D4.2.2 – FINAL SYSTEM PROTOTYPE AND USER MANUAL

Release 1.0

2014.08.01

Short Description

D4.2.2 will update D4.2.1 taking into account the lessons learned from the first evaluation cycle

Authors - Contributors

Almende, UnivPM, Fraunhofer, VTT, UniKassel, Cofely, CNet Svenska, UPC

EXECUTIVE SUMMARY

The purpose of this deliverable is to report on the final SEAM4US system prototype, and it includes a thorough description of the architecture as well as an extensive explanation, aimed at users and developers, about how to setup the system. This document is an updated version of D4.2.1 (delivered at month 24), which presented only a preliminary version of the system since several components still had to be finalized and integrated.

The overall result of the combined effort of the consortium partners can be considered satisfactory. The system has been built according to the specifications of D4.1 ('*System specification report*', delivered at month 12) and no considerable divergence from the expected outcome needs to be reported.

The synergistic development of the different components has led to a smooth integration phase, which has been successfully carried out during the last year of the project. This result is particularly remarkable: such a complex system made of heterogeneous components developed sometimes from scratch (e.g., the Graphical User Interface and the CCTV-based crowd density estimator) and other times tailoring existing products (e.g., the LinkSmart middleware) works effectively to perform an intelligent control of the devices (fans, lights, and escalators) of the pilot station of Passeig de Gracia - L3. All technical (such as the interfaces with the existing TMB HW/SW systems) and non-technical (such as the legal aspects) issues have been successfully tackled.

It is important to emphasize that the only interaction between the SEAM4US users (i.e., project engineer and station manager, as defined in D4.1) and the system is via a straightforward Graphical User Interface, which is fully documented in this deliverable. However, we have decided to add additional information regarding the setup of the system, such as a user manual for the LinkSmart middleware and for the CCTV-based crowd density estimator. This choice has been made to encourage the reuse of the components in future projects as well as to document in a transparent way the work done.

Finally, this deliverable complements two other deliverables: D3.2.2 ('*Final user, thermal, and control models*'), which describes the details of the two main subsystems of SEAM4US ('Control' and 'Models') as well as the 'Monitoring' subsystem, and D5.1.2 ('*Final energy management system deployment handbook*'), which describes all details of the hardware. The three of them together give a comprehensive vision of all the hardware, software, and algorithmic parts of the system. In order to facilitate the reading, this deliverable is relatively short, as we have preferred conciseness to redundancy and references to repetitions. The result is a succinct report, which thoroughly illustrates the more relevant aspects of our work; the less relevant, yet interesting, details (such as the description of the events and the methods) have been moved to dedicated appendices.

