using air purifiers, ventilation systems and opening/closing the windows to eliminate unpleasant impacts. S&C aims to provide more robust solutions with focus on clean environment and optimised energy use.

In particular, the project has developed an application capable of monitoring real-time indoor and outdoor air quality information. Without this information, usually we ventilate late and too long. Smart Healthy Indoor Air indicates when a room should be ventilated taking into consideration that windows are the easiest ventilation option but not the healthiest one depending on the outdoor air quality. There are other ways to provide healthy flow of air throughout the room, such as turning on the air conditioner, or individual air purifiers. The automatic control of some devices, as air purifiers, ventilation and air conditioning systems are out of the scope of this application.

5.1.1.1.1. Design

Outdoor air quality data is provided by a Public Service offered by the Generalitat de Catalunya[23] but it could be provided by other federated platforms dedicated to offering Smart Cities' services. Also, given the limited number of monitoring stations available and placed at representative spots to record the outdoor air quality, an accurate assessment of spatial variation is highly required. Spatial interpolation techniques applied to the available monitoring data to provide air quality information closest to the location of the smart home are used. This functionality is provided by the component named as Interpolator, which is also used in the smart mobility use case. This showcases how symbIoTe enables the realization of high level services, involving different IoT platforms, and offers their results to different (end-user) applications. The application sends the GPS location of the smart home and gets the estimated value about the air quality for this specific location from symbIoTe, in particular from the interpolator. This is the main benefit of using symbIoTe. The project can offer a more robust and precise application without developing new functionalities (Figure 4).

[22] http://www.sensingcontrol.com/solutions/customizable-iot-platform.html

[23] http://dtes.gencat.cat/icqa/start.do?lang=en

On the other hand, the S&C's platform, nAssist, makes available to symbIoTe data related to the indoor air quality at home: temperature, humidity, CO levels, and luminosity. As already explained, exposure to indoor air pollution has been linked to the development of everything from infections to asthma to poor sleep. It can also cause less serious side effects such as headaches, dry eyes and nasal congestion. This information can be helpful for remote healthcare applications.

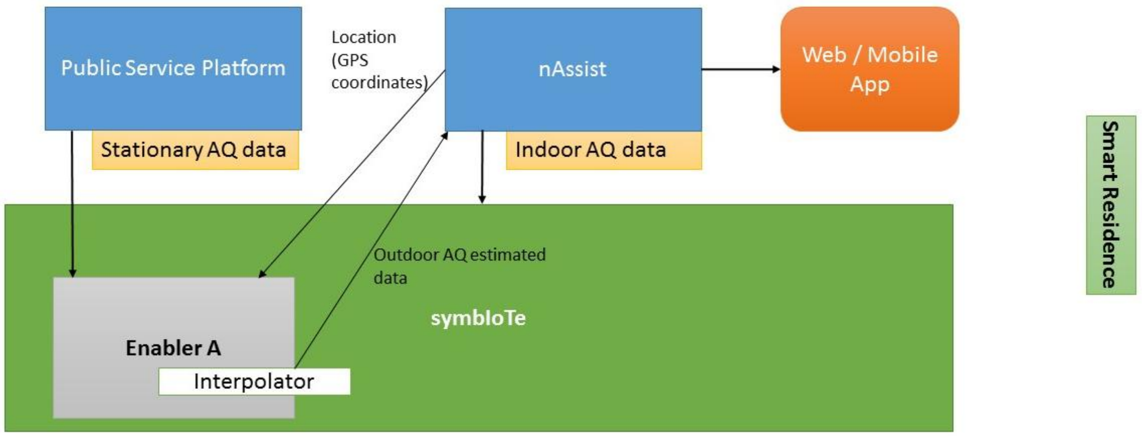


Figure 4: High-level architecture showing the involved platforms, applications and involved symbIoTe components (e.g., Enabler) for Smart Healthy Indoor Air application

5.1.1.1.2 Compliance Level

The S&C's platform nAssist (symbIoTe L1-compliant) acquires, stores and processes all data that measure the indoor and outdoor air quality. Temperature, humidity, CO, NO2, O3, PM10, and luminosity levels data are published within symbIoTe.

5.1.1.1.3 Platform

The service can be accessed through a mobile app (iOS and Android) or the website.

5.1.1.1.4 User Interaction

The GUI design for the app was based on the current S&C's product, enControl[24]. The application has GUI based on web and smartphone iOS and Android (Figure 5 - Figure 8).

[24]http://www.encontrol.io/

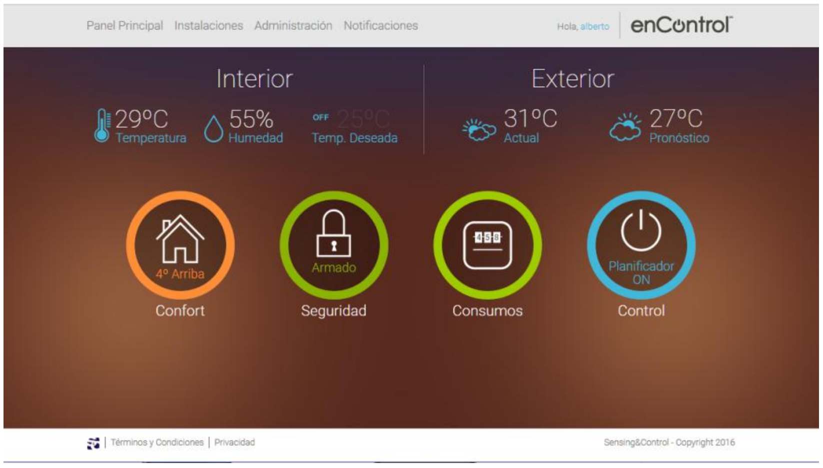


Figure 5: Main screen for all functionalities provided by enControl: comfort, security, energy consumption and automatic control of devices

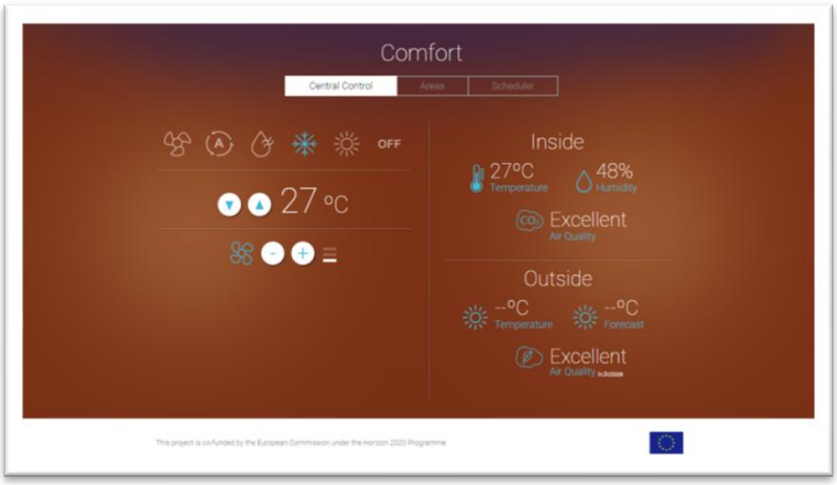


Figure 6: Main screen for comfort with the indoor and outdoor air quality levels

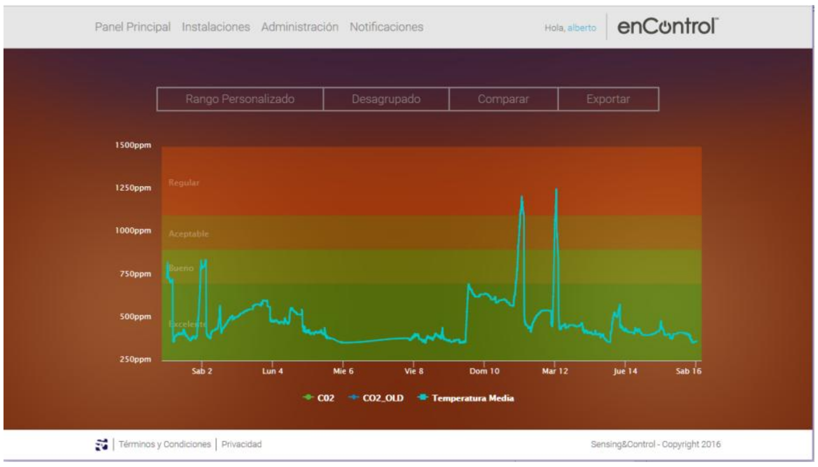


Figure 7: Indoor air quality values

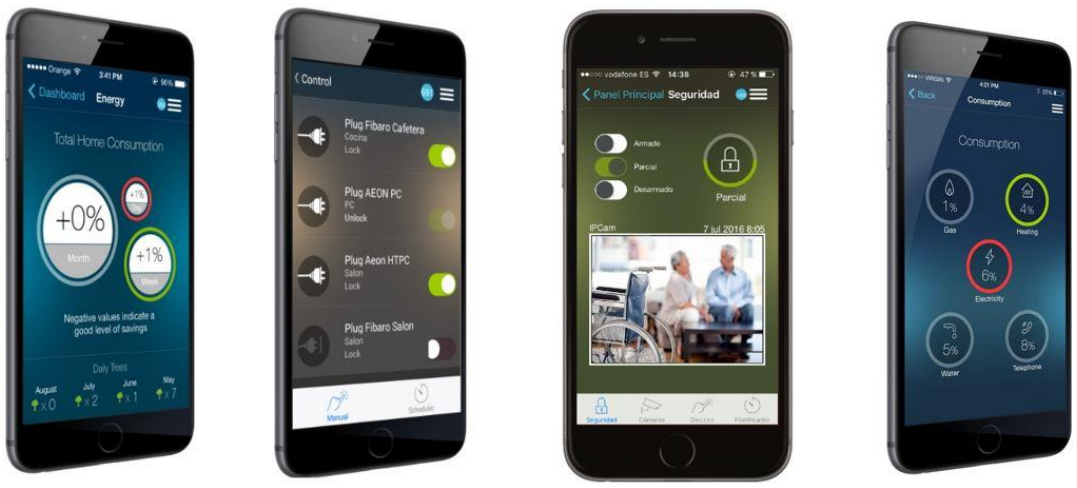


Figure 8: Examples of GUI for iPhone

5.1.1.1.5 Implementation

The application involves the implementation of three main different functionalities:

- Monitoring real-time indoor air quality and providing recommendations about when should ventilate the home by using the smart home system. The following figures show how the application interprets the levels of both indoor and outdoor air quality and the recommendations to improve the indoor air quality by taking some actions:

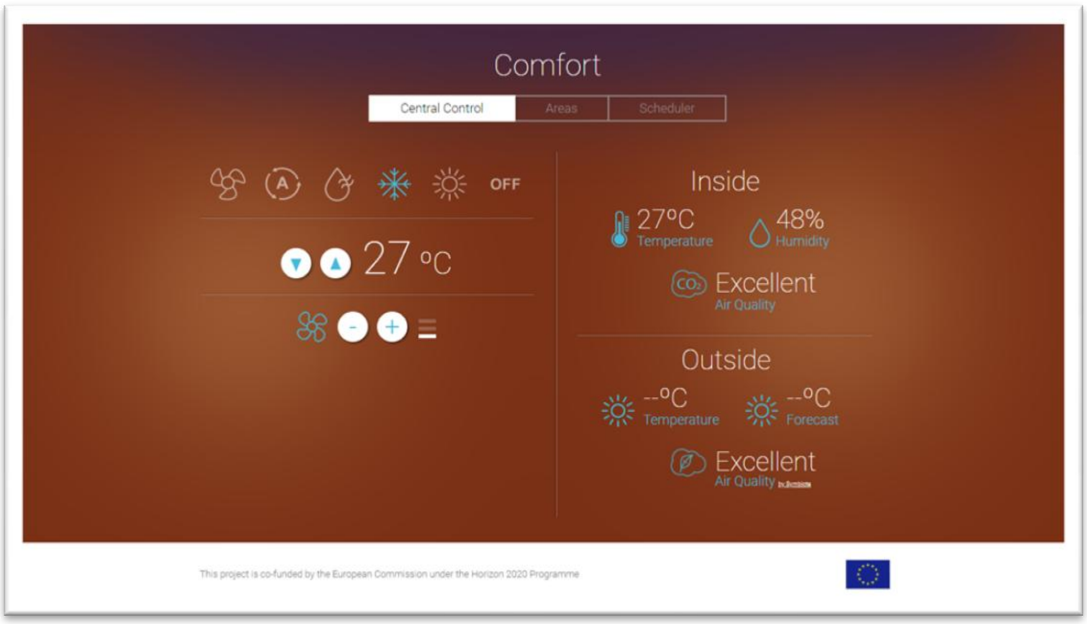
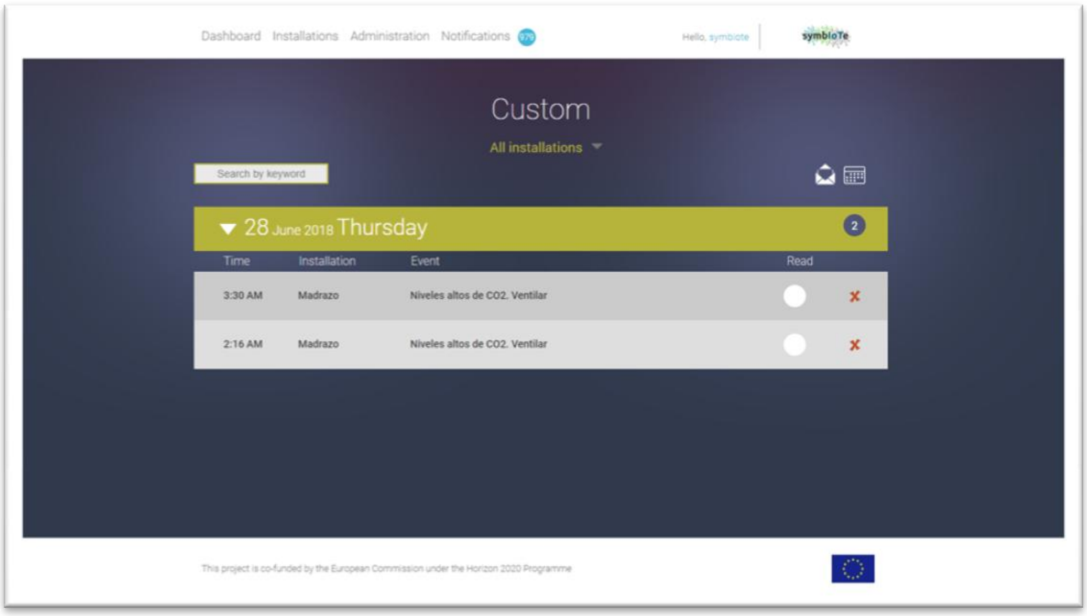


Figure 9: Indoor and outdoor air quality



Custom

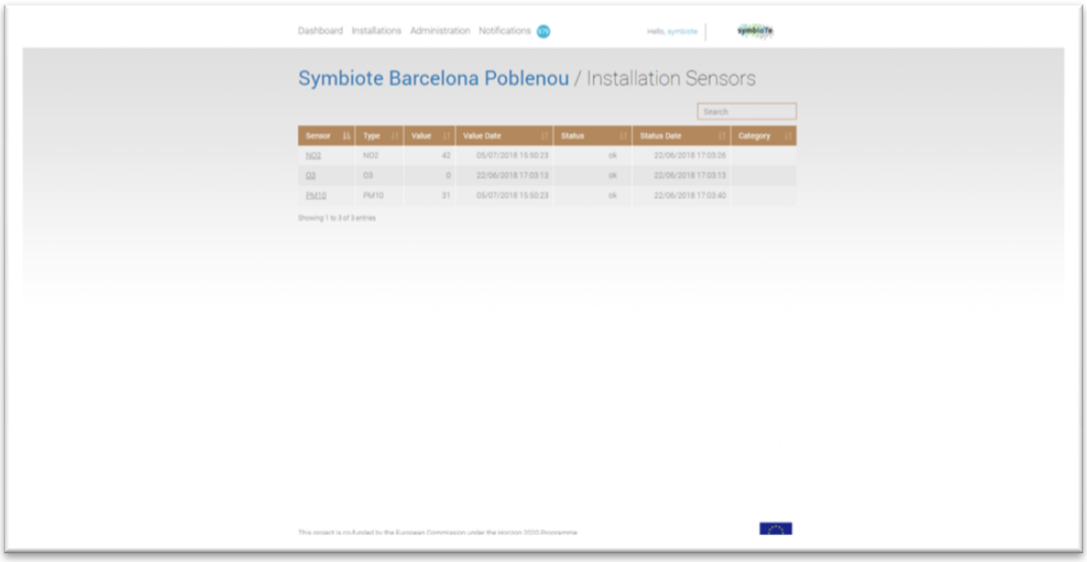
All installations

28 June 2018 Thursday 2

Time	Installation	Event	Read
3:30 AM	Madrazo	Niveles altos de CO2. Ventilar	
2:16 AM	Madrazo	Niveles altos de CO2. Ventilar	

Figure 10: Recommendations to improve the indoor air quality

- Acquisition of outdoor air quality data from The Atmospheric Pollution Vigilance and Forecast Network[25] (the XVPCA) from the Ministry for Territory and Sustainability at the Generalitat of Catalonia. Outdoor air quality data could be provided by other federated platforms dedicated to offering Smart Cities' services but not in this trial. The application shows the values of the O3, NO2 and PM10, as shown in the following figure:


Dashboard
Installations
Administration
Notifications
Hello, symbiote
symbIoTe
Search
This project is co-funded by the European Commission under the Horizon 2020 Programme

Symbiote Barcelona Poblenou / Installation Sensors

Sensor	Type	Value	Value Date	Status	Status Date	Category
NO2	NO2	42	05/07/2018 15:50:23	ok	22/06/2018 17:03:26	
O3	O3	0	22/06/2018 17:03:13	ok	22/06/2018 17:03:13	
PM10	PM10	31	05/07/2018 15:50:23	ok	22/06/2018 17:03:40	

Showing 1 to 3 of 3 entries

Figure 11: Visualization of air quality parameters provided by the outdoor stations

[25] http://dtes.gencat.cat/icqa/start.do?lang=en

- Interpolation of the outdoor air quality values taking into consideration the location of the smart home used provided by the module Enabler Logic from symbIoTe.

5.1.1.1.6 Initial Functional Tests

The initial functional tests were done with 5 home installations located in Barcelona. The outdoor air quality data acquisition is provided by a Public Service of the Generalitat of Catalonia.

5.1.1.2 Smart Area Controller

The mobile application will be capable of controlling CPS devices located in a room, selected by the user, according to his/her needs. The main interface will allow to navigate the structure (building, floor, room, etc.) for selecting the device to control (registered by the platforms that reside in that space). For example, the user will be able to change the temperature in the room controlled by a Netatmo thermostat of a Platform A, or to move the curtains and change the luminance level controlled in Platform B.

The application will leverage on a specific enabler, which gives the possibility to filter the devices in the space, based on their position (building, floor, room, etc.). In this way, the application has to query the symbIoTe Enabler in order to retrieve the list of the CPSs in the selected area and then allow the user to control them.

5.1.1.2.1 Design

Figure 12 shows a high-level diagram of the application communicating with the symbIoTe Enabler and the symbIoTe compliant platforms involved in the use case.