CONTENTS

EXECUTIVE SUMMARY II

CONTENTS III

FIGURES V

TABLES VII

1. INTRODUCTION 8

2. FINAL SEAM4US SYSTEM ARCHITECTURE 9

3. LINKSMART MIDDLEWARE 12

3.1. NETWORK MANAGER 12

3.2. SUPERNODE 13

3.3. EVENT MANAGER 14

4. MONITORING PROXIES 15

4.1. ENVIRONMENTAL MONITORING NETWORK PROXIES 15

4.1.1. Setup of the AMPASE platform in SEAM4US *15*

4.1.2. Environmental sensor proxies *16*

4.1.3. Sensor network management UI *17*

4.1.4. Software and protocols *18*

4.1.5. Time synchronization *19*

4.1.6. Duty cycling *19*

4.1.7. Routing *21*

4.2. CCTV PROXY AND CCTV-BASED CROWD DENSITY ESTIMATOR 21

4.2.1. Working principles *22*

4.2.2. Calibration *22*

4.2.3. Algorithms for crowd density estimation *24*

4.2.4. Using the crowd density estimator *27*

4.3. TRAIN ARRIVAL PROXY 27

4.4. WEATHER FORECAST PROXY 28

4.5. ENISTIC METERS PROXY 28

4.6. SOCOMEC METERS PROXY 30

4.7. SACI METERS PROXY 31

5. ACTUATOR PROXIES 33

5.1. FAN PROXY 33

5.2. LIGHT PROXY 34

5.3. ESCALATOR PROXY 34
6. GRAPHICAL USER INTERFACE 35
6.1. STATION MAP TAB 36
6.2. MONITORING DATA TAB 38
6.3. CONTROL TAB 40
7. SUPERVISION 41
7.1. ARCHITECTURE 41
7.2. THE *SUPERVISABLE* INTERFACE 42
7.3. WORKING PRINCIPLES 43
8. REPRESENTATIONS 45
8.1. SPATIAL REPRESENTATION 45
8.2. EVENT REPRESENTATION 46
9. DATABASE AND DATABASE MANAGEMENT 48
10. CONCLUSIONS 51
APPENDIX A - EVENT FORMAT 52
A.1 RAW DATA 52
A.1.1. Raw data for monitoring 52
A.1.2. Raw data for actuators 57
A.2 POSTPROCESSED DATA 59
A.2.1. Postprocessed data for monitoring 59
A.2.2. Postprocessed data for control 66
A.3. DB ACCESS EVENTS 67
APPENDIX B - METHODS 69
B.1 ACTUATORS 69
B.3 LINKSMART 71
GLOSSARY AND ABBREVIATIONS 73

FIGURES

Figure 1. Final SEAM4US system architecture 9
Figure 2. LinkSmart configuration page 13
Figure 3. Setup of the SEAM4US environmental monitoring network 16
Figure 4. Sensor network device proxy GUI 17
Figure 5. Environmental monitoring network management UI 18
Figure 6. AMPASE main network protocols timeline 20
Figure 7. Battery replacement interval of nodes 21
Figure 8. Flowchart of the calibration phase of the crowd density estimation algorithms 22
Figure 9. GUI for the initialization of the ROI (left) and the perspective correction (right). 23
Figure 10. (left) Input frame used as input image in the rest of this section; (right) trained background. 23
Figure 11. Flowchart of the crowd density estimation algorithm. 24
Figure 12. Binary image of the foreground (left) before and (right) after noise removal 24
Figure 13. (left) Output of 'Canny edge detection' on the combined RGB channels (right) AND operation of the Canny edge detection and the foreground image after noise removal, see Figure 12. 25
Figure 14. (left) Logic OR between the foreground mask and the edge image; (right) dilated and eroded image in which blobs of people are segmented. 25
Figure 15. Examples of CCTV mulfunctioning: (left) occluded camera; (right) scratched camera 26
Figure 16. Topology of the existing CCTV system deployed at PdG-L3 station 26
Figure 17. SQLite event storage 30
Figure 18. SQLite Key/Value storage 30
Figure 19. Communication between the SEAM4US Server and a SOCOMEC meter via the TMB network . 30
Figure 20. Differences between Modbus TCP and Modbus RTU messages 31
Figure 21. SOCOMEC address map 31
Figure 22. Communication between the SEAM4US Server and a SACI meter via the TMB network 32
Figure 23. Fan proxy communication between the server and the new devices 33
Figure 24. User identification screen to access the GUI: each SEAM4US role has different credentials 35
Figure 25. Example of check on the GUI functionality 35
Figure 26. 'Station map' tab showing the position and values of all sensors 36
Figure 27:. 'Station map' tab showing the position and values of selected sensors 37
Figure 28. Historical values of a selected sensor 37
Figure 29. The chart of the historical values can also be printed or downloaded in multiple formats 38
Figure 30. Example of historical values for a consumption sensor 38
Figure 31. Values of one sensor in the 'monitoring data' tab 39
Figure 32. Values of multiple sensors in the 'monitoring data' tab 39
Figure 33. Example of the SEAM4US GUI 'control' tab 40
Figure 34. Architecture of the SEAM4US system supervisor component 41
Figure 35: Static data structure of the supervisable interface 42
Figure 36: Workflow of a supervision cycle 43
Figure 37. Example of a supervisor alert mail 44
Figure 38. SEAM4US nomenclature applied over the station model 45
Figure 39. Topology of the PdG-L3 station represented with the SEAM4US spatial zones 46
Figure 40. Structure of the SEAM4US storage DB 48