The platforms register their devices specifying information with respect to their indoor locations; the Administration console, accessible from the Enabler, allows the SSP administrator to manage the hierarchy of the resources' locations. Lastly, the mobile application is the user entry-point to interact with platform devices. In this specific use case scenario, no Web application is required.

The main benefit of using symbIoTe is the possibility to use a single application for dynamically controlling the different devices and platforms present in the house.

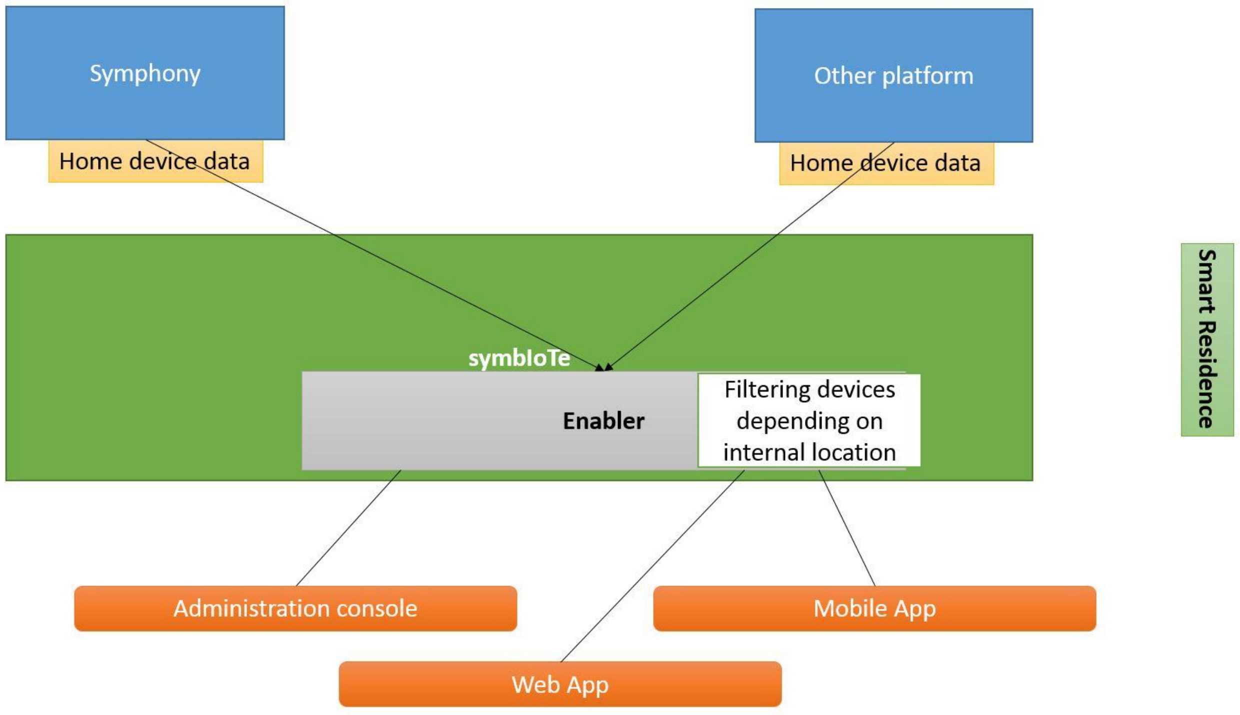


Figure 12: High-level architecture showing the involved platforms, applications and symbIoTe components (e.g., Enabler) for the Smart Area Controller and Home Comfort applications

5.1.1.2.2 Compliance Level

The application is L3 compliant according to the Smart Space definitions and features.

5.1.1.2.3 Platform

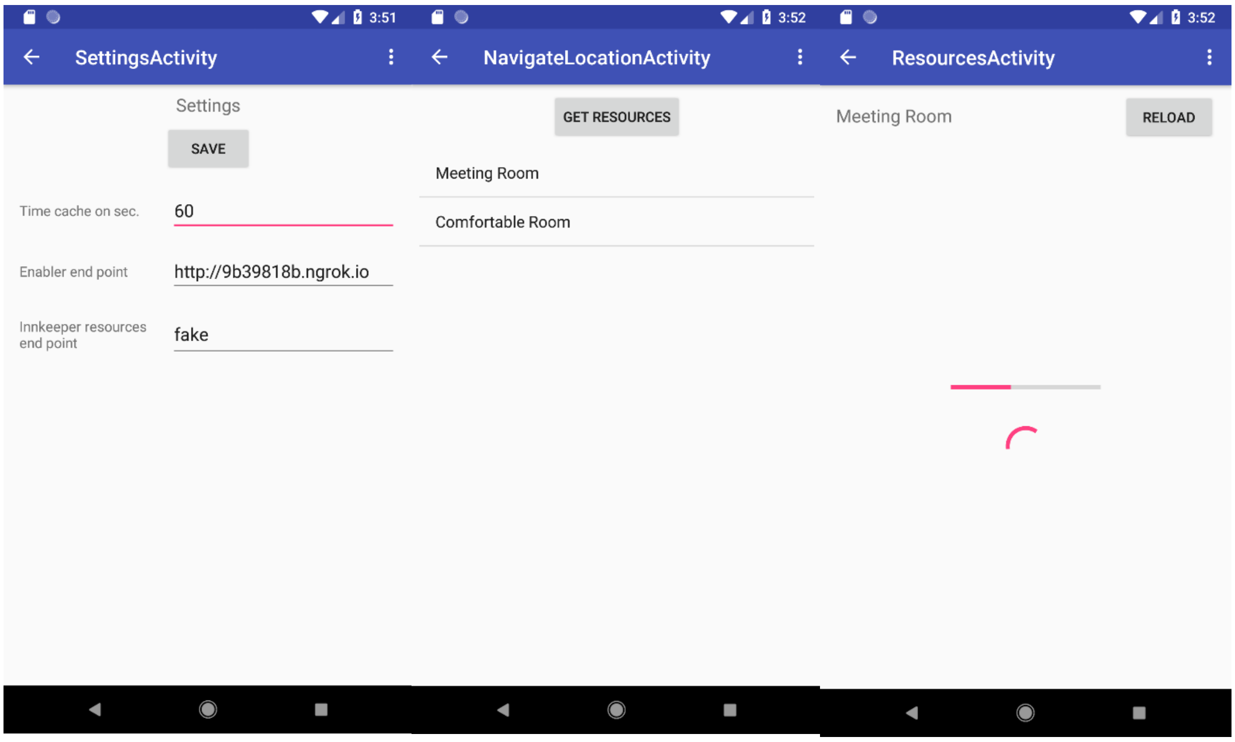
The Smart Area application is a mobile application compatible with Android platform.

5.1.1.2.4 User Interaction

First of all, the user will select the area in which she/he desires to control the devices (Figure 13 and Figure 14): it could be a room, a flat, the entire building or even a desk, according to the configuration made in the service enabler. Afterwards, the interactions will be related to the devices present and may vary according to the nature of the CPSs themselves.

5.1.1.2.5 Implementation

The language used for implementation is Java, since it depends on the mobile platform of the application (Android). The application is specific for this scenario purposes, but it can also be used for every use case which needs a direct control of local devices. At the time of writing this document, the implementation of the application has been completed and it is currently under testing. All the developments are based on symbIoTe version 2.0.0.



Settings

Time cache on sec. 60

Enabler end point http://9b39818b.ngrok.io

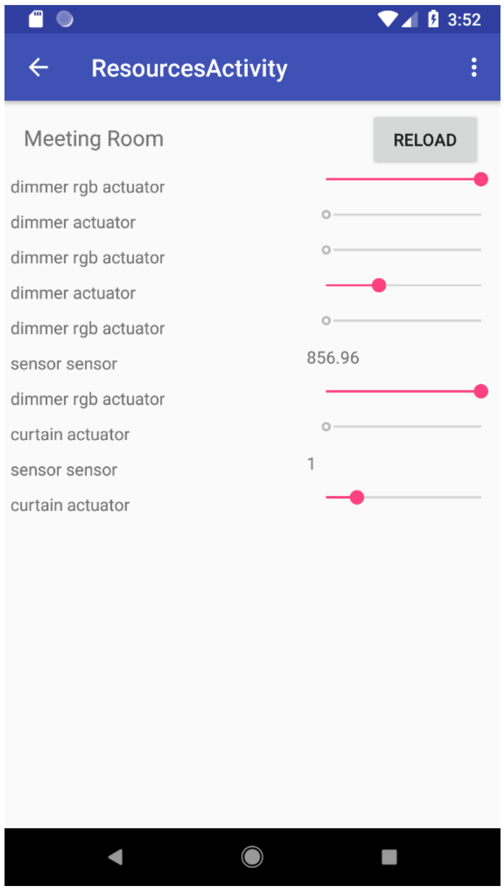
Innkeeper resources end point fake

Meeting Room

Comfortable Room

Meeting Room

Figure 13: From left to the right: (1) Settings; (2) Navigation in the hierarchy of locations; (3) Loading before getting the resources



Meeting Room

dimmer rgb actuator
dimmer actuator
dimmer rgb actuator
dimmer actuator
dimmer rgb actuator
sensor sensor 856.96
dimmer rgb actuator
curtain actuator
sensor sensor 1
curtain actuator

Figure 14: Resource with which to interact regarding the selected location

5.1.1.2.6 Initial Functional Tests

The initial functional tests have been performed with the current installations located in Pisa. Different devices (some lights, rgb lights, curtain, presence sensor and luminosity sensors) located in three different room situated in two different floors have been used.

5.1.1.3 Home Comfort

The L3 compliant application will be used to automatically control home devices in order to keep comfort values for home environmental parameters like temperature, luminosity, etc. The application is composed by both a backend and a frontend part: the first one manages the core operations, by constantly monitoring the surrounding and controlling devices in order to reach a desired comfort state; the second one acts as a configurator of comfort set-points.

Consequently, the backend of Home Comfort application run on a server having a frontend accessible via web for configuration purposes.

5.1.1.3.1 Design

As described in Section 5.1.1.1, the platforms register their devices by specifying information with respect to their indoor locations, so the SSP administrator configures the hierarchy of the resources' locations through the administration console. The Web Application allows the resident to configure all the set-points for the home devices through a graphical user interface (for example turn on light in a room when someone is present).

The main benefit of using symbIoTe is the possibility to leverage on the interoperability between the various platforms present in the house for driving the environment towards a comfort state.

5.1.1.3.2 Compliance Level

The application will be L3 compliant, according to the Smart Space definitions and features.

5.1.1.3.3 Platform

The backend application will be Linux based and the frontend can be accessed through a web page.

5.1.1.3.4 User Interaction

The core of the application is a service which does not need any interaction from the user apart from the configuration of the comfort set-points.

5.1.1.3.5 Implementation

The language used for the implementation of the backend is Java and of the frontend is HTML and Javascript.

At the time of writing this document, the implementation of the application is almost ready to move over to the testing phase. All the developments are based on symbIoTe version 2.0.0.

5.1.1.3.6 Initial Functional Tests

The initial functional tests are performed with the current installations located in Pisa.

5.1.1.4 Smart Health Mirror

5.1.1.4.1 Design

The smart health mirror SMILA (Smart Mirror Integrated Living Assistant) is an interactive voice-driven mirror assisting elderly people in managing their health. It was specifically designed for people suffering from chronic heart failure where regular measurement of vital parameters (e.g., weight or blood pressure) are necessary for treatment. SMILA was constructed of a wooden frame (40x30x4cm, gross weight 2.5kg) housing the semi-transparent mirror hiding a Samsung Galaxy Tab A 10.1 Android tablet. By using off the shelf materials and simple constructions the total costs for one smart mirror are around 300€.

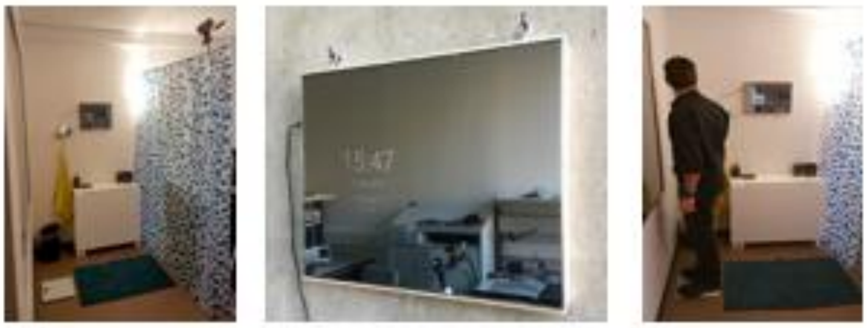


Figure 15: From left to right: (1) Living lab evaluation scenario; (2) SMILA and (3) user interacting with the device

5.1.1.4.2 Compliance level

The smart mirror app is connected to the KIOLA eHealth platform which is L1-compliant.

5.1.1.4.3 Platform

The KIOLA eHealth platform was developed on Python and the web framework Django. The smart mirror app was developed on top of the Google Android operating system.

5.1.1.4.4 User interaction

The overall workflow can be described as follows: The user enters the bathroom and SMILA detects the presence of a wristband. It queries the symbIoTe Core using the ID of the wristband and gains access to a symbIoTe-enabled eHealth platform KIOLA. The eHealth platform is then accessed directly through the reverse access proxy provided by the symbIoTe Core. Subsequently, SMILA gathers data, such as sensor data and personal information about the user. Using a health measurement profile, SMILA then asks the user to perform a weight measurement, asks questions related to personal wellbeing and finally transmits the collected data using the reverse access proxy to the KIOLA platform.

5.1.1.4.5 Implementation

SMILA is powered by an Android-based, symbIoTe-enabled app responsible for managing devices and user interaction. In idle mode the mirror displays the current time and the weather situation at its current location. It was decided against showing more information for several reasons: (1) to prevent information overload and only provide context-relevant information and (2) to keep battery and bandwidth usage low. As we chose the setting of cardiovascular diseases, a number of vital parameters are interesting with respect to therapy: regular measurements of heart rate, blood pressure, body weight and daily activity are relevant to the therapy in order to assess effectiveness as well as to detect any deterioration in a person’s health status. For initial evaluation we chose connecting the smart mirror to a Bluetooth-enabled scale for two reasons. (1) a scale is most likely to be found in bathrooms and (2) a sudden increase in weight (> 1 kg / per night, > 2kg/ over three days, > 2,5 kg/ per week) might indicate a severe deterioration of the overall health status. Moreover, SMILA uses voice input to collect information on personal well-being using Google’s Cloud Speech API. Moreover, we considered several options for user identification: (1) identification and authorization by a separate device using PIN codes once the device is close to the mirror (2) facial recognition and (3) wearable Bluetooth low energy beacons for identification. Bluetooth low energy (BLE) beacons are devices transmitting signals containing their ID along with other technical information on a regular interval. Devices such as smartphones or tablets can identify these radio signals within a limited range and apps can react to the presence of such beacons. We favoured this approach especially to biometric identification as it is (1) more privacy-preserving and (2) BLE beacons are in general built into fitness wristbands, thus combining identification with reading further health measurements. BLE beacons have been used in various scenarios, most prominently in e-Commerce settings offering consumers guide within shops. For our initial evaluation we decided to use wristband type beacons or devices that can be attached to a keychain (D15 UFO Bluetooth). The latter can be incorporated in a necklace as well. Apart from BLE wristbands, support for fitness trackers (Fitbit Ionic smart watch) was implemented as mode of user identification. If users wear a fitness tracker instead of a wristband, additional data is displayed on the mirror with respect to daily activity.

5.1.1.4.6 Initial Functional Tests

Initial functional tests were conducted in two living lab trials: one was conducted in with students of the University of Vienna and a second trial was conducted to test the smart mirror with elderly people.

5.1.2 Smart Mobility and Ecological Routing

The Smart Mobility and Ecological Routing Use Case addresses the problems regarding environment pollution and air quality in the major European cities. It does so by collecting air quality data from multiple IoT platforms in different countries and uses such measurements for runners, joggers and cyclists to plan the best routes to their destination.

Through symbIoTe, air quality measurement are obtained from different platforms. Due to the nature of routing algorithms, these measurements need substantial pre-processing with the purpose of associating air measurements to the map’s street segments.

Having streets correctly classified by their air quality, routing engines can take that information into account when computing the most ecological routes for the application’s users. Route calculation can also benefit from other factors such as traffic density and available parking spaces for bikes, in case the platforms have access to this kind of information.

Finally, users should be able to search for Points of Interest (POIs) following certain criteria. Routes for the selected POI can be computed using the previously mentioned routing service.

All in all, this use case showcases platform interoperability within the application and cloud domain. More details can be found in Section 6.4 of Deliverable D1.3 [5].

There are three platforms providing services and data to the use case:

- OpenIoT from UNIZG-FER provides air quality data from users' wearables,
- openUWEDAT from AIT provides air quality data from stationary sensors and a routing service for the city of Vienna,
- MoBaaS (Mobility Backend as a Service) from Ubiwhere provides their routing service.

Additionally, the OpenStreetMap[26] service is used to obtain cities' POIs.

The mobile application (symbIoTe SMEUR[27]) aims at delivering to users ecological green routes to their destinations. These routes direct the user to their destination, avoiding highly polluted areas. Additionally, the application also provides the user with the ability to search for POIs and, subsequently, obtain a route to the selected POI.

As such, there are two main functionalities that the application should provide to the user:

- Computation of ecological green routes
- POI search.

The application will access these services through the Smart Mobility and Ecological Routing Enabler, which handles the exchange of data and services between the platforms involved in the use case. As such, the application communicates with the Enabler to provide the routing and POI Search services for the user.

Users of the application are presented with a map after logging in. Users can then choose an origin and destination point for their desired route and a preferred means of transportation and are presented with the best ecological route computed.

Users can also request POIs, select their filtering preferences from a range of possible criteria, such as the type of POI, distance to a certain location, etc. POIs matching the users' criteria are presented. Users can then choose and be presented with an ecological route to the selected POI.

[26] https://www.openstreetmap.org/

[27] https://play.google.com/store/apps/details?id=com.ubiwhere.symbiote&hl=en_GB

5.1.2.1 Design

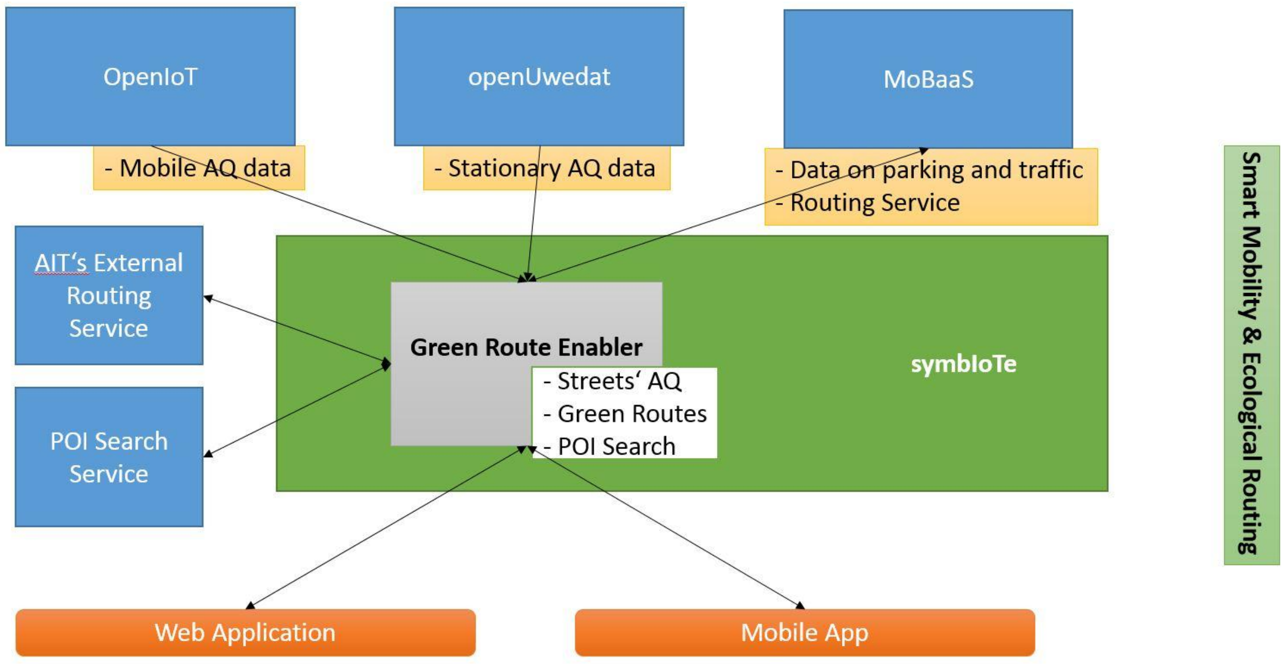


Figure 16: High-level architecture showing the involved platforms, applications and involved symbIoTe components (e.g., enabler) for Smart Mobility and Ecological Routing applications

As can be seen in Figure 16, the Green Route Enabler orchestrates the activity in the use case. It obtains air quality data from the platforms and interpolate it with the street segments of the map being used in order to obtain the air quality of a given street. These data are provided to the routing services (either the ones residing within a platform or external services) which, combined with other data such as traffic or parking, will compute green routes. Additionally, the data provided by the platforms can also be used to obtain POIs of interest to users.

It is clear from the figure how symbIoTe is relevant to this use case. It shows how developers using symbIoTe can use different data from different platforms from different domains easily. This is very advantageous in the development process, helping developers create complex systems using various sources of data. In the context of smart cities, it will also show how advantageous it is for platform owners and cities to provide their data through the symbIoTe ecosystem, allowing developers/organizations to easily create valuable services to citizens.

5.1.2.2 Compliance Level

The application interfaces with the developed Enabler, which complies with symbIoTe's L1.

5.1.2.3 Platform

The application will be developed for the Android mobile operating system.

5.1.2.4 User Interaction

The user primary means of interacting with the application is through the map, where it can set, for example, start and end points of his destination or the area near which the user is looking for POIs. It is also be through this map that the user is able to see the results of the requests, being the route to the destination or the POIs return from the search. Figure 1Figure 17 shows a screenshot of the mobile application.

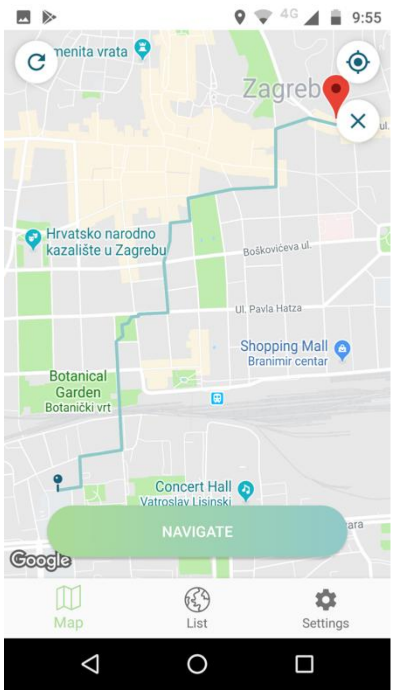

9:55
Zagreb
Hrvatsko narodno kazalište u Zagrebu
Boškovićeva ul.
Ul. Pavla Hatza
Shopping Mall Branimir centar
Botanical Garden Botanički vrt
Concert Hall Vatroslav Lisinski
NAVIGATE
Google
Map
List
Settings

Figure 17: Smart mobility mobile App

5.1.2.5 Implementation

The core of the mobile application is based on the Ionic framework[28], a free and open source mobile SDK. Ionic is, in turn, built on top of AngularJS[29] and Apache Cordova[30].

Currently, a first version of the application has been developed, where it is already possible for users to obtain routes and search for POIs using the symbIoTe enabler.

5.1.2.6 Initial Functional Tests

It is expected that the use case will run trials in at least three European cities. Each trial will have at least 20 users, for a period of 30 days with different types of end-users which will actively use the ecological urban routing application and in parallel will contribute with air quality and traffic data.

[28]https://ionicframework.com

[29] https://angularjs.org

[30] https://cordova.apache.org

5.1.3 EduCampus

5.1.3.1 User Interaction

The IOSB Mobile application provides a user interface to retrieve room information and to place room reservations. The application scans for BLE beacon signals and connects to the IOSB backend server.

The figure below shows some screenshots of the application.

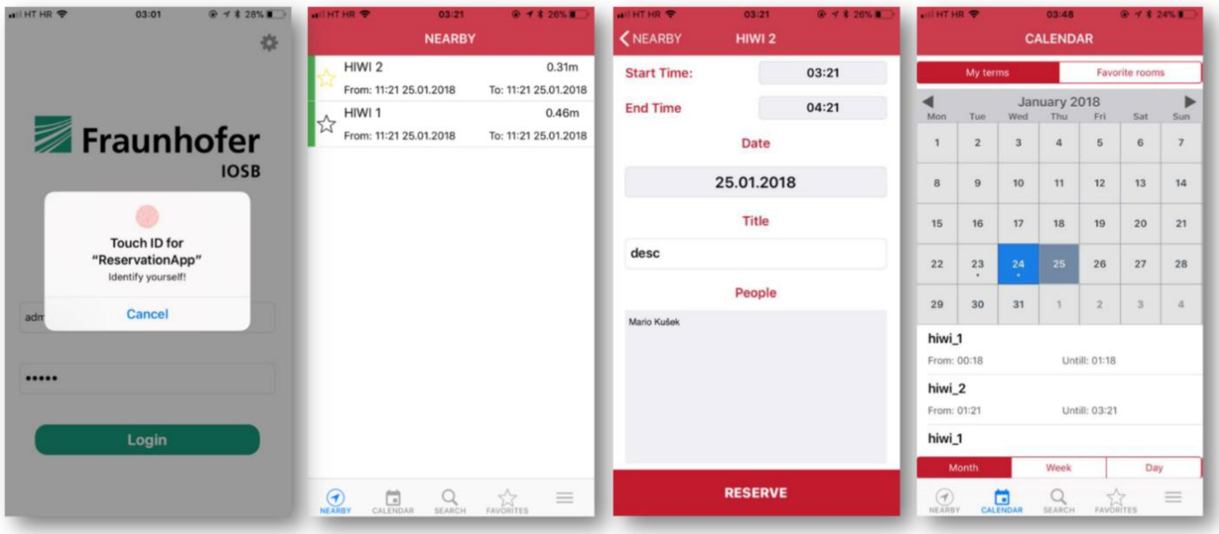
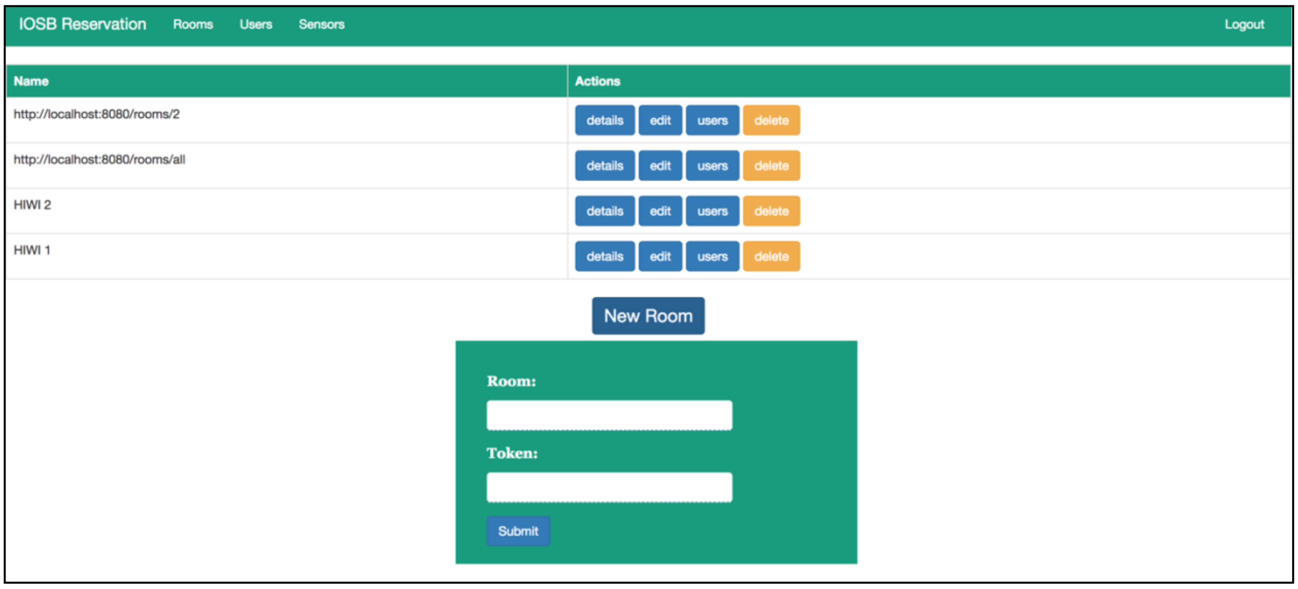


Figure 18: IOSB mobile app screenshot

5.1.3.2 IOSB Administration Site

The IOSB administration site is used for administration purposes. It allows the BLE beacon registration and room assignment and the user management.


IOSB Reservation Rooms Users Sensors Logout

Name	Actions
http://localhost:8080/rooms/2	details edit users delete
http://localhost:8080/rooms/all	details edit users delete
HIWI 2	details edit users delete
HIWI 1	details edit users delete

New Room

Room:

Token:

Submit

Figure 19: IOSB administration site screenshot

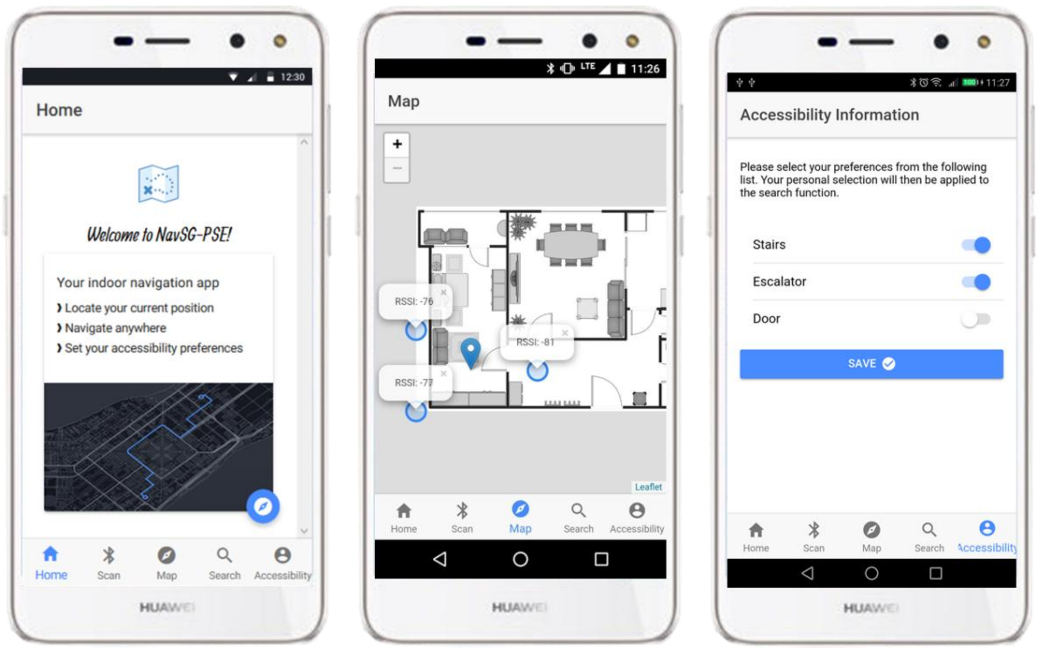
5.1.3.3 IOSB Backend Server

The IOSB backend server implements the IOSB room information model, the beacon management and the room reservation service. The Sensor Information will be stored in the FROST Server, which is an implementation of the OGC SensorThings API standard.

In its final release it will implement the symbIoTe RAP for L2.

5.1.3.4 KIT Mobile App

The KIT mobile application is a user frontend for the KIT navigation service. It reads BLE beacon information and connects to the KIT backend server. The user can select indoor areas and request navigation information starting from the current position. Some screenshots are shown in the figure below.



Welcome to NavSG-PSE!

Your indoor navigation app

› Locate your current position
› Navigate anywhere
› Set your accessibility preferences

Please select your preferences from the following list. Your personal selection will then be applied to the search function.

Stairs [✓]
Escalator [✓]
Door

Figure 20: KIT mobile app screenshots

5.1.3.5 KIT Administration Site

The KIT administration site is used for the beacon management and the room layout definitions. The interface is browser based and served by the KIT backend server.

5.1.3.6 KIT Backend Server

The KIT backend server implements the KIT information model, the beacon management and the navigation service.

In its final release it will implement the symbiote RAP for L2.

5.1.3.7 Compliance Level

The application complies with symbIoTe's L2.

5.1.3.8 Semantic Mapping of Platform Specific Information Models

The main feature of the EduCampus use case is the mapping between two different platform specific information models. The backend applications of both platforms will publish their resources with their original concepts.

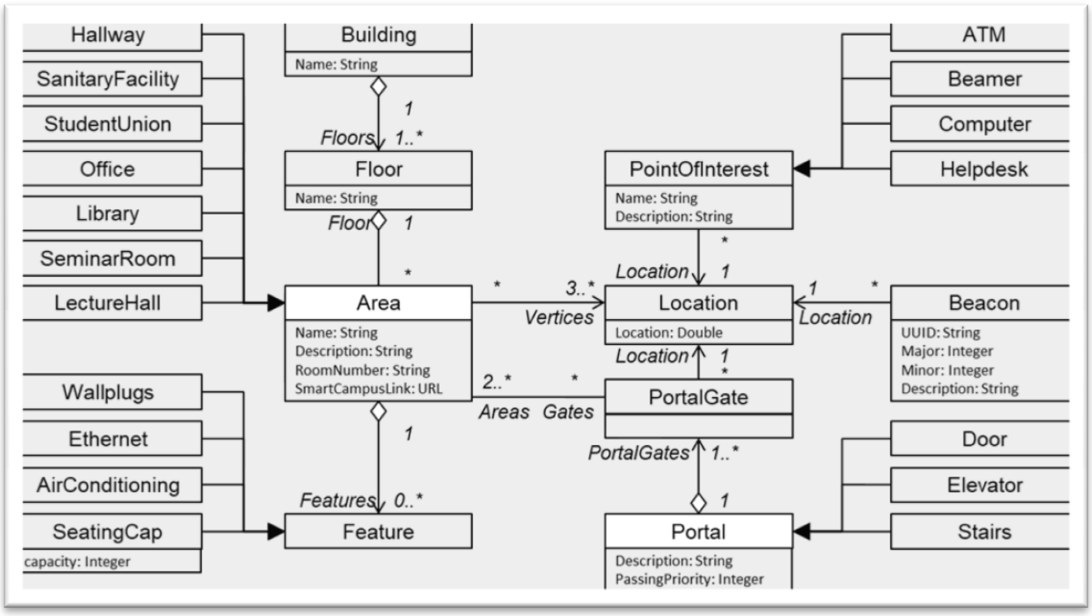
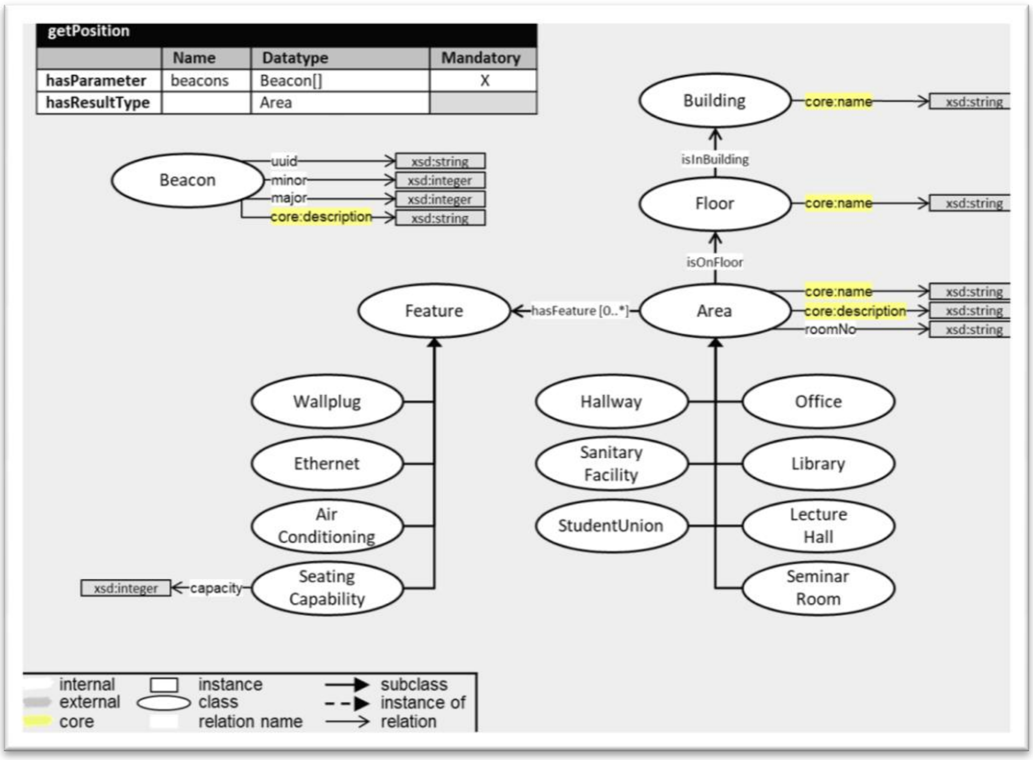


Figure 21: KIT application model

For the KIT application this will be an Area Location object, which is subtyped to more specific classes.



getPosition	Name	Datatype	Mandatory
hasParameter	beacons	Beacon[]	X
hasResultType		Area	

Figure 22: KIT platform specific information model

The IOSB application only supports generic Room objects with room related attributes.