Figure 41. EventItem DB table *48*
Figure 42. Context DB table *48*
Figure 43. ObservableProperty DB table *49*
Figure 44. Observation DB table *49*
Figure 45. Event flow through the EventManager *49*

TABLES

Table 1. Messages of the AMPASE platform 18
Table 2. Template for the event format of SEAM4US raw data 47
Table 3. Raw data format for the solar radiation 52
Table 4. Raw data format for the anenometers 53
Table 5. Raw data format for the differential pressure 53
Table 6. Raw data format for the absolute pressure 54
Table 7. Raw data format for the PM10 (lower concentrations) 54
Table 8. Raw data format for the PM10 (higher concentrations) 54
Table 9. Raw data format for the CO2 55
Table 10. Raw data format for the temperature 55
Table 11. Raw data format for the humidity 56
Table 12. Raw data format for the power consumption 56
Table 13. Raw data format for the occupancy 57
Table 14. Raw data format for the train arrivals 57
Table 15. Raw data format for the escalator speed 58
Table 16. Raw data format for the fan frequency 58
Table 17. Raw data format for the fan control status 59
Table 18. Postprocessed data format for the air change rate 59
Table 19. Postprocessed data format for the air flow rate 60
Table 20. Postprocessed data format for the air speed 60
Table 21. Postprocessed data format for the PM10 61
Table 22. Postprocessed data format for the CO2 61
Table 23. Postprocessed data format for the absolute pressure 62
Table 24. Postprocessed data format for the temperature 62
Table 25. Postprocessed data format for the humidity 63
Table 26. Postprocessed data format for the power consumption 63
Table 27. Postprocessed data format for the passenger occupancy 64
Table 28. Postprocessed data format for the frequency of the trains 64
Table 29. Postprocessed data format for the weather station 64
Table 30. Postprocessed data format for the weather forecast 65
Table 31. Postprocessed data format for the fan frequency 66
Table 32. Postprocessed data format for the fan control status 67
Table 33. Postprocessed data format for the update of the passenger occupancy 67
Table 34. Format for the DB request 67
Table 35. Format for the DB response 68

1. INTRODUCTION

The finalization of the SEAM4US system has required a considerable effort from all partners, yet the outcome is rewarding: the current system has been fully integrated and tested. The results of its application to the pilot are currently under evaluation, and they will be reported in the deliverables of WP6. Remarkably, the project has succeeded in building up an effective architecture starting from some very limited existing resources in terms of HW (e.g., the AMPASE platform developed by VTT) and SW (e.g., the LinkSmart middleware developed by CNET and FIT) thanks to the synergistic work of the partners, which have contributed with their expertise to steer the implementation to a scientifically sound and technically innovative direction.

This deliverable describes in detail the components of the final architecture (except for those reported in D3.2.2 and D5.1.2, as it will be explained below) and illustrates the rationale behind it. The result is a robust distributed system that not only works effectively but also interacts smoothly with the existing TMB subsystems, such as the TMB SCADA (also called with the Spanish acronym TMB CCIF) and the actuators already deployed at the pilot station. The information given in this deliverable updates the one presented in D4.2.1 and complements the one presented in D3.2.2 and D5.1.2. It is also important to emphasize that this deliverable describes the Graphical User Interface, which is the only way in which the SEAM4US users (i.e., project engineer and station manager, as defined in D4.1) can interact with the system (additional interfaces have been built for the developers).