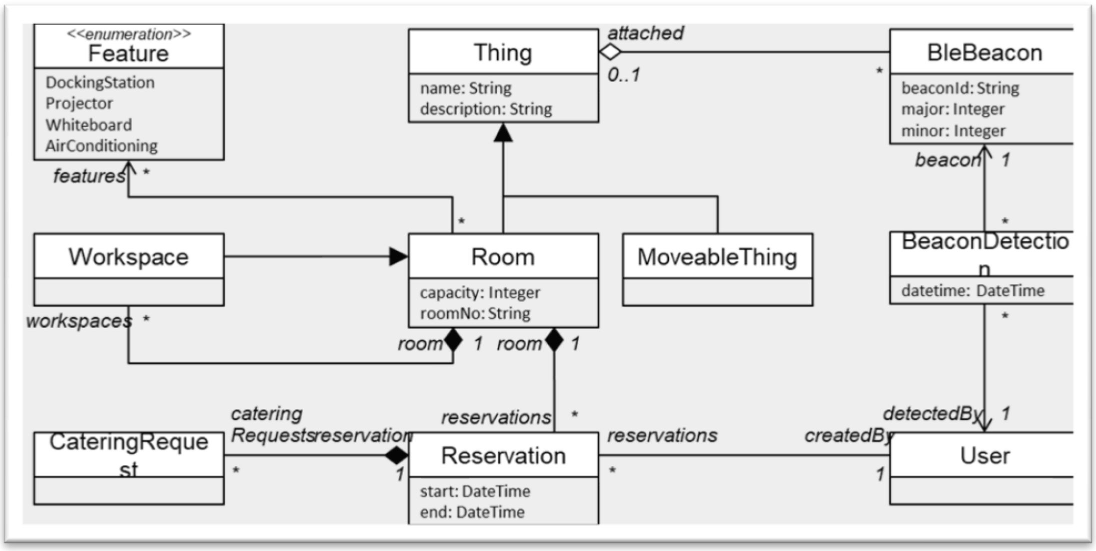
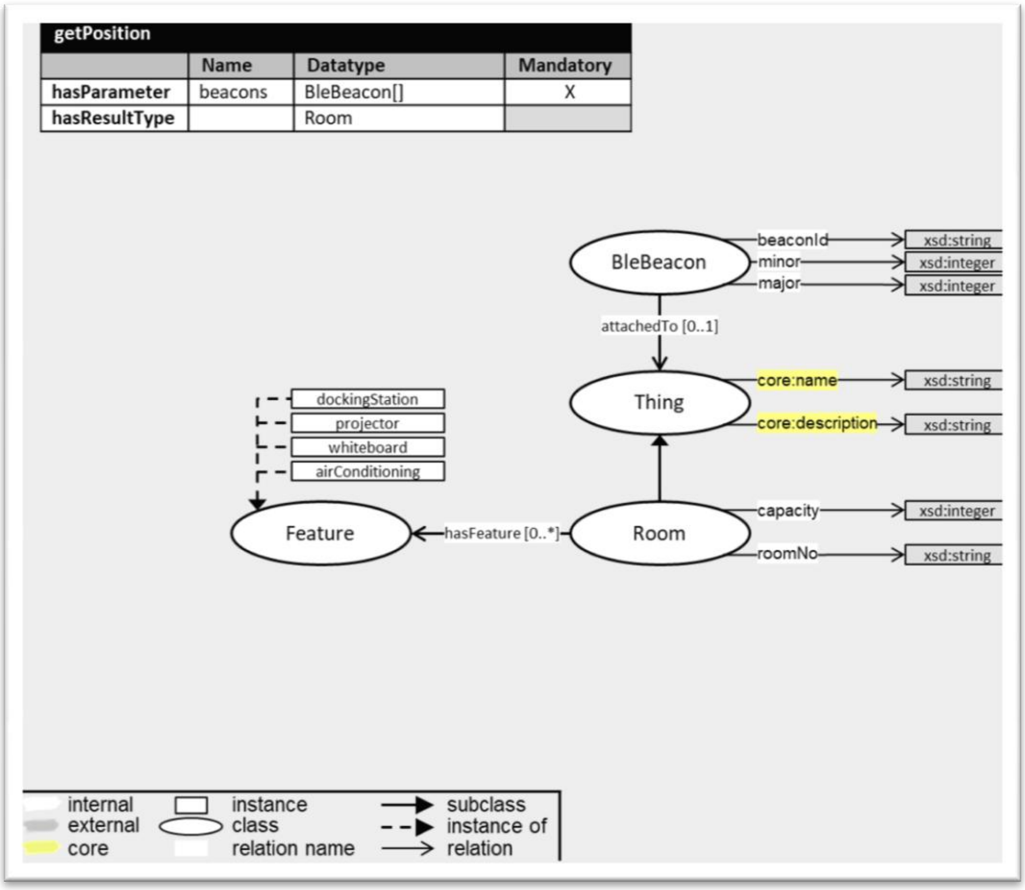


Figure 23: IOSB application model



getPosition	Name	Datatype	Mandatory
hasParameter	beacons	BleBeacon[]	X
hasResultType		Room	

Figure 24: IOSB platform specific information model

5.1.4 Smart Stadium

Smart Stadium enhances the user experience of visitors coming to a stadium. In the retail context, it provides that both visitors and retailers get closer even in large distances across the stadium.

Although being an IoT project, Smart Stadium does not involve sensors and actuators, which are usually the first thing that comes to one's mind when mentioning IoT devices. In this use case, however, smartphones, sales terminals and smart TVs are the IoT players.

Different IoT platforms can live together in the stadium, offering access to their devices to all other platforms and client applications through symbIoTe. Three different types of applications have been designed for this use case:

- Visitor application.
- Retailer application.
- Promowall application.

Visitors are identified by their smartphones, while retailers (both moving carts and physical shops) are identified by their Point of Sale Terminal and beacons. From the visitor's point of view, Smart Stadium brings the opportunity for detecting closest retailers, place orders independent of where they are, for receiving products they bought directly in their seat.

On the other hand, retailers can broadcast their offers and promotions to all visitors inside the stadium, or those that are moving near specific areas inside the stadium. Retailers can send their promotions to large SmartTVs (Promowalls) strategically placed throughout the stadium.

5.1.4.1 Visitor Application

The visitor application, in order to get much more downloads and use, have been integrated into the application of the club in which the trials take place, Atlètic Terrassa Hockey Club. The visitor application, named "ATLETIC Terrassa Oficial", can be downloaded from either Play Store (Android) or App Store (iOS), and provides visitors in the stadium access to all retailer information as well as an entry point of news and promotions. As soon as the user arrives to the stadium, the app registers user's location based on the proximity to beacons. From now on, the user is discoverable and accessible thanks to this application and its backend.

5.1.4.1.1 Design

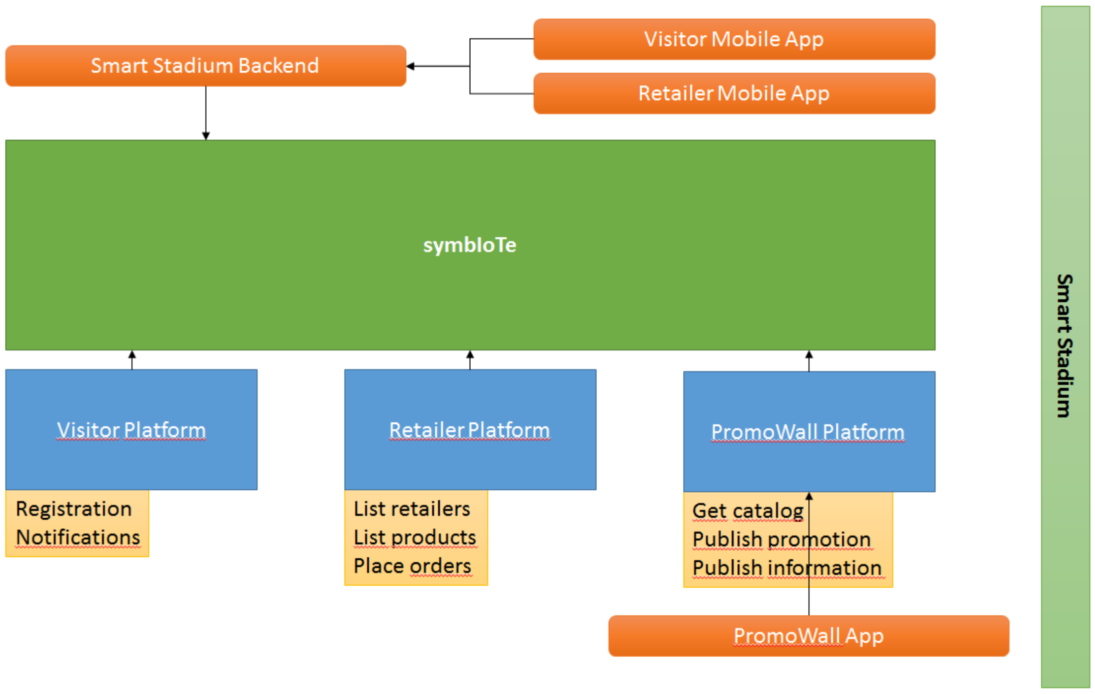


Figure 25: Components and interactions for all smart stadium use cases

The main benefit that symbIoTe provides to the application is the discoverability for new incoming devices to the stadium as well as the continuous status updates. It also standardizes the communication process by defining common information models, and rules to add custom ones.

The application uses a typical client and server architecture, where the server is the one that knows and interacts with symbIoTe. This way the mobile app capabilities can grow with no deep details on where those services come from (new platforms, a new enabler merging data from different platforms, etc.). Visitor platform provides access to known services registered in symbIoTe. The platform acts as a facade. All IoT devices are located using a custom symbolic location based on proximity to beacons spread throughout the stadium. All beacons emit unique identifiers that are used to locate IoT devices inside the stadium.

For example, a mobile app running on a Bluetooth low energy (BLE) capable device detects the following known beacons:

- Beacon 1 tagged as 'door 14' at a distance of 1 meter (near)
- Beacon 2 tagged as 'corridor 3' at a distance of 15 meters (far)
- Beacon 3 tagged as 'floor 1', at a distance of 1 meter (near)

The device is then located at the symbolic location "near door 14, near floor 1, far corridor 3". Physical shops can be easily located by using a specific beacon at the entrance door.

The following diagram depicts the device registration performed by both visitor and retailer applications as well as all components involved in the Smart Stadium use case.

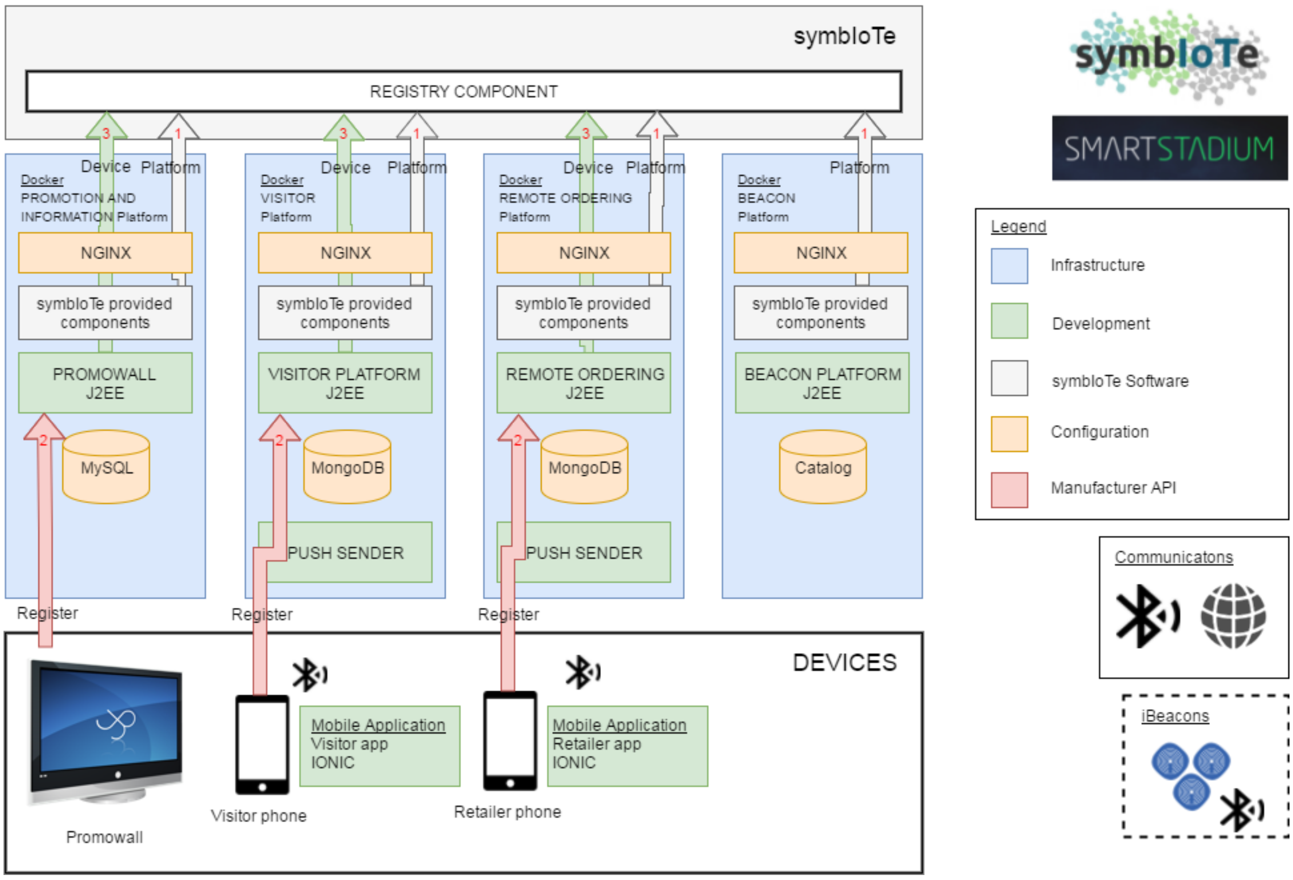


Figure 26: Smart Stadium Device Registration

5.1.4.1.2 Compliance level

The primary compliance level of symbIoTe integration is L1.

5.1.4.1.3 Platform

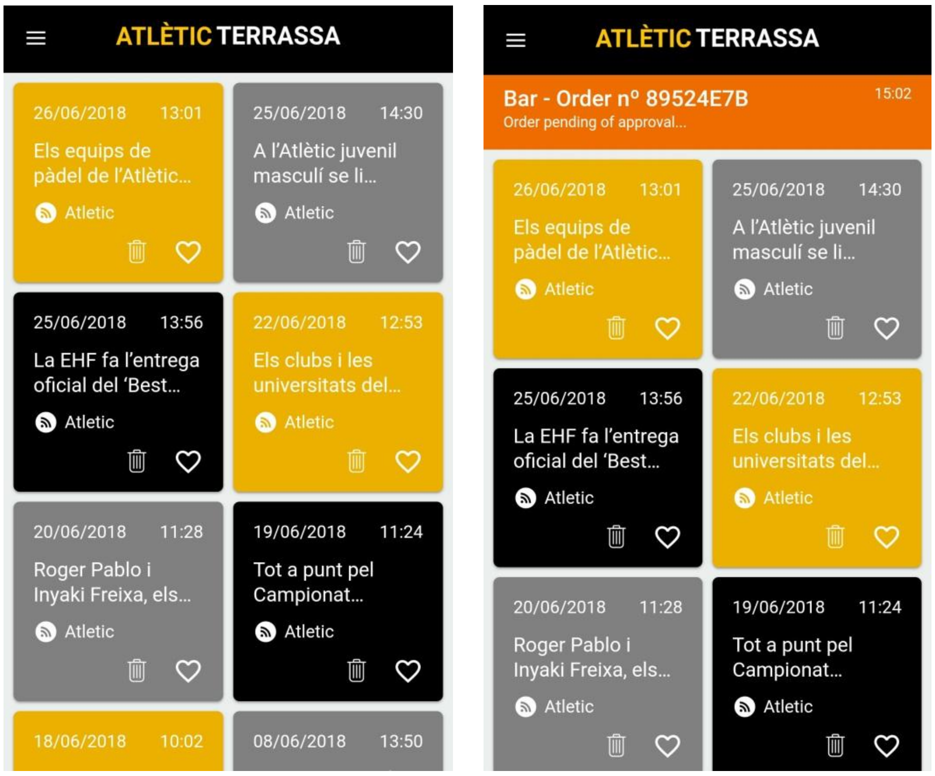
The visitor application solution is implemented using the following technology stacks:

- Mobile application: hybrid application developed using Cordova as a native envelope and Ionic and AngularJS as application core.

- Application: runs on Android devices with BLE capability to locate near beacons.
- Backend: J2EE stateless RESTful services implemented using Spring Framework and MongoDB.

5.1.4.1.4 User Interaction

The visitor application brings to people arriving to the stadium all information and services they need to get updated of sport events and products they might be interested in, for example, sport stuff, food and drinks. The other great functionality the application provides to the user is passive to the user: receiving incoming data from devices/people with granted access to symbIoTe and Visitor Platform related devices: push notifications.


ATLÈTIC TERRASSA

26/06/2018 13:01 Els equips de pàdel de l'Atlètic… Atletic

25/06/2018 14:30 A l'Atlètic juvenil masculí se li… Atletic

25/06/2018 13:56 La EHF fa l'entrega oficial del 'Best… Atletic

22/06/2018 12:53 Els clubs i les universitats del… Atletic

20/06/2018 11:28 Roger Pablo i Inyaki Freixa, els… Atletic

19/06/2018 11:24 Tot a punt pel Campionat… Atletic

18/06/2018 10:02

08/06/2018 13:50

ATLÈTIC TERRASSA

Bar - Order nº 89524E7B 15:02
Order pending of approval…

26/06/2018 13:01 Els equips de pàdel de l'Atlètic… Atletic

25/06/2018 14:30 A l'Atlètic juvenil masculí se li… Atletic

25/06/2018 13:56 La EHF fa l'entrega oficial del 'Best… Atletic

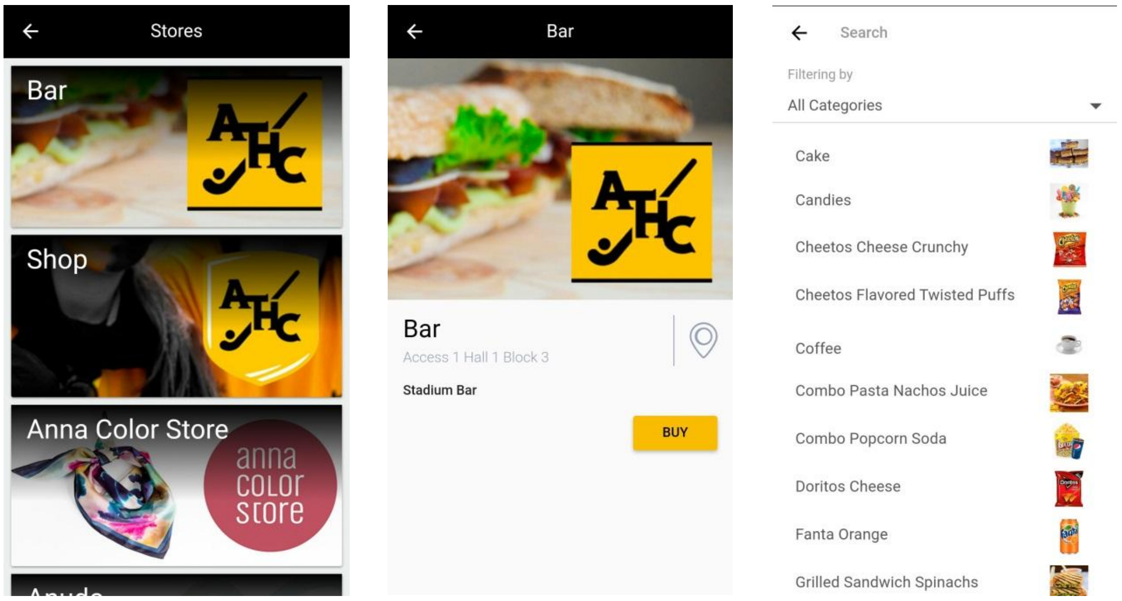
22/06/2018 12:53 Els clubs i les universitats del… Atletic

20/06/2018 11:28 Roger Pablo i Inyaki Freixa, els… Atletic

19/06/2018 11:24 Tot a punt pel Campionat… Atletic

Figure 27: Reception of notifications on visitor application

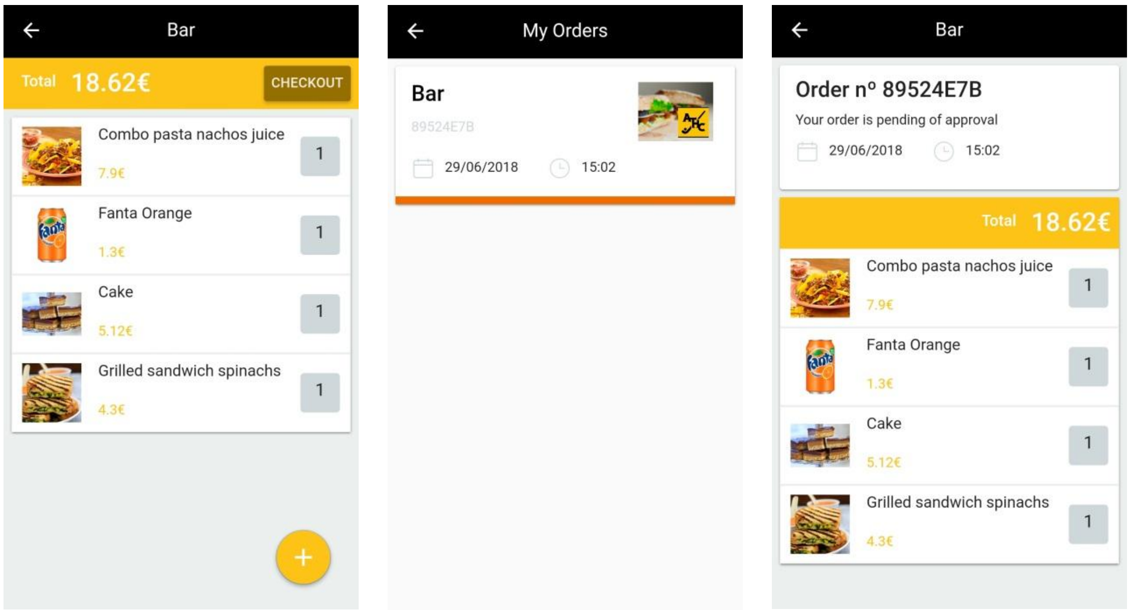
The visitor application looks for shops near the visitor and displays the products they sell; visitor prepares an order with some products (and maybe coupons and discounts) and sends it to the retailer, waiting for confirmation.


Stores
Bar
Shop
Anna Color Store
anna color store

Bar
Bar
Access 1 Hall 1 Block 3
Stadium Bar
BUY

Search
Filtering by
All Categories
Cake
Candies
Cheetos Cheese Crunchy
Cheetos Flavored Twisted Puffs
Coffee
Combo Pasta Nachos Juice
Combo Popcorn Soda
Doritos Cheese
Fanta Orange
Grilled Sandwich Spinachs

Figure 28: Closest shops to the visitor, selection, and list of available products


Bar
Total 18.62€ CHECKOUT
Combo pasta nachos juice 7.9€ 1
Fanta Orange 1.3€ 1
Cake 5.12€ 1
Grilled sandwich spinachs 4.3€ 1

My Orders
Bar
89524E7B
29/06/2018 15:02

Bar
Order nº 89524E7B
Your order is pending of approval
29/06/2018 15:02
Total 18.62€
Combo pasta nachos juice 7.9€ 1
Fanta Orange 1.3€ 1
Cake 5.12€ 1
Grilled sandwich spinachs 4.3€ 1

Figure 29: Visitor selects products and place the order

The communication between the Visitor platform and the mobile app is implemented by means of push messages.

5.1.4.1.5 Implementation

As mentioned above, there are two pieces of software involved in this application which are implemented in the following way:

- Backend: J2EE application implemented using the following frameworks and tools
  - Spring Framework: core framework
  - Apache Camel: to implement relevant processes

  - Spring Data: data layer abstraction
  - MongoDB: NoSQL database
  - Apache CXF: implement RESTful services
  - CAS: security for REST API
- Mobile application: hybrid application
  - Cordova: envelop providing access to native capabilities as well as platform specific application stores
  - Ionic and AngularJS: core framework to develop application logic and UI
  - Flux: data flow pattern for large applications

Moreover, this use case required a modification of the Core Information Model to add a generic entity *Device* as an abstraction for *Sensors* and *Actuators*, letting us to use different hardware (current version 2.3.0 reflects this).

Currently, both the mobile application and backend have completed their implementation.

5.1.4.1.6 Initial Functional Tests

In order to guarantee code quality, the backend has been analyzed using SonarQube[31] and JaCoCo[32] (code coverage), and the RESTful API has been tested using Postman. On the other hand, the mobile application has been tested using functional test cases. These functional tests took place in a first stage at the own premises of Worldline in Barcelona during the months of April and May 2018, and in a second stage at the trial location, on the premises of Atlètic Terrassa Hockey Club, during the month of June 2018.

We faced performance issues that caused the service to be down intermittently. Users' location updates were triggered too frequently and symbIoTe Core could not manage them.

5.1.4.2 Retailer Application

The retailer application provides retailers the opportunity to publish their services and products to anyone in the stadium. This application let retailers manage the order inbox and all the orders being processed and delivered. It also lets the seller to emit specific discounts and coupons to either visitors' devices or Promowalls located in certain sections of the stadium. It is intended to be integrated into the Point of Sale terminals of each retailer and, hence, be adapted to its particular look and feel (custom UI).

5.1.4.2.1 Design

The main benefit of using symbIoTe is the device discoverability via the Search registry to access all kind of symbIoTe enabled platforms that are currently defined but much more that could raise in the future.

[31] https://www.sonarqube.org/

[32] http://www.eclemma.org/jacoco/trunk/index.html

The application is divided into a regular client and server architecture (Figure 25), where the server is the one that knows and interacts with symbIoTe. This way client app capabilities can grow with no deep details on where those services come from (new platforms, a new enabler merging data from different platforms, etc.). Remote Ordering platform provides access to known services registered in symbIoTe, related to the Smart Stadium use case. The platform acts as a facade. All IoT devices are localized using a custom symbolic location based on proximity to beacons spread throughout the stadium.

All beacons emit unique identifiers that can be used to locate IoT devices inside the stadium.

5.1.4.2.2 Compliance Level

The primary compliance level of symbIoTe integration is L1.

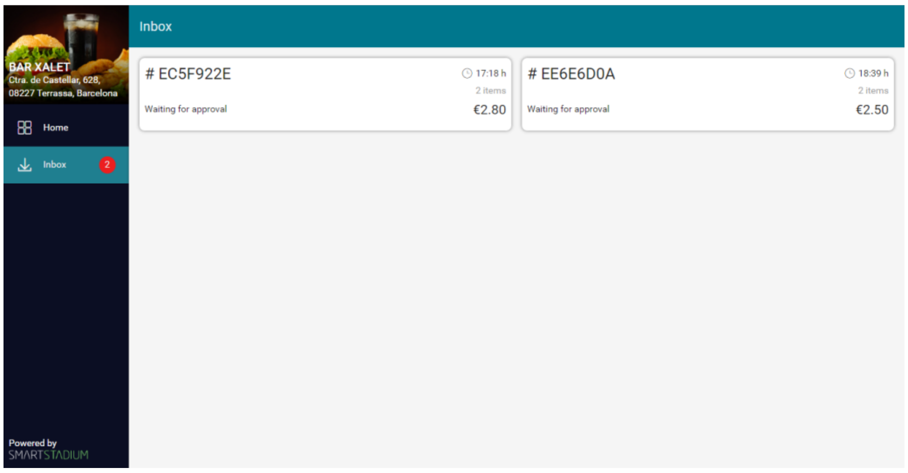
5.1.4.2.3 Platforms

The retailer application solution is implemented using the following technology stacks:

- Desktop application: web application developed using Electron as a native envelope and Ionic and AngularJS as application core.
- App runs on a RaspberryPi device with a plugged display.
- Backend: J2EE stateless RESTful services implemented using Spring Framework and MongoDB.

5.1.4.2.4 User Interaction

The retailer application allows sellers to make their products and services accessible to anyone in the stadium, send messages and offers to visitors and promowalls. As already described in the Visitors' application, the visitor can accept a promotion or place an order to a close retailer. The retailer application allows the retailer to receive and process all orders placed from visitors.



EC5F922E 17:18 h
2 items
Waiting for approval €2.80

EE6E6D0A 18:39 h
2 items
Waiting for approval €2.50

Figure 30: Retailer application receives orders from visitors

The retailer gets a specific order and checks its contents. If there is any problem, the order can be rejected and the visitor that placed the order notified about the rejection. Otherwise, the retailer prepares the order and, when ready, accepts the order. The visitor is notified that the order is ready.

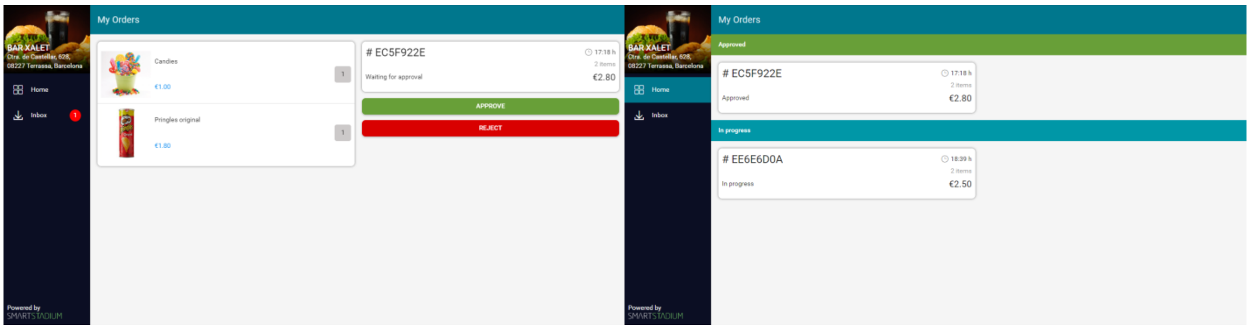


Figure 31: Retailer selection and acceptance of orders

5.1.4.2.5 Implementation

As mentioned above, there are two pieces of software involved in this application that are implemented in the following way:

- Backend: J2EE application implemented using the following frameworks and tools
  - Spring Framework: core framework
  - Apache Camel: to implement relevant processes
  - Spring Data: data layer abstraction
  - MongoDB: NoSQL database
  - Apache CXF: implement RESTful services
  - CAS: security for REST API
- Mobile application: desktop application developed using HTML5
  - Electron: envelop providing access to OS capabilities as well as platform specific installer and runtime
  - Ionic and AngularJS: core framework to develop application logic and UI
  - Flux: data flow pattern for large applications

Moreover, this use case required a modification of the Core Information Model to add a generic IoT device, as until then it only covered sensors and actuators.

Currently, both the mobile application and backend have completed their implementation.

5.1.4.2.6 Initial Functional Tests

In order to guarantee code quality, the backend has been analyzed using SonarQube and JaCoCo (code coverage), and the RESTful API has been tested using Postman. On the other hand, mobile application has been tested using functional test cases. These functional tests took place in a first stage at the own premises of Worldline in Barcelona during the months of April and May 2018, and in a second stage at the trial location, on the premises of Atlètic Terrassa Hockey Club, during the month of June 2018.

5.1.4.3 Promowall Application

Promowall is an existing solution from Worldline that offers the ability to publish stylish promotions and limited coupons to customers in two different channels: Promowall mobile app and large touch-screen Smart TVs, the *Promowalls*.

5.1.4.3.1 Design

The main benefit of using symbIoTe is the broadcasting of the Promowall published information thanks to the discoverability of new devices and the ability to introduce new symbIoTe enabled platforms that could use Promowall.

The Promowall backoffice application was implemented as a monolithic piece of software containing a RESTful API, used by the mobile app, and a backoffice GUI implemented using ZK Framework (server-side rendering).

The Promowall solution is divided into three applications:

- Promowall backoffice (out of Smart Stadium use case, because it's replaced by the Retailer application described above).
- Frontend HTML5 application running on Promowall devices displaying promotions, relevant information and the QR code to activate promotions using any QR scanner application from their devices (Promowall mobile app is not even required).
- Promowall mobile application for final users.

5.1.4.3.2 Compliance Level

The primary compliance level of symbIoTe integration is L1.

5.1.4.3.3 Platforms

The Promotion and Information platform is nothing else than the symbIoTe enabled Promowall backend.

The retailer application solution is implemented using the following technology stacks:

- RESTful API: J2EE web application using Spring Framework.
- SmartTV frontend: HTML5 application
- Mobile application: Android native application running on a BLE capable device

5.1.4.3.4 User Interaction

The Promowall solution relevant for Smart Stadium use case involves the interaction of visitors with Promowalls. The visitor can select and capture a promotion on the promowall device and use the QR code of the promotion to redeem it at the physical store, or place an order and pay on the mobile device if the retailer has enabled this option.

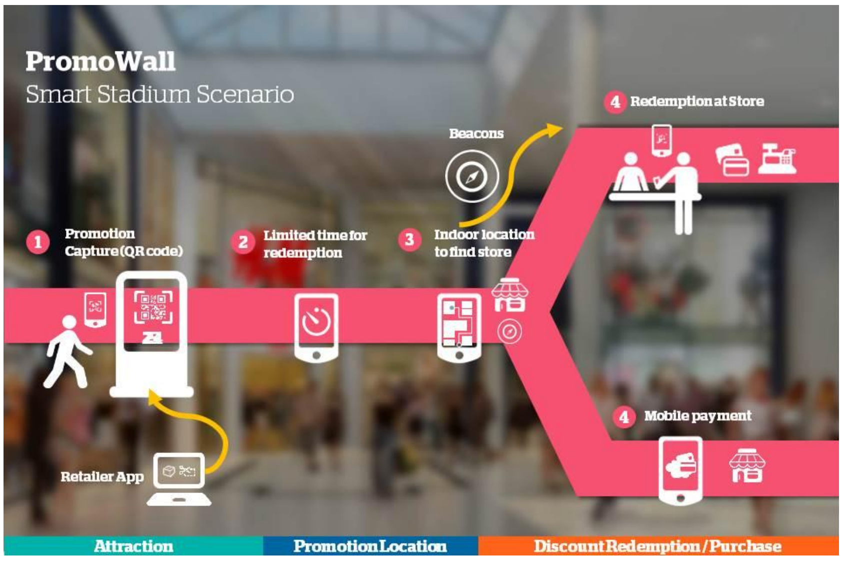
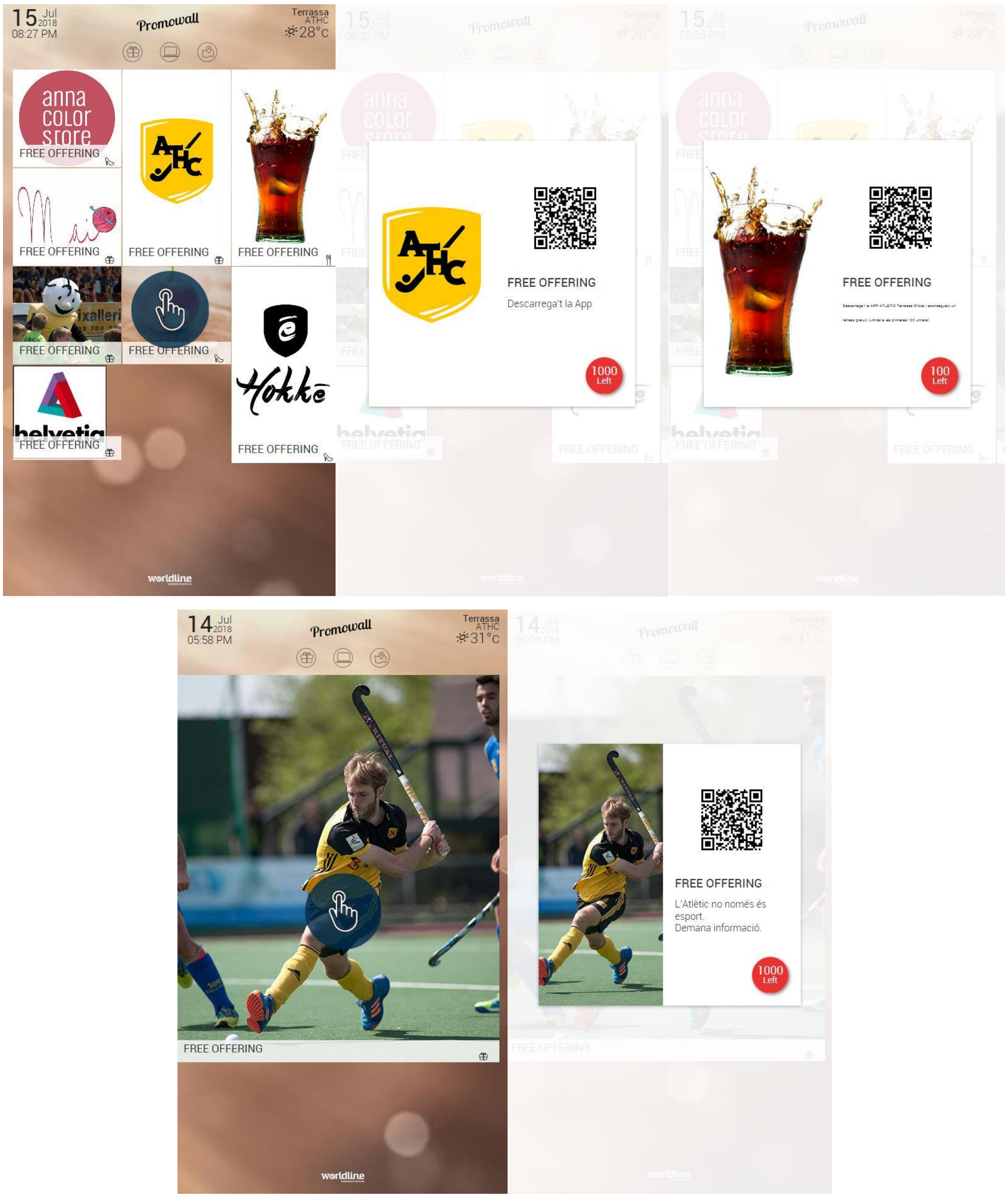

PromoWall
Smart Stadium Scenario
1 Promotion Capture (QR code)
2 Limited time for redemption
3 Indoor location to find store
Beacons
4 Redemption at Store
4 Mobile payment
Retailer App
Attraction
Promotion Location
Discount Redemption / Purchase

Figure 32: Promowall in the Smart Stadium scenario

The following figure shows some of the promotions that have been used for the Smart Stadium use case on the Promowall device.