The structure of this document reflects the one of the architecture. Therefore, it starts with a brief overview of the final SEAM4US architecture in Sec. 2, and it follows with one section per subsystem, except for the monitoring network and the actuators (which have been described in D5.1.2) and the monitoring, control and the models (which have been described in D3.2.2). Therefore, Sec. 3 is about the LinkSmart middleware and its setup; Sec. 4 gives the details of the monitoring proxies; Sec. 5 discusses the actuator proxies; Sec. 6 concerns all aspects of the Graphical User Interface. The rest of the document is devoted to the parts of the system that are not directly involved into the definition of the best control policies, but that are necessary for having a working system. In particular, they are the supervisor, discussed in Sec. 7; the representations of space and events, discussed in Sec. 8; the database and the database management, discussed in Sec. 9. As usual, the last section, Sec. 10, includes the conclusions and the final remarks.

2. FINAL SEAM4US SYSTEM ARCHITECTURE

The SEAM4US architecture was first specified in D4.1 but it has undergone a revision, whose result in shown in Figure 1.

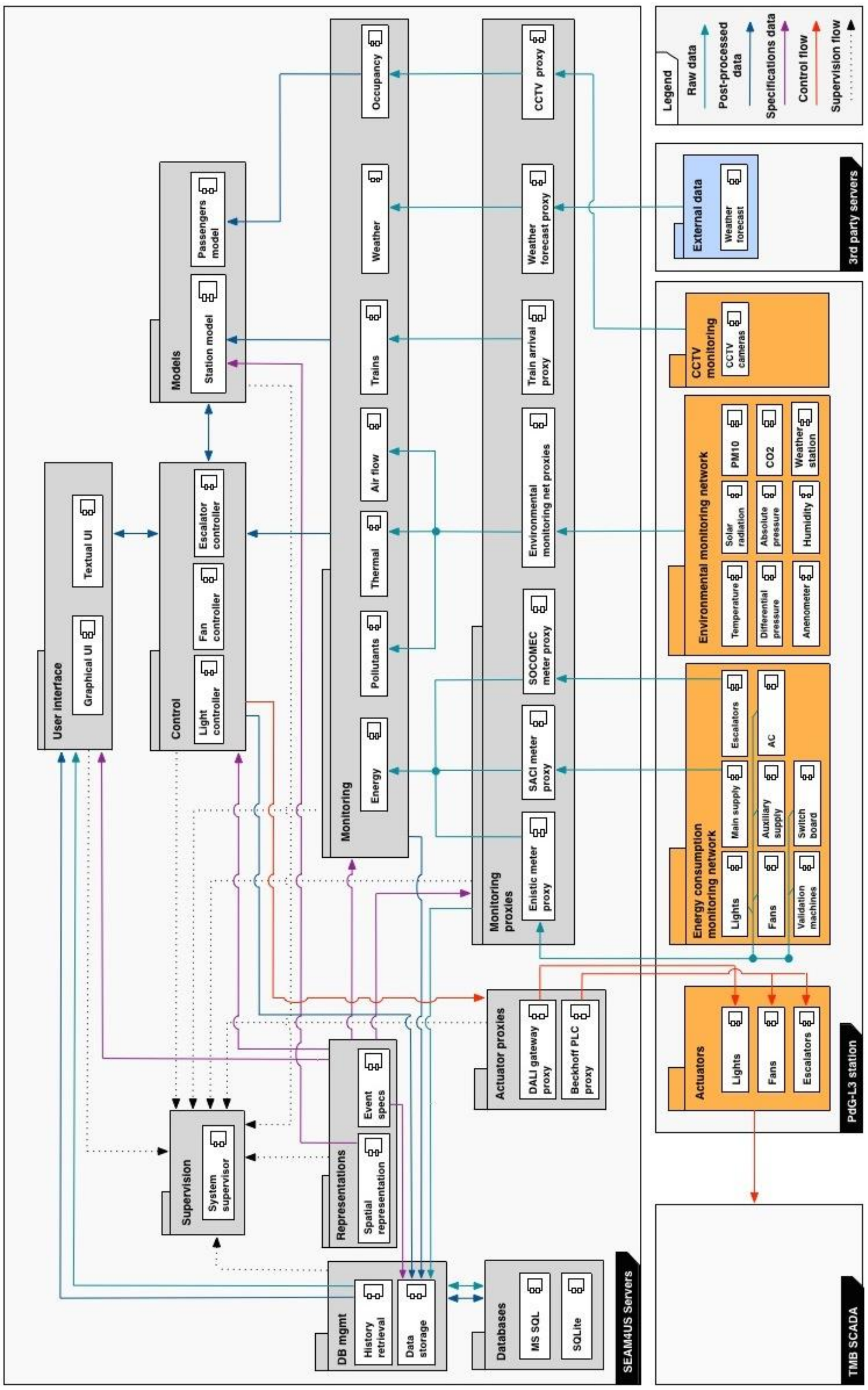


Figure 1. Final SEAM4US system architecture

The final version is composed of several subsystems, which are described in the rest of this deliverable. However, in the following we give a general overview of each subsystem, in order to illustrate in short the working principles of the SEAM4US system.

SEAM4US relies on several data sources, most of which are deployed at the Passeig de Gràcia – Line 3 (PdG-L3) station (*energy consumption network*, *environmental monitoring network*, and *CCTV monitoring*) and one of which is provided by a 3rd party (i.e., *weather forecast*). The main function of the whole monitoring network, whose details have been described in D5.1.2, is to collect information about the station, the passengers, and the external environment. A summary of the sensors and devices of the monitoring network is reported in Appendix A.

The data gathered from the sources are then processed by the *monitoring proxies* (see Sec. 4), which are software modules (of different complexities) that interface the 'real world' with the SEAM4US system. In general, they provide some methods to access the monitoring data and map them into the standard SEAM4US format (see Appendix B).