L'Atlètic no només és esport.
Demana informació.

Figure 33: Promotion at the Smart Stadium Promowall

5.1.4.3.5 Implementation

As mentioned above, there are three pieces of software involved in this application that are implemented in the following way:

- Backend: J2EE application implemented using the following frameworks and tools
    - Spring Framework: core framework
    - MySQL database

    - Apache CXF: implement RESTful services
- SmartTV frontend: HTML5 site
    - jQuery: core library
    - Mustache.js: logic-less HTML template library
    - Hammer.js: gesture library for web
    - Isotope: dynamic masonry tile layout
- Mobile application: Android native application

This application does not require any special attention. The effort on this platform is to convert this in a real *symbIoTe platform*.

Currently Promowall application has been completed and adapted as a symbIoTe enabled platform.

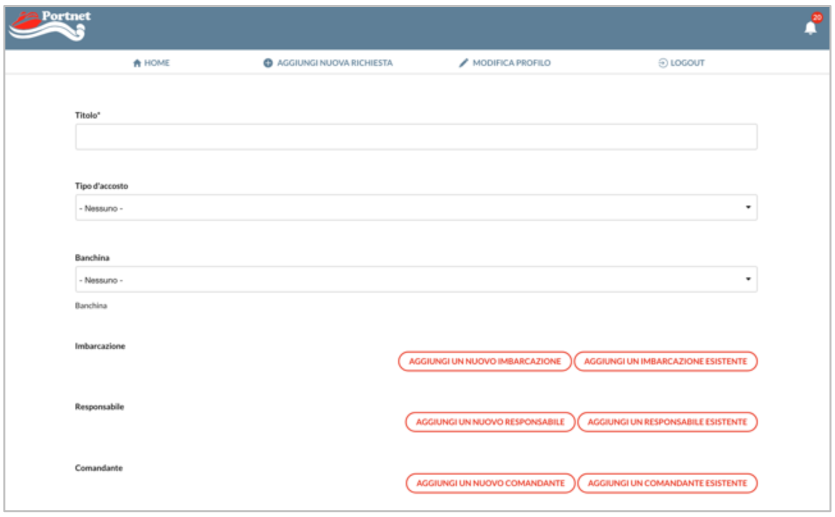
5.1.4.3.6 Initial Functional Tests

This application is fully developed and it does not need any new functional tests.

5.1.5 Smart Yachting

The focus of Smart Yachting is to provide advanced services for the Yachting industry based on IoT solutions. From an implementation viewpoint, the use case focuses on two specific showcases: Smart Mooring and Automated Supply Chain (ASC).

The former aims to automate the mooring procedure of the Port, in itself a quite bureaucratic and tedious process, since Marinas operate in strongly regulated contexts. For the use case, the workflow logic is provided by the Navigo application Portnet.


Portnet
20
HOME
AGGIUNGI NUOVA RICHIESTA
MODIFICA PROFILO
LOGOUT
AGGIUNGI UN NUOVO IMBARCAZIONE
AGGIUNGI UN IMBARCAZIONE ESISTENTE
AGGIUNGI UN NUOVO RESPONSABILE
AGGIUNGI UN RESPONSABILE ESISTENTE
AGGIUNGI UN NUOVO COMANDANTE
AGGIUNGI UN COMANDANTE ESISTENTE

Titolo*

Tipo d'accosto
- Nessuno -

Banchina
- Nessuno -
Banchina

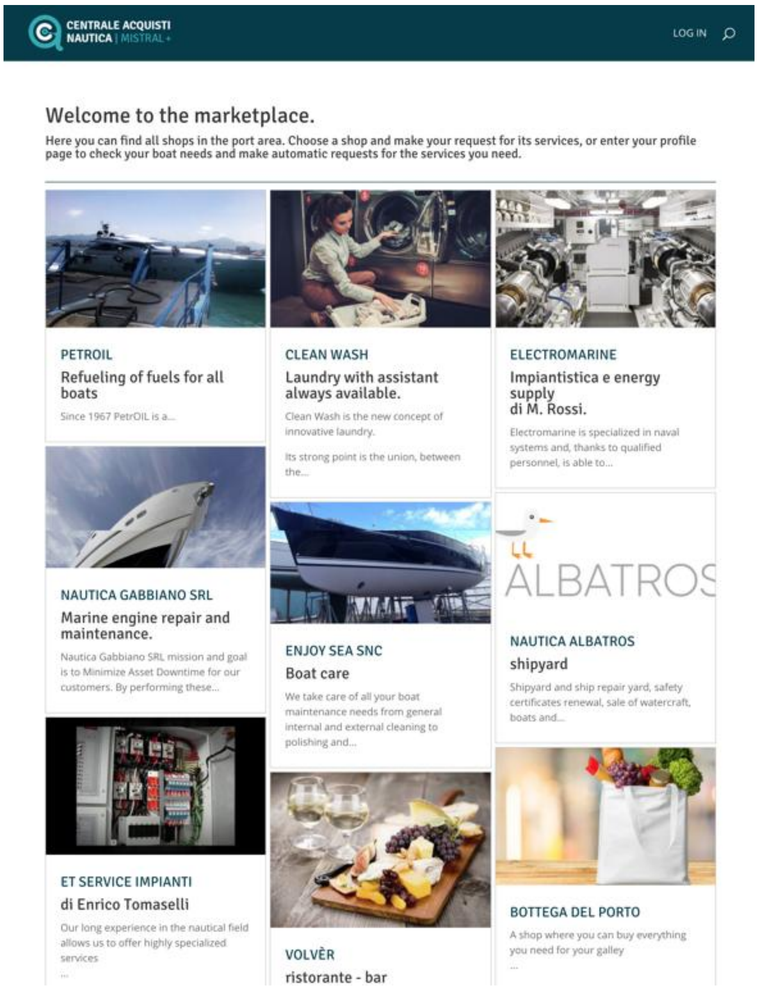
Imbarcazione

Responsabile

Comandante

Figure 34: A screenshot of the Portnet application

ASC aims to automatically identify the needs for goods and services on board of the Yacht, so that automated requests for offers can be issued on the marketplace platform of the Port, provided by another application of the Navigo infrastructure, Centrale Acquisti.



Welcome to the marketplace.

Here you can find all shops in the port area. Choose a shop and make your request for its services, or enter your profile page to check your boat needs and make automatic requests for the services you need.

PETROIL
Refueling of fuels for all boats
Since 1967 PetrOIL is a...

CLEAN WASH
Laundry with assistant always available.
Clean Wash is the new concept of innovative laundry.
Its strong point is the union, between the...

ELECTROMARINE
Impiantistica e energy supply di M. Rossi.
Electromarine is specialized in naval systems and, thanks to qualified personnel, is able to...

NAUTICA GABBIANO SRL
Marine engine repair and maintenance.
Nautica Gabbiano SRL mission and goal is to Minimize Asset Downtime for our customers. By performing these...

ENJOY SEA SNC
Boat care
We take care of all your boat maintenance needs from general internal and external cleaning to polishing and...

NAUTICA ALBATROS
shipyard
Shipyard and ship repair yard, safety certificates renewal, sale of watercraft, boats and...

ET SERVICE IMPIANTI
di Enrico Tomaselli
Our long experience in the nautical field allows us to offer highly specialized services
...

VOLVÈR
ristorante - bar

BOTTEGA DEL PORTO
A shop where you can buy everything you need for your galley
...

Figure 35: A screenshot of the Centrale Acquisti application

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

The choice of implementing Enablers in the use case has been motivated by the need to facilitate the integration of Ports' business applications with symbIoTe. This way it is possible to encapsulate the technical details of the whole integration logic and expose only the minimum set of methods that application developers must implement to integrate their specific Mooring Workflow Management Systems and Marketplace solutions in the use case.

A description for each of the two showcases of Smart Yachting follows.

5.1.5.1 Smart Mooring

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

The showcase wants to intercept a particular phase of the Mooring process that starts when the Yacht is approaching – at a distance – the destination port and ends when it finally berths into one of its piers.

It is assumed that the initial mooring request (a sort of "booking" for the boat in the Port) always starts offline or in any case outside symbIoTe.

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

When approaching the port, the vessel is first detected through LoRaWAN (we assume that a LoRA antenna is controlled by the Port's IoT platform). When the Yacht is near the port, through LoRaWAN or other strategies, Wi-Fi credentials are transmitted to the Yacht which starts a full Internet connection and can connect to the Smart Space (SSP).

Since the Yacht is a roaming Smart Device (L3 & L4 symbIoTe compliance levels), its resources and properties must be updated in the Registry. In particular the ConnectionStatus and ConnectedInPort attributes allow to know whether a Yacht is connected to a Port's Smart Space and in which Port this has happened. To reflect this situation, the SSP's Innkeeper aptly updates the Yacht/SDEV properties in the Registry.

When the Yacht is fully connected to the SSP, the PortNet's Enabler can invoke the SDEV services to retrieve data from boat sensors: M2M data is passed to the Mooring application and attached to the specific approval workflow. In particular we assume to acquire from the vessel's sensors:

- Latest route, mapped as a sequence of waypoints, each described by geographical coordinates.
- Average Yacht speed (in knots)
- Average Fuel Consumption per nautical mile (in litres)
- Fresh, Grey and Black Water tanks level (in litres)
- Service Fuel and Storage Fuel Oil tanks level (in litres)
- Port Exhaust and Starboard Exhaust temperature (in degree Celsius).

The whole process allows to greatly simplify the mooring management of the Port. First of all, it is possible to automatically detect the vessel when it is approaching the port area but still at a distance, allowing to send alerts to the port personnel in the piers to wait for the incoming boat; yacht data that must be specified in the authorization procedure can be automatically acquired and there is no need to manually copy them on paper forms; last but not least, there is no need for the yachtsman to physically go to the Port Authority's offices, unless any problems are detected and reported.

5.1.5.1.1 Design

The Mooring Workflow application interacts, through symbIoTe's components, with the Yacht's IoT platform to receive data from sensors that must be attached to the authorization workflow, while sensors in the Port area, managed by its IoT platform, can recognize when the Yacht has finally berthed on the assigned pier.

For the use case, we are integrating IoT platforms and applications of project partners, namely Nextworks' Symphony and Navigo's Navigo Digitale and Portnet.

Allowing other Mooring Applications, beyond Navigo's Portnet, to become symbIoTe-enabled is at the same time essential and a critical factor, since there isn't any standard, nor a market leader in this arena. In order to encourage software vendors to adopt this model, we must simplify their work; that is why it has been decided to encapsulate the integration details within an enabler which hides all the possible complications and provides simple cooperation mechanisms.

5.1.5.1.2 Compliance Level

Smart Mooring has:

- L3 compliance: as said, we see the Yacht as a Smart Device (SDEV) and the Port as a Smart Space (SSP). The implementation of the showcase becomes similar to the symbIoTe scenario of a Smart Device (the Yacht) entering a Smart Space (the Port).
- L4 compliance: we assume that the Yacht maintains its ID when moving between Ports. The Yacht will be therefore seen as an example of a Roaming Device.

By default the use case also implies L1 symbIoTe compliance.

5.1.5.1.3 Platforms

Smart Mooring involves the use of Navigo Digitale IoT platform and of the Navigo's business application Portnet, the latter integrated in symbIoTe through an Enabler. On the Yacht side, Nextworks' Symphony IoT platform is used.

5.1.5.1.4 User Interaction

No specific GUI is needed for this showcase. The only GUIs are those of the Portnet application which is beyond the scope of symbIoTe.

5.1.5.1.5 Implementation

The following programming languages have been used for the development of the showcase:

- Navigo Digitale IoT platform: its backend has been developed in Python while the frontend is a web application. It is an L1 compliant platform.
- Portnet is a web application developed in PHP by using the Drupal 8 framework.
- The Enabler has been implemented in Java.
- The Smart Device logic (L3/L4), integrated with the Yacht's IoT platform (Nextworks' Symphony), has been implemented in Python on a Raspberry Pi 3.

5.1.5.1.6 Initial Functional Tests

A specific test plan has been defined to cover all kinds of tests, from functional to integration and possibly load testing.

Moreover the showcase is tested live through trials foreseen in the Port of Viareggio: a real yacht, with all the hardware and software components foreseen for the showcase, will navigate towards the Viareggio port. Before, a mooring workflow procedure will be initiated on the Portnet application, to make sure that all systems are ready for the arrival of the boat.

To make the trial successful, it is necessary that the following events are correctly recognized and managed by the involved systems:

- The yacht is detected through LoRaWAN when still at a distance from the port; accordingly, a message is sent to the Portnet application, which successfully manages this communication by updating the workflow and alerting both the Port Authority operators and the Port Area workers.
- The yacht's IoT platform, when the vessel is near the berthing pier, connects to the port's Wi-Fi network and, through the S3M, to the symbIoTe infrastructure. Data from sensors on board are sent to Portnet and attached to the workflow of the current mooring procedure. Communications to the aforementioned users are sent: in particular Port Area workers are requested to move to the berthing pier to wait for the incoming yacht.
- The presence sensors on the pier detect when the yacht has finally berthed: a communication is sent to the Portnet application that can successfully close the workflow and inform the Port Authority operators.

Trials have already started and will continue until September 2018.

5.1.5.2 Automated Supply Chain (ASC)

The purpose of ASC showcase is to allow the Ports' marketplace web applications to access the resources of a Yacht to retrieve information about the needs of goods or services on board, as identified by the vessel's sensors.

Like in the previous case, we assume that the showcase always starts offline or in any case outside symbIoTe.

5.1.5.2.1 Design

Similarly to the case of Smart Mooring, we have an application here (Navigo's Centrale Acquisti) that exploits M2M and symbIoTe to automatically get the list of the possible needs of goods and services on board of the Yacht as detected by its IoT platform (for the use case, Nextworks' Symphony). Again, we aim to involve other third party software vendors that provide applications similar to Centrale Acquisti: in order to simplify their integration task, we decided to use an enabler to mediate the interaction with the symbIoTe infrastructure.

The use of symbIoTe in the Centrale Acquisti application simplifies how Yachtsmen can resupply or execute maintenance tasks on the Yacht by automatically finding possible sellers or service providers in the area, even on their first visit to the (symbIoTe enabled) Port.

5.1.5.2.2 Compliance Level

Similarly to the former case, we assume that the Yacht is a Roaming device that interacts with the symbIoTe's ecosystem through the Port's Smart Space. Therefore the showcase is compliant to the L1, L3 and L4 symbIoTe levels.

5.1.5.2.3 Platforms

ASC is based on the integration of the Navigo's business application Centrale Acquisti in symbIoTe through an enabler. On the Yacht side, Nextworks' Symphony IoT platform is used.

5.1.5.2.4 User Interaction

No specific GUI is needed for this showcase. The only GUIs are those of the Centrale Acquisti application (beyond the scope of symbIoTe).

5.1.5.2.5 Implementation

The following programming languages have been used for the development of the showcase:

- Navigo Digitale IoT platform (L1 compliant): its backend has been developed in Python while the frontend is a web application.
- Centrale Acquisti: it consists of a PHP application, implemented with the WordPress framework.
- Enabler: implemented in Java.

5.1.5.2.6 Initial Functional Tests

A specific test plan has been defined to cover all kinds of tests, from functional to integration and possibly load testing.

Moreover live testing of this showcase has already started in the trial foreseen in the Port of Viareggio; other trials will also be performed in the Marina Cala De' Medici port, again in Tuscany, Italy.

The trial again involves a Yacht, this time berthed on a pier of the Port and steadily connected through Wi-Fi at the Port's Smart Space. The Yacht is configured (through real actions or simulations) to express a certain amount of maintenance or supply needs.

The trial consists of the following steps:

- From the Centrale Acquisti web interface a request to access the Yacht's machine data is made.
- Centrale Acquisti accesses the Yacht's resources – through its enabler – to have the list of the needs on board. The application must perform a corresponding match-making with the possible suppliers in the Port area (in particular of those involved in the trial).

The Port Authority operators supervise the flow of requests and the successful execution of the matchmaking actions performed by Centrale Acquisti, given the machine data received from the yacht.

The local suppliers use the backend of the Centrale Acquisti to answer to the requests of offer automatically generated by the system: in particular they evaluate if the information acquired by sensors on board and received through symbIoTe's services are detailed enough (or simply useful) to allow them to produce an offer, without the need to directly contact the yachtsman.

The test will be repeated simulating different conditions (and therefore needs) on board until September 2018.

5.2 *Applications Developed for Demos*

5.2.1 Demo Web App

The symbIoTe demo app in a Web Application is developed to showcase L1 symbIoTe functionalities. This includes search of resources, access to historical data and actuation capabilities. It only makes use of public resources, thus it does not have login capabilities, but it uses the symbIoTe Client component to handle requests.

5.2.1.1 Design

The web app mostly interacts with symbIoTe using the symbIoTe Client, which handles some access logic such as setting the security headers and the fetching of certificates.

The application has been developed to be generic enough to be able to show any registered public resource in symbIoTe. This includes showing their location, their type and the platform they belong to. For sensors, the app is also able to access and show sensor readings. For actuators, specific widgets were developed to interact with light bulbs (changing RGB lights, light intensity, on/off), although they will work with lightbulbs registered in different platforms. In the future, this behaviour could become more generic by presenting widgets according to the registered parameters of the actuator.

5.2.1.2 Compliance Level

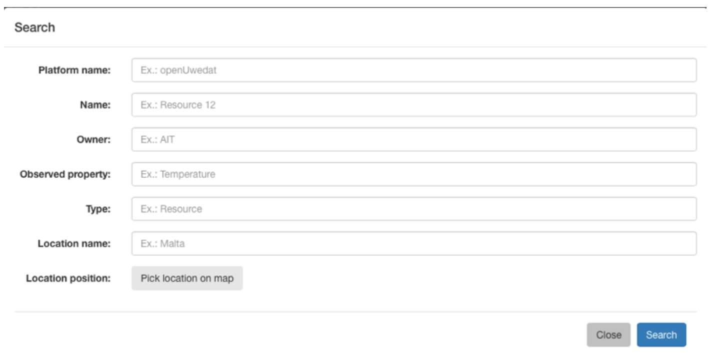
The web app is L1 symbIoTe compliant.

5.2.1.3 Platforms

The web app was designed to work with any kind of platform registered within symbIoTe (L1 compliance), but it has mostly been tested with sensor readings from the openUwedat, OpenIoT and MoBaaS platforms and with actuators from Symphony and OpenHub platforms.

5.2.1.4 User Interaction

The user is firstly presented with a search screen (Figure 36) where search parameters can be inserted. It is also possible to not provide any input, thus receiving every result available.