The next layer of the architecture contains all *monitoring* components that filter and aggregate the data of the monitoring network. This subsystem is composed of six components: *energy, pollutants, thermal, air flow, trains, weather, occupancy*.

The next layer of the architecture contains the two most important subsystems of the architecture, both of them described in D3.2.2. The first one is the *models* that contains two components: the s*tation model*, which includes a calibrated model of the station based on the real data coming from the lower layers, and the *passenger model*, which includes a model of the passenger occupancy and flows. The other main SEAM4US subsystem is the *control* that, based on the inputs of the monitoring and the models, calculates the best (in terms of energy consumption and passenger comfort) parameters of the controlled devices. The *control* can also disable other subsystems in case of malfunctioning.

The interface between the *control* subsystem and the *actuators* is performed by the *actuator proxies* (see Sec. 5), which map the SEAM4US control events into the appropriate control signals required by the actuators.

Besides those mentioned above, the SEAM4US system relies on additional components that are used to interact with the users (i.e., the *graphical user interface*, see Sec. 6), monitor the SEAM4US subsystems (i.e., the *supervisor*, see Sec. 7), describe the representation of space and events (i.e., the *representations*, see Sec. 8), and store/retrieve data (i.e., the *database* and the *database management*, see Sec. 9).

The interface among all subsystems is performed by the *LinkSmart middleware* (see Sec. 3). The SEAM4US components pull or push information to other components resulting in a loose coupling, which is a key requirement for such a highly distributed system. Pulling information is done either through OSGi interfaces or through web service calls; both are inherent properties of *LinkSmart*. Pushing information is done through the publish/subscribe functionality, which is provided by the *LinkSmart Event Manager*; of course, data can also be pushed via remote procedure calls, using setter methods. These characteristics, together with the networking functionality (*LinkSmart Network Manager*), make LinkSmart the ideal 'glue' of the system, stretching throughout all architectural layers.