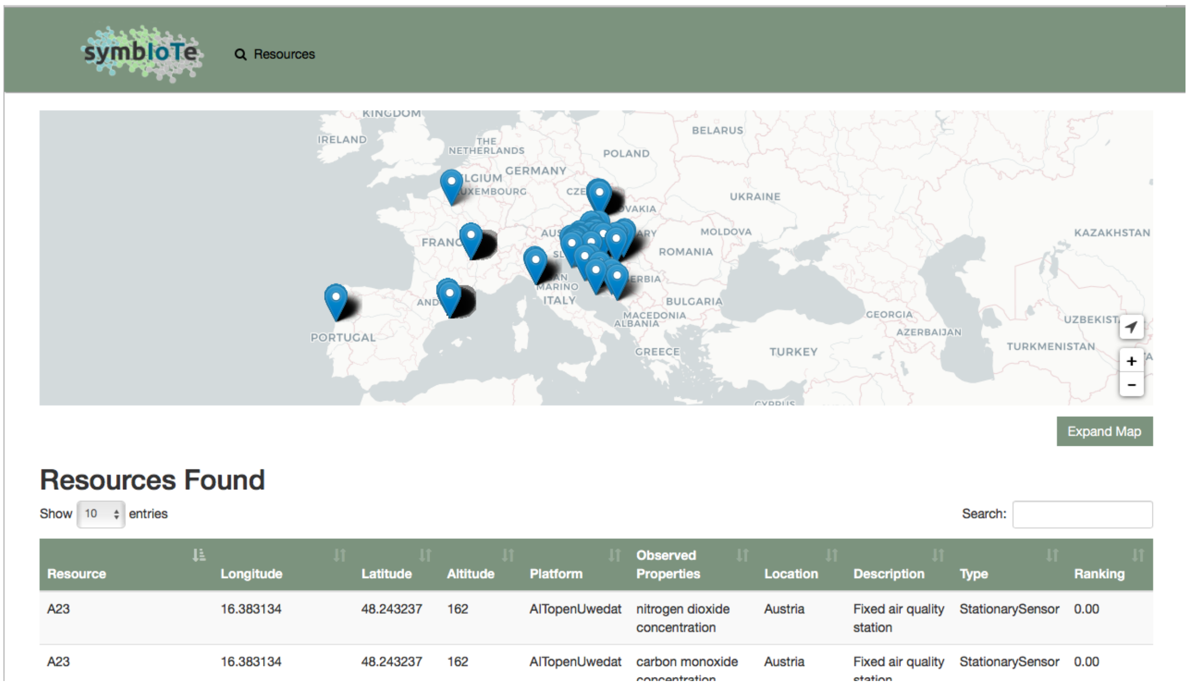


Search

Platform name:	Ex.: openUwedat
Name:	Ex.: Resource 12
Owner:	Ex.: AIT
Observed property:	Ex.: Temperature
Type:	Ex.: Resource
Location name:	Ex.: Malta
Location position:	Pick location on map

Figure 36: Web App Search

The user is then taken to the main screen, where he/she is presented with a map with pins indicating resource location and a list containing the resources' metadata (Figure 37).



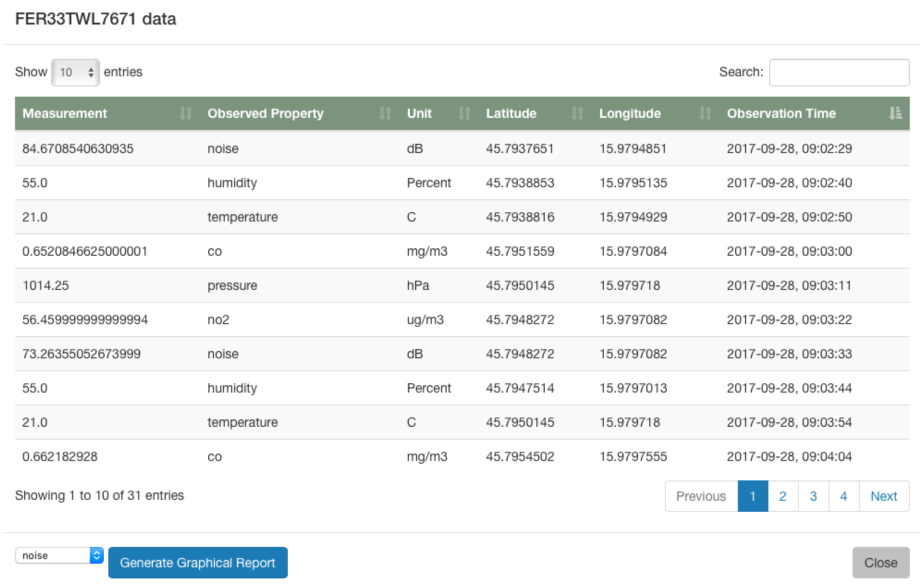
Resources Found

Show 10 entries Search:

Resource	Longitude	Latitude	Altitude	Platform	Observed Properties	Location	Description	Type	Ranking
A23	16.383134	48.243237	162	AITopenUwedat	nitrogen dioxide concentration	Austria	Fixed air quality station	StationarySensor	0.00
A23	16.383134	48.243237	162	AITopenUwedat	carbon monoxide	Austria	Fixed air quality	StationarySensor	0.00

Figure 37: Web App main screen

If the user clicks a specific pin on the map, information regarding that pin is presented, such as the platform it belongs to, its coordinates and its type. If the user clicks on an entry of the list, the web app accesses the sensor historic information and presents it to the user (Figure 38).



FER33TWL7671 data

Show 10 entries Search:

Measurement	Observed Property	Unit	Latitude	Longitude	Observation Time
84.6708540630935	noise	dB	45.7937651	15.9794851	2017-09-28, 09:02:29
55.0	humidity	Percent	45.7938853	15.9795135	2017-09-28, 09:02:40
21.0	temperature	C	45.7938816	15.9794929	2017-09-28, 09:02:50
0.6520846625000001	co	mg/m3	45.7951559	15.9797084	2017-09-28, 09:03:00
1014.25	pressure	hPa	45.7950145	15.979718	2017-09-28, 09:03:11
56.459999999999994	no2	ug/m3	45.7948272	15.9797082	2017-09-28, 09:03:22
73.26355052673999	noise	dB	45.7948272	15.9797082	2017-09-28, 09:03:33
55.0	humidity	Percent	45.7947514	15.9797013	2017-09-28, 09:03:44
21.0	temperature	C	45.7950145	15.979718	2017-09-28, 09:03:54
0.662182928	co	mg/m3	45.7954502	15.9797555	2017-09-28, 09:04:04

Showing 1 to 10 of 31 entries

Figure 38: Web App sensor Readings

The functionalities presented until this point are generic, making use of symbIoTe capabilities to search for and present information regarding all registered resources, but specific logic was also developed to actuate light bulbs. With this functionality, the web app is able to interact with any light bulb from any platform, as long as it has been registered using the symbIoTe information model.

5.2.1.5 Implementation

The web app has been developed with JavaScript, using bootstrap plugins and leaflet libraries.

5.3 Enabler-based Applications

5.3.1 Indoor Positioning based on Location Enabler

Indoor positioning (and subsequently navigation) has been a field of recent interest, though so far there has not been much visible results, beyond some trials and a small number of locations where such services have been offered, based on specialized implementations. While outdoor positioning and navigation can be considered “solved” (in most outdoor cases) with the help of Global Navigation Satellite Systems (GNSS) system such as GPS, Galileo, GLONASS and BeiDeo, providing rather quick and precise positioning, practical, useful and universal indoor positioning is still quite far away.

Useful Indoor Positioning and subsequent Navigation has numerous applications in various use cases (both observed within symbIoTe scope and outside) – from simple finding a car in a large closed parking garage, finding a shop or ATM within a large shopping centre all the way to complex indoor navigation solutions (finding various resources in a complex corporate, academic, sport or health campus), not mentioning the upcoming need to enable vehicle positioning in tunnels for some future autonomous driving applications.

Based on the design defined in D2.6 (chapter 5.4) [6] an indoor positioning system has been implemented which can use multiple available infrastructures (BLE beacons, WiFi access points, mobile network cell information) to provide a most precise indoor position. Such hybrid solutions are foreseen in the upcoming 3GPP-based mobile networks (5G networks), thus we can expect future interest and implementations of indoor positioning, with the ultimate goal of a standardized implementation.

The application is currently developed in a Smart Campus environment (on the example of Vipnet Zagreb Žitnjak main campus and a testing setup on UNIZG-FER in their IoT Lab environment), though the design and development approach enables simple creation of the specific implementation for the other environments.

5.3.1.1 Design

Indoor Positioning is determined by trilateration (positioning by calculating distance from pre-determined points) based on the various radio transmitters that exist within the environment. Transmitters can include existing public mobile network base stations, existing wireless LAN access points but it can also include specialized BLE beacon transmitters installed for the purpose of the augmenting indoor navigation. Distance to the fixed transmitters is determined by the RSSI (Received signal strength indication) signal level received by the mobile phone wireless receivers and calculated using the algorithms that use transmit power and propagation loss.

Algorithms are different for each wireless technology and there are issues with signal propagation through obstacles (including people present in the room. Thus, there are limits to precision that can be achieved, which depend for example on the number of transmitters and their positioning in the space. Also, often WiFi access points are not distributed in a practical fashion for trilateration. Finally, today mobile network cells (even in indoor coverage systems) are still too large for practical and precise indoor positioning. Mobile network cells will become smaller in the future with upcoming 5G mobile technologies, which will increase the usage of smaller micro-cells and nano-cells.

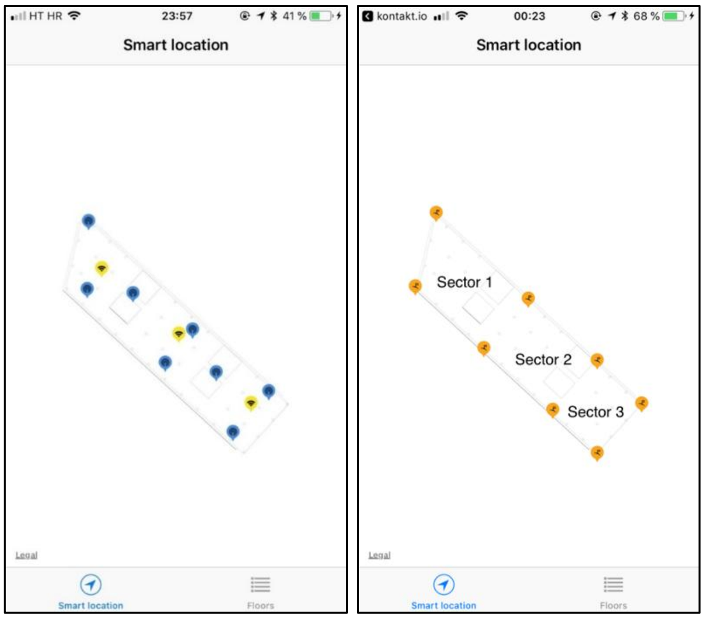


Figure 39: Placement of the transmitters (BLE and WiFi) and sectors (rooms)

In all cases it is necessary to describe in advance the transmitters and calculate their exact physical location within the building (coordinates in three dimensions, as the floor on which the user is needs to be determined). Apart from providing precise position for each transmitter we need a method to describe rooms (which is done by defining their corners) and a method to provide the floorplan of the indoor space, as without visual and symbolic name it is difficult to understand the indoor (Figure 39).

5.3.1.2 Compliance Level

Global Location Enabler and Specific Location Enablers register as a service in symbIoTe core. Search in symbIoTe core is used to discover positioning service providers (Specific Enablers) for certain area of interest. Therefore this solution is considered as L1-compliant, thought it does not register any sensors or resources as a typical L1-compliant platform, but only registers the service it provides.

5.3.1.3 Platforms and Enablers

Due to the necessary modularity of the solution (which needs to discover and incorporate various infrastructures that exist at the certain location), an enabler design is used; depending on the location from which Clients contact, the top-level Global Location Enabler queries Specific Location Enablers which operate on the location. Another enabler (Symbolic Location Enabler) translates the physical location to symbolic location that is definitively more useful in the indoor (building, floor, room) than the exact physical location (Figure 40).

- *Smartphone Applications:* There are two types of smart phone applications, one for iOS and one for Android. They collect fingerprint (iOS just collects BLE beacons due to iOS limitations in accessing detailed network information) and send collected data to the proxy server or directly to the Global Location Enabler. They show current user location and ground plans for buildings in which the user is.
- *Proxy Server:* Proxy Server is not a mandatory part of this system. Android and iOS application can communicate directly with the Global Location Enabler but if in the future other functions want to be added it is better to have a unique endpoint and that is the purpose of this server. This server also has a web administrator application for adding rooms/spaces and transmitters into FROST server.
- *FROST:* FROST server is an implementation of SensorThings API and its database stores all information about transmitters and rooms. This data is used by the Symbolic Location Enabler to translate physical location to symbolic location. In our current implementation, FROST is also used by Specific Location Enablers as well, but they can also have their own separate databases describing transmitters and rooms (having SensorThings API compatibility is a useful and practical addition).
- *symbIoTe Core:* symbIoTe core is used by Global Location Enabler to find Specific and Symbolic Enablers for the last known user location (which is sent initially by the user Application). Symbolic Location Enabler and Global Location Enabler register location services into symbIoTe core (as they provide location service for certain area).
- *Global Location Enabler:* The Global location enabler has a function to combine results from various Specific Location Enablers that operate in the area and then pass that result to Symbolic Location Enabler. The Global location enabler returns collected information to Mobile Applications.
- *Specific Location Enablers:* Various types of specific location enablers exist. This use case has up to three Specific Location Enablers (one for BLE infrastructure, one for WiFi and one for mobile network infrastructure). The purpose of the Specific Location Enabler is to calculate user location from RSSI using trilateration.
- *Symbolic Location Enabler:* Symbolic Location Enabler uses absolute location to determine symbolic user location. By using the data stored in FROST server (which describe buildings, floors and spaces) it can translate a physical location (coordinates and heights) to a symbolic name which is returned to the application.

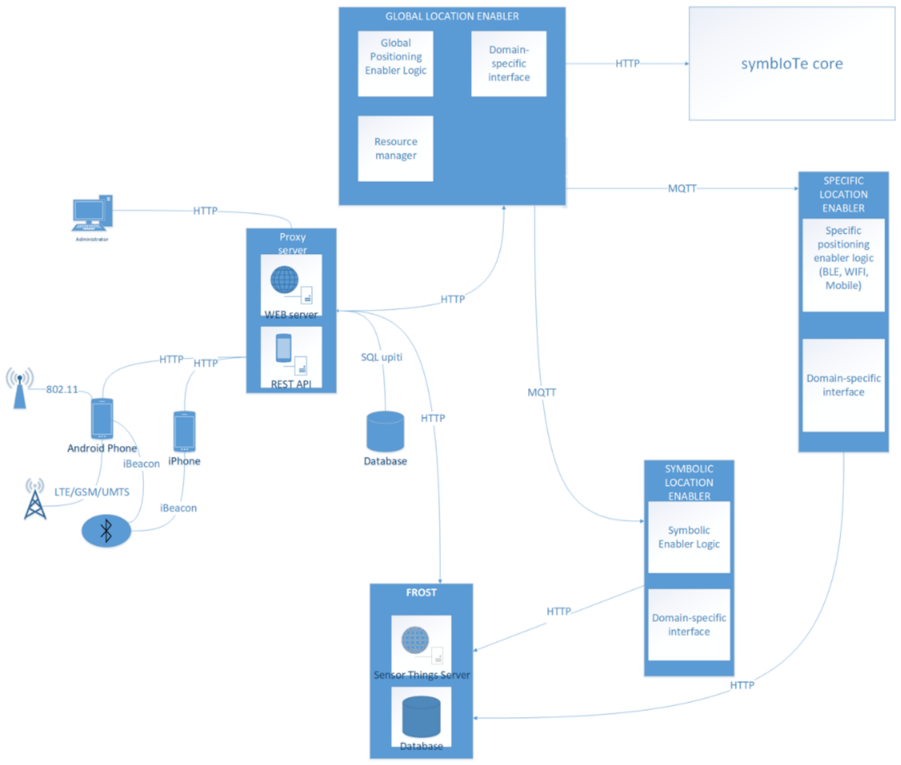


Figure 40: Architecture of the Indoor Positioning enabler system

In this initial implementation, one FROST database is used to store transmitter and room (sector) data, which is then used by both Specific Location Enablers and Symbolic Location Enablers. In the general case, each Specific Location Enabler can have its own geospatial database, which is preferably automatically populated from the infrastructure management system (BLE beacon database, WiFi network controller or mobile network infrastructure database).

5.3.1.4 End-User Interaction

Android (Figure 41) and iOS application (Figure 42) follow similar logic; they display the floorplan of the identified floor overlaid over supported map (Google Earth or Apple Maps) with user location indicated by a pin. Apart from the position on the floorplan, a symbolic location is shown as a text description (room ID or position description). In multi-floor buildings, floorplans of all floors can be seen with their symbolic designation and designation of “active” floor (where User is currently located) by selecting the Floors tab.

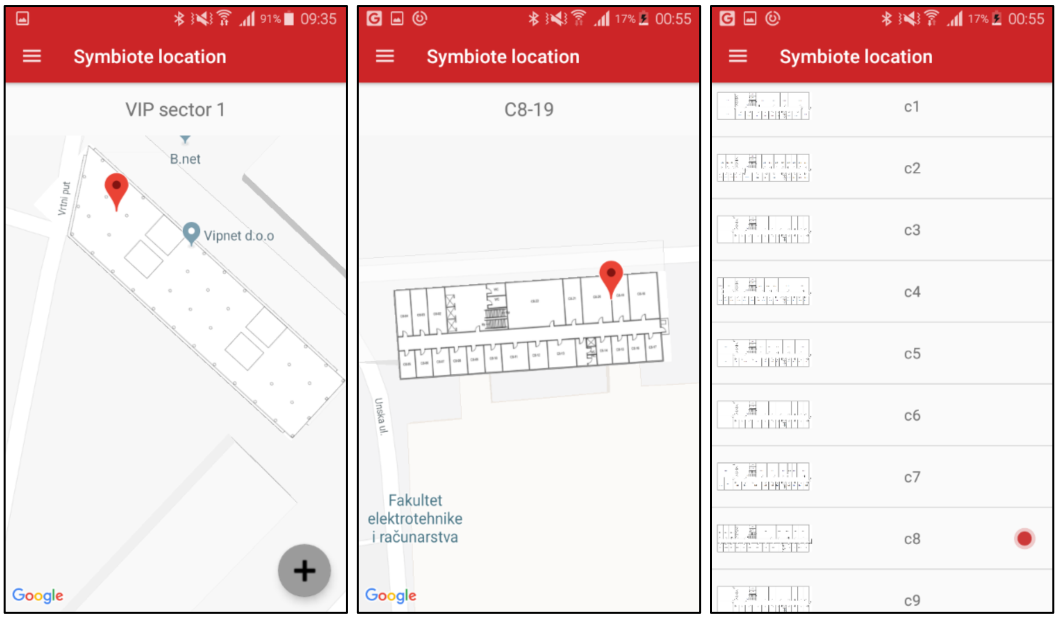


Figure 41: Android application (VIP and UNIZG-FER campus)