3. LINKSMART MIDDLEWARE

Several components of the LinkSmart middleware (which is the main result of the Hydra project[1]) have been used in SEAM4US. Some of them could be used right away; others had to be extended to fit our system. Also, new technologies, which can be also framed within LinkSmart, have been developed.

As central components for the data distribution, we have used the original *Network Manager* as well as the *Supernode*. Several device proxies have been implemented, which include proxies for environmental monitoring network and energy consumption monitoring network as well as proxies for the weather forecast service and the CCTV subsystem. We extended the original LinkSmart proxy concept from a pure translator to an encapsulated representation of the sensor. This includes hiding push or pull communication to the sensor as well as administering meta-information, such as the location. The *Event Manager* is the component that has been extended most: apart from revising the event structure, we added functionalities for storing every sent event to a database. In order to make the *Network Manager* and the *Event Manager* more independent from each other, we have also added subscription persistence to both components.

In the following sections, we describe the functionalities and the configuration of the three LinkSmart components used in SEAM4US: *Network manager*, *Supernode*, and *Event manager*.

3.1. Network manager

The core LinkSmart components are packed as a binary distribution .zip, which can be downloaded, unpacked, and used as OSGi target platform. The source code can also be downloaded and complied from the LinkSmart repository at

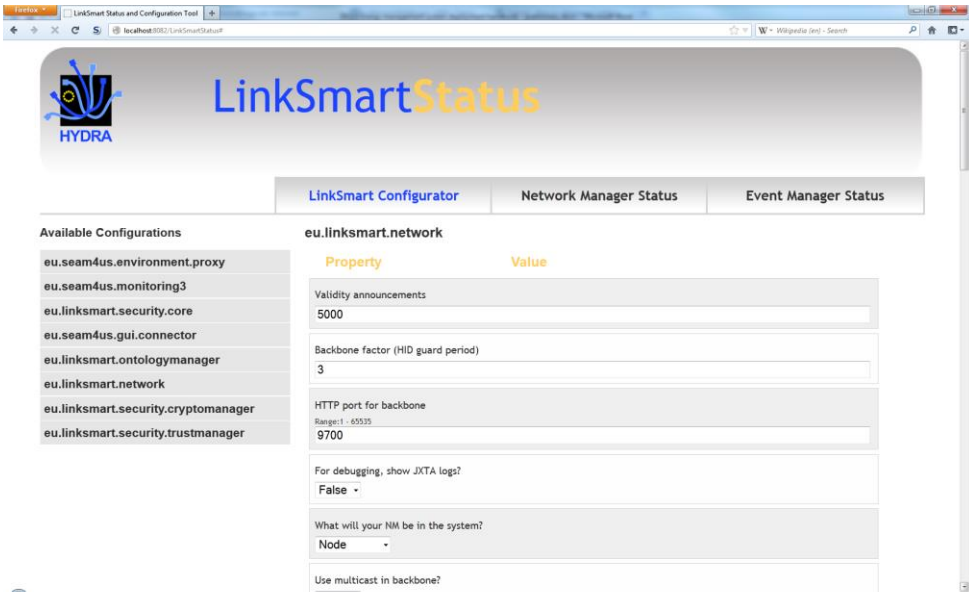
```
https://svn.code.sf.net/p/LinkSmart/code
```

To unfold the full power of security, LinkSmart makes use of the Java Cryptography Extension for unlimited strength encryption. In order to enable this feature, the following needs to be done:

- Download Java Cryptography Extension Unlimited Strength Jurisdiction Policy Files 6
- Unpack and copy the two files `local_policy.jar` and `US_export_policy` to the following location depending on your operating system
  - On Windows, if you have both, a JDK and JRE installation, copy the files to both: `%PATH_TO_JDK\jre\lib\security` and `%PATH_TO_JRE\lib\security`
  - On MacOS X, the security options should be OK as shipped with the OS. If you have any issues, copy the above mentioned files to `$JAVA_HOME/lib/security`

[1] The Hydra project is co-funded by the European Commission within the Sixth Framework Programme in the area of Networked Embedded Systems under contract IST-2005-034891. See also http://www.hydramiddleware.eu/

LinkSmart can be executed thanks to a start-up script which is provided both for Windows (.bat) and MAC (.sh). Alternatively, LinkSmart can be started within Eclipse. If everything has been set up correctly, then a website like the one shown in Figure 2 will appear if the configuration page `http://localhost:8082/LinkSmartStatus` is called from the browser.



Available Configurations

- eu.seam4us.environment.proxy
- eu.seam4us.monitoring3
- eu.linksmart.security.core
- eu.seam4us.gui.connector
- eu.linksmart.ontologymanager
- eu.linksmart.network
- eu.linksmart.security.cryptomanager
- eu.linksmart.security.trustmanager

eu.linksmart.network

Property	Value
Validity announcements	5000
Backbone factor (HID guard period)	3
HTTP port for backbone (Range:1 - 65535)	9700
For debugging, show JXTA logs?	False
What will your NM be in the system?	Node
Use multicast in backbone?	

Figure 2. LinkSmart configuration page

On the left side of the configuration page, we find a configuration option for each component that has enabled it. For instance, the *Network Manager* configuration can be accessed in the 'eu.LinkSmart.network' tab where the *Network Manager* name or HID can be modified. The top tab called 'Network Manager Status' shows an overview of the *Network Managers* in the same LinkSmart network plus detailed information like HID or IP address. These are the two options that should be usually configured when setting up LinkSmart on a computer for the first time in order to make the *Network Manager* individual. The HID will be used to find the *Network Manager* in the LinkSmart network. An overview of the provided services can also be requested.

3.2. Supernode

The *Supernode* bundle is available at

```
https://svn.code.sf.net/p/LinkSmart/code/components/NetworkManagerSuperNode
```

A run script (`run_bg.sh`) can be found in the main folder. After starting the bundle, a directory named `NetworkManagerSuperNode` is created. Going to the `config` folder, there is the configuration file `NM.properties`, which consists of two lines to configure the external address where the *Supernode* will be located:

```
extAddrHttp=<ip_address>:<port>
```

and

```
extAddrTcp=<ip_address>:<port>
```

There are certain ports that must be opened in the NAT/Firewall for the server IP address. Ports can be changed. The last step is to configure the `seeds.txt` of all the *Network Managers* that shall connect to the supernode with the seeds:

```
http://<extAddrHttp>
```

and

```
tcp://<extAddrTcp>
```

defined in `NM.properties`.

We added a call of `run_bh.sh` to `/etc/rc.d/rc.local` in order to start the *Supernode* automatically after reboot.

3.3. Event manager

The *Event Manager* is available at:

```
http://svn.code.sf.net/p/LinkSmart/code/branches/1.2/components/DotNetCoreMidd
leware/EventManager/
```

This version needs to be compiled first in a normal .NET/Visual Studio environment. The binaries can be found in the subfolder `\EventManager\bin\Debug`

No matter where one got the binaries from, the *Event Manager* has to be configured first by opening the file `EventManager.exe.config` and modify the description name of the *Event Manager* by changing the value of the `EventManagerDesc` setting. A name that is unique in the network should be used (e.g., `EventManager:<yourName>)`. If one is not using a *NetworkManager* on the same machine, the address of the NetworkManager also needs to be changed by adjusting the value of the `EventManager_NetworkManager1_NetworkManagerApplicationService` setting.

Start-up:

- Start the LinkSmart OSGi Configuration, including the *Network Manager* and your publisher and subscriber bundles
- Start the *Event Manager* last (and make sure to start it as an administrator)
- The *Event Manager* first shows an empty window, then it displays the subscriptions
- Published events are displayed as they are arriving at the *Event Manager*. They are now distributed to the subscribers

For publishing and subscribing to events from OSGi, there are two wrapper libraries (`EventPublicationWrapper` and `EventSubscriptionWrapper)` that provide an API that allows to subscribe to a particular topic at a configurable *Event Manager* and publishing an event to an *Event Manager*, respectively, with a one-line command.

4. MONITORING PROXIES

SEAM4US proxies integrate low-level technologies and provide access to devices and subsystems via the LinkSmart web service layer; they can be developed either in Java or .NET. Thanks to this approach, in principle the SEAM4US components can be used in other environements besides the pilot project, since with LinkSmart every proxy translates whatever underlying technology it abstracts into web services.

As shown in Figure 1, there are seven different kinds of monitoring proxies: the environmental monitoring proxies (described in Sec. 4.1), the CCTV proxy (described in Sec. 4.2), the train arrival proxy (described in Sec. 4.3), the weather forecast proxy (described in Sec. 4.4), and the three proxies of the 'smart meters' of the energy consumption monitoring network. In particular, the Enistic meters (described in Sec. 4.5), which measure lightning, ventilation, switch board, AC validation machines, and auxiliary supply; the SOCOMEC meters (described in Sec. 4.6), which measure the escalator movement; the SACI meters (described in Sec. 4.7), which measure the main supply (a throughout description of the devices of the energy consumption monitoring network is provided in D3.2.2).

4.1. Environmental monitoring network proxies

In the SEAM4US environmental monitoring, the challenge is to provide a large multihop sensor network in which the maintenance costs due to battery replacements and network management is optimized. For this purpose, we have chosen to use wireless sensoring, which is often needed when appropriate communication and power infrastructure are not available or the sensoring is temporary. During SEAM4US, the consortium has carried out some research on wireless sensor networks (WSN) regarding beyond state-of-the-art energy-efficient solutions, specifically by dual-radio approach. Nevertheless, since the developed solutions are not mature enough for large scale deployment in the pilot project, we have resorted to deploy a more conventional solution called AMPASE which has been adjusted to the project needs. For the sake of the readers', this section includes not only an explaination about the proxies, but also a decription of the main characteristics of the AMPASE platform.

4.1.1. Setup of the AMPASE platform in SEAM4US

AMPASE is a platform for advanced wireless sensoring developed by VTT. It provides low maintenance costs, easy setup and management interfaces, automatic recovery, alert functions, and low energy consumption protocols and hardware. The examples of enabling features are online parameter changing for each sensor node independently for measurement interval, transmission interval and encryption as well as over-the-air-programming to re-program sensor nodes remotely. The SEAM4US environmental monitoring network setup is shown in Figure 3.

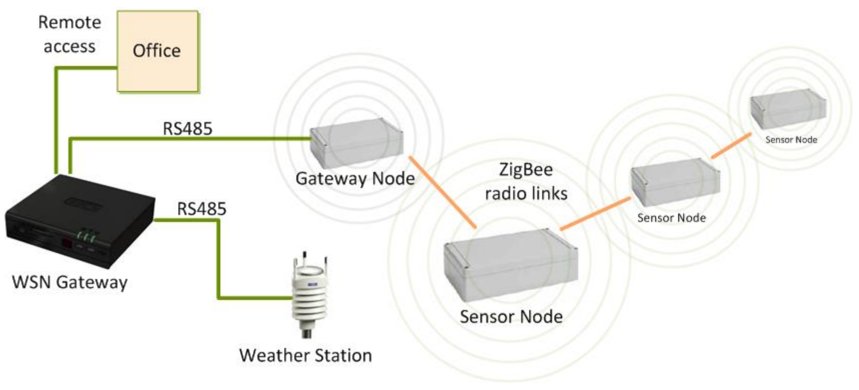