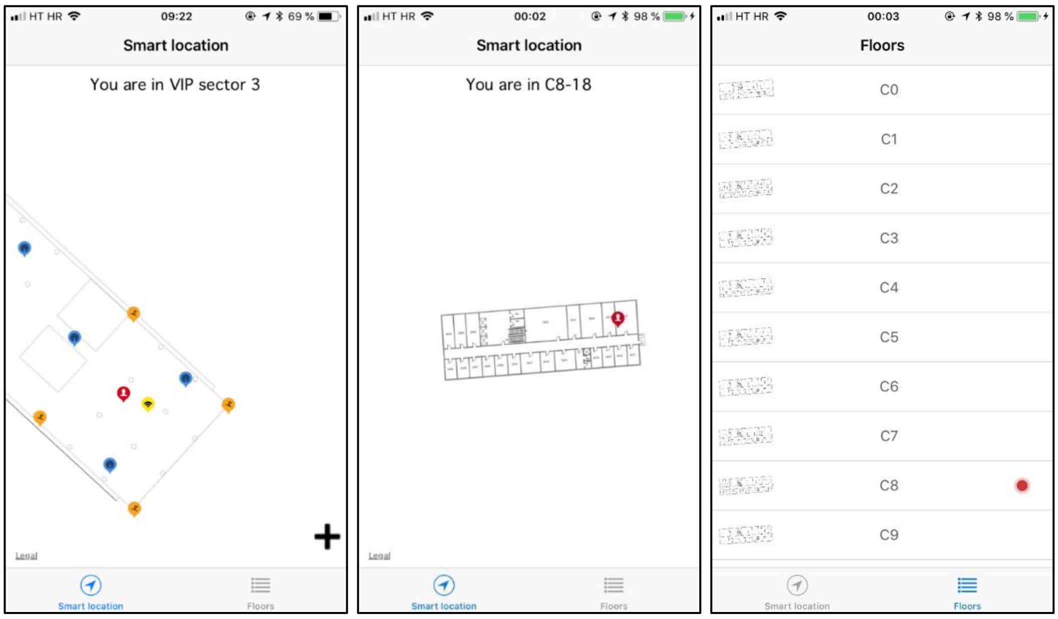
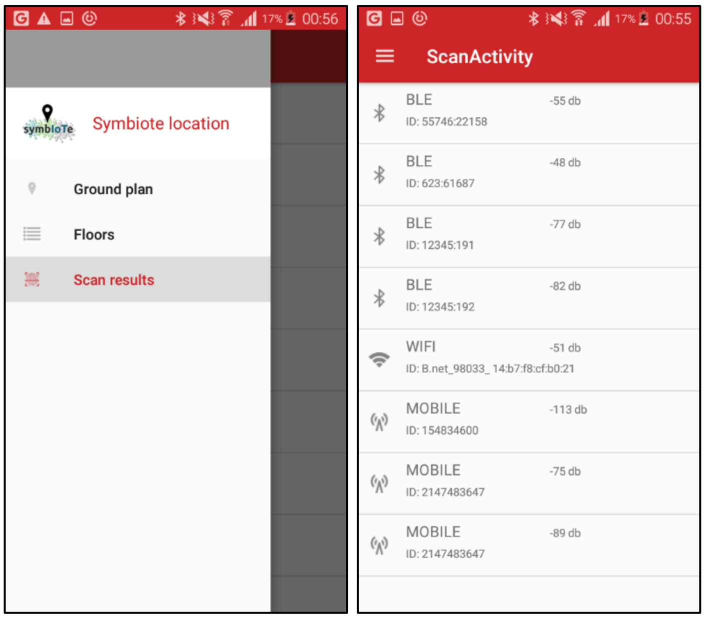


Figure 42: iOS application (VIP and UNIZG-FER campus)

An additional feature of the Android application is the “Scan result” tab (Figure 43) which can be used for simple and quick analysis of surrounding network infrastructures as it can show BLE beacons, WiFi access points and identification of mobile network cells to which the phone is attached. By providing the additional information in FROST (or other database storing transmitter information) the application can be used for additional scanning and testing of the performance and quality of the positioning information.



BLE ID: 55746:22158	-55 db
BLE ID: 623:61687	-48 db
BLE ID: 12345:191	-77 db
BLE ID: 12345:192	-82 db
WIFI ID: B.net_98033_ 14:b7:f8:cf:b0:21	-51 db
MOBILE ID: 154834600	-113 db
MOBILE ID: 2147483647	-75 db
MOBILE ID: 2147483647	-89 db

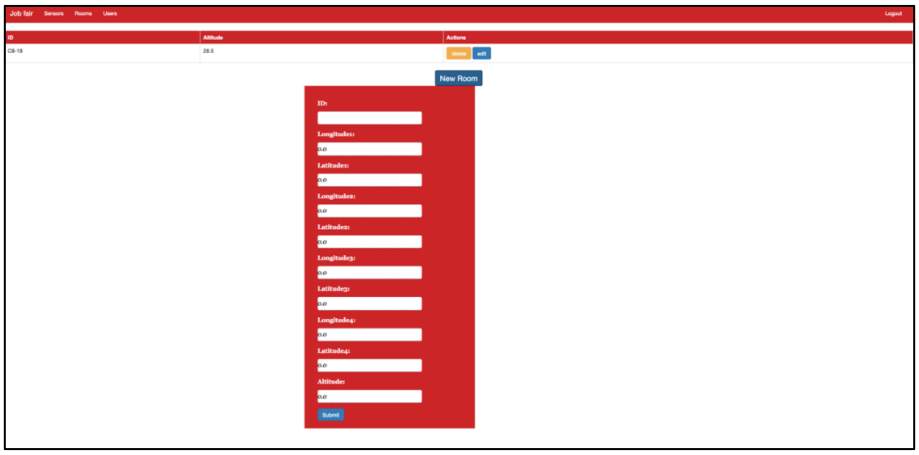
Figure 43: Wireless fingerprint and floorplans in Android Application

Absolute feature parity between iOS and Android is not possible, due to the limitation of iOS API which only allows reading out the BLE beacons from the surrounding area (by using iBeacon protocol), while Android allows access to both surrounding WiFi and mobile networks.

5.3.1.5 Backend User Interface

Simple web user interface is used to manage the data stored in the FROST server (containing room information and transmitter information). It includes three sections: Sensors (editing the information on BLE, WiFi and mobile network transmitters), Rooms (defining symbolic naming for the buildings, floors and rooms) and Users (defining additional users and user rights).

Room is defined by four corners (given with absolute latitude/longitude) and by the relative height of the room floor from the surrounding ground (convention of using absolute height is also possible). Room does not have to be rectangular but is described by four outside points, though in the case of some unusual room shapes, multiple quadrilaterals can be defined identifying the same room (Figure 44).



ID	Altitude	Actions
C8-18	28.5	delete edit

ID:

Longitude1: 0.0

Latitude1: 0.0

Longitude2: 0.0

Latitude2: 0.0

Longitude3: 0.0

Latitude3: 0.0

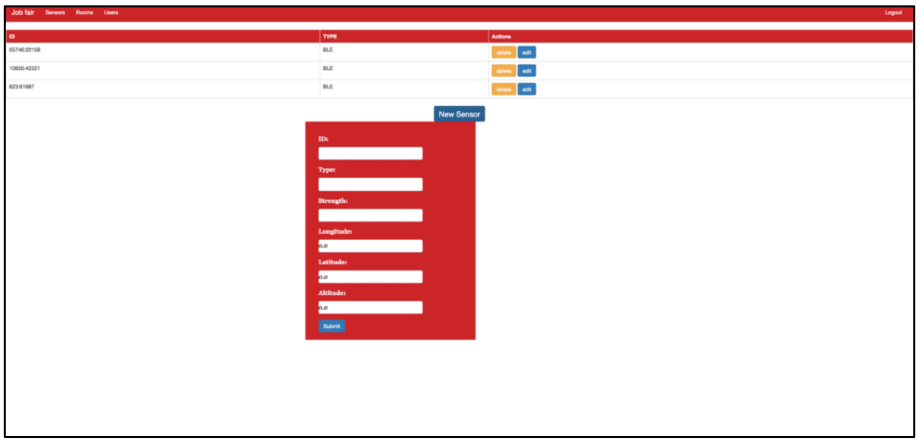
Longitude4: 0.0

Latitude4: 0.0

Altitude: 0.0

Figure 44: Entering Room information

Sensor (transmitter) is defined by its identification (major:minor ID for BLE beacons, MAC address for WiFi access point and CellID information for mobile network). Exact location (longitude and latitude) and altitude is required as it is needed to detect the floor on which the user is currently located. Information on transmit strength is needed for the RSSI measurement and trilateration algorithm to work properly (Figure 45).

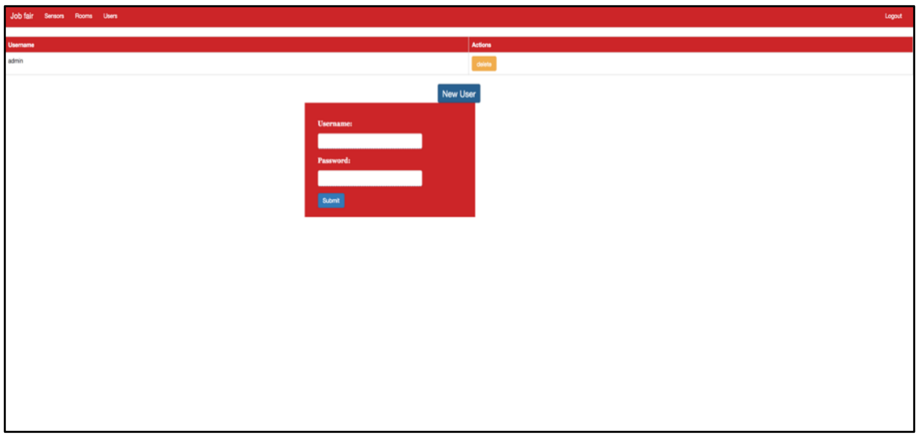


ID	TYPE	Actions
55746:22158	BLE	delete edit
10655:42221	BLE	delete edit
623:61687	BLE	delete edit

ID:	
Type:	
Strength:	
Longitude:	0.0
Latitude:	0.0
Altitude:	0.0

Figure 45: Entering and managing Sensor information

User tab allows for creation and deletion of additional users for the FROST backend system, allowing adding users with non-admin roles (for example just editing the Sensor or Room information (Figure 46).



Username	Actions
admin	delete

Figure 46: User management

In case of using other infrastructure database (not the one built on FROST) different backend user interface would be provided. The ideal option would be retrieving the information on transmitters directly from their management system where some description fields would store longitude/latitude/altitude information together with the transmitted signal strength. This approach would retrieve information from its source management systems without intervention and any changes to sensor location and characteristics would be immediately propagated for the Specific Location Enabler.

5.3.1.6 Implementation

Various tools and toolkits have been used to implement the elements of the solution, as it includes both iOS and Android smartphone applications and several server-side components.

- *FROST* is a Java implementation of SensorThings API that provides syntactic and semantic interoperability of Internet of Things. SensorThings API has a data model which has entities for storing location data and is developed for Internet of Things and interoperability. SensorThings follows REST principles, JSON data encoding, the OASIS OData protocol and URL conventions. SensorThings has also MQTT extension so IoT devices or users can publish subscription updates.
- *iOS Smartphone Application* is developed using Xcode IDE, written in Swift4 programming language using CocoaPods. Classical MVC architecture is used for developing the application. Due to iOS limitations, only iBeacon protocol and BLE beacons are supported.
- *Android Smartphone Application* is developed also in MVC model and is developed in Android Studio IDE using Java and Kotlin programming languages. BLE beacons are scanned by using Altbeacon library, while WiFi information is collected via WiFiManager library and mobile base station data via TelephonyManager library.
- *Global, Specific and Sybolic Location Enablers* are developed in IntelliJ IDEA using the Spring framework for development of web servers and applications. They also implement FROST client library to connect to FROST server.

5.3.1.7 Initial Functional Tests

Testing has been performed on two locations (VIP main campus and UNIZG-FER campus in Zagreb) and with two application implementations (Android and iOS) to demonstrate the implementations in the Smart Campus environment and perform the testing of the precision for the Indoor Positioning System (IPS). Testing on VIP campus was performed on 1400m^2 space (one floor of the main office building) where existing WiFi installation (with not usefully spaced access points) consisting of 3 access points was augmented by BLE beacons (Figure 47) positioned in what was considered a best practice for trilateration (three WiFi access points were augmented by 8 BLE beacons).

Figure 47: BLE Beacons used for the testing

Precision of the positioning (measured repeatedly on pre-determined test points) was between 1m and 3m, with application obtaining the new position within 5-10 seconds (Figure 44). Experience from the measurements was then re-applied to positioning of the beacons to improve the precision. General conclusion is that precision increases with adding more BLE beacons as they provide better transmitter granularity than WiFi as they can be easily moved around (WiFi access points are pre-installed to certain ceiling location while BLE beacons can easily be moved to better location). Android application showed somewhat better results than the iOS application, which is also an area of future testing and implementations.

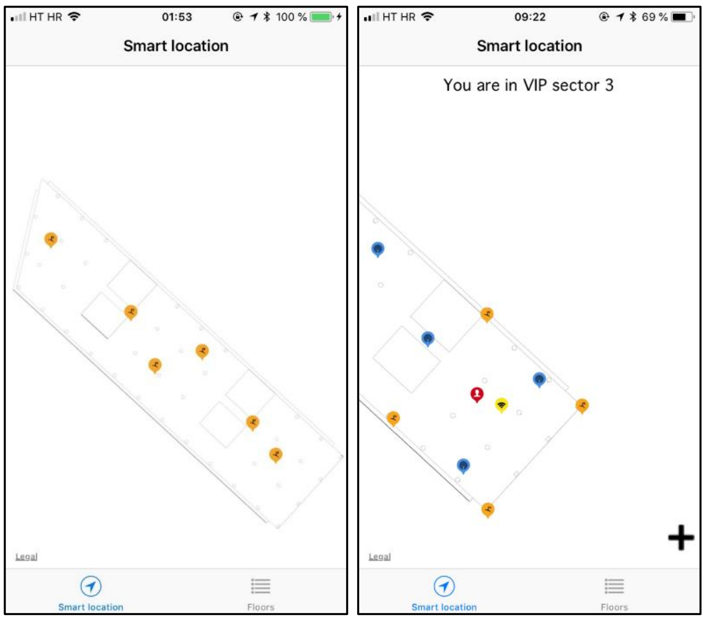


Figure 48: iOS application showing test points and measurement

Additional tests are planned in the Smart Stadium environment (closed indoor arena) where higher-precision mobile network is implemented with massive small cells design, where mobile network data would also provide good precision granularity. The general idea for the implementation would be to include these features in company-specific or location-specific smartphone applications.

6 Conclusions

This deliverable contains the report of the final release of symbIoTe software. One of the outcomes of this work is the source code and its documentation, published as an open source project in the GitHub service: `https://github.com/symbiote-h2020`. The main software components/libraries (26 in total) in the final release (Release 3.0.0) include 4 common libraries (symbIoTeLibraries, symbIoTeSecurity, symbIoTeSemantics, symbIoTeMapping) and one common component (Authentication and Authorization Manager), 10 core components (Administration, Cloud-core Interface, Core Interface, Core Resource Access Monitor, Core Resource Monitor, Registry, Search, Semantic Manager, Core Bartering and Trading Manager, Core Anomaly Detection), 9 platform components (Federation Manager, Monitoring, Platform Registry, Registration Handler, Resource Access Proxy, Subscription Manager, Trust Manager, Bartering and Trading Manager, SLA Manager) and 2 Smart Space components (Smart Space Middleware, SDEV SymbIoTe Agent). The components are organized in three GitHub super-repositories: SymbioteCore, SymbioteCloud and SymbioteSmartSpace. Four supporting projects are also used, CoreConfigService and CloudConfigService, as well as EurekaService and ZipkinService (with separate versions for core and cloud modes), which are located in separate repositories. The information models created within symbIoTe are located at the Ontologies repository.

Also, five use cases of symbIoTe that make use of the following applications have been identified:

- The *Smart Residence* applications that cover the indoor, house environment involving air quality control, health monitoring, and comfort and device control.
- The *Smart Mobility and Ecological Routing* web and mobile applications that provides green ecological routes based on data collected from several IoT platforms and analysed by the SMEUR enabler.
- The *EduCampus* scenario with the indoor location service for multiple campus solutions to ease a student's life by, e.g., allowing to book a room of navigate inside buildings.
- The *Smart Yachting* applications that aim to automate the mooring process of the port (Smart Mooring) and automatically identify the needs for goods and services on board of the Yacht (ASC).
- The *Smart Stadium* applications (Visitor, Retailer and Promowall applications) that involve the provisioning of location based services e.g., orders and purchases, offers, promotions, etc. to brings visitors and retailers at a stadium closer.

Most of the use cases utilize IoT platforms at L1 and L2 compliance levels, while Smart Residence and Smart Yachting use case scenarios also make use of the symbIoTe platform L3 and L4 functionality.

7 References

[1]The symbIoTe consortium. (2016). D1.2 – Initial Report on System Requirements and Architecture.

[2] The symbIoTe consortium. (2017). D5.2 - Report on System Integration and Application Implementation.

[3] The symbIoTe consortium. (2017). D1.4 – Final Report on System Requirements and Architecture.

[4] The symbIoTe consortium. (2016). D5.1 - Implementation Framework.

[5] The symbIoTe consortium. (2017). D1.3 - Final Specification of Use Cases and Initial Report on Business Models.

[6] The symbIoTe consortium. (2017). D2.6 - symbIoTe Domain-Specific Enablers and Tools.

8 Acronyms

AIT	Austrian Institute of Technology GmbH
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
APP	Application
ATOS	ATOS Spain SA
BIM	Building Information Modelling
BLE	Bluetooth Low Energy Beacon
CIM	Common Information Model
CLD	Cloud Domain (symbIoTe domain layer)
CPS	Cyber Physical Systems. A mechanism controlled or monitored by computer-based algorithms
CRAM	Core Resource Access Monitor
DAO	Data Access Object
DoA	Description of Action
DTO	Data Transfer Object
FER	Faculty of Electrical Engineering and Computer Science, University of Zagreb
H2020	“Horizon 2020” EU Research and Innovation Programme
HTTP	Hypertext Transfer Protocol
IAQ	Indoor Air Quality
ICOM	Intracom Sa Telecom Solutions
ICT	Information and Communication Technology
IOSB	Fraunhofer Gesellschaft zur Förderung der Angewandten Forschung ev
IoT	Internet of Things
JSON	Javascript Object Notation, a human readable data exchange format
JVM	Java Virtual Machine
KIOLA	Telehealth Service Platform
KIT	Karlsruhe Institute of Technology
LoRa	Long Range
LoRaWAN	LoRa Alliance Technology Low Power Wide Area Network
MIM	Minimum Information Model
MoBaaS	Mobility Backend as a Service

MVC	Model View Controller
M2M	Machine to machine
NAVIGO	Na.Vi.Go. Societa Consortile a Responsabilita Limitata
NXW	Nextworks
OData	Open Data Protocol, an open protocol to allow the creation and consumption of queryable and interoperable RESTful APIs
openUwedat	AIT’s platform to manage observations and related data
PIM	Platform Information Model
POI	Point of Interest
RAP	Resource Access Proxy
RDF	Resource Description Framework, a description standard for semantical relations
REST	REpresentational State Transfer
RH	Registration Handler
S&C	Sensing & Control Systems SL
SD	Device Domain (symbIoTe domain layer)
SMILA	Smart Mirror Integrated Living Assistant
SPARQL	A query language for semantically linked data sets (see RDF)
SSP	Smart Space Domain (symbIoTe domain layer)
symbIoTe	Symbiosis of Smart Objects across IoT Environments
S3M	symbIoTe Smart Space Middleware
ToC	Table of Contents
UNIDATA	Unidata Spa
UNIVIE	Universität Wien
UNIZG-FER	Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
URL	Uniform Resource Locator
UW	Ubiwhere Lda
VIP	Vipnet d.o.o.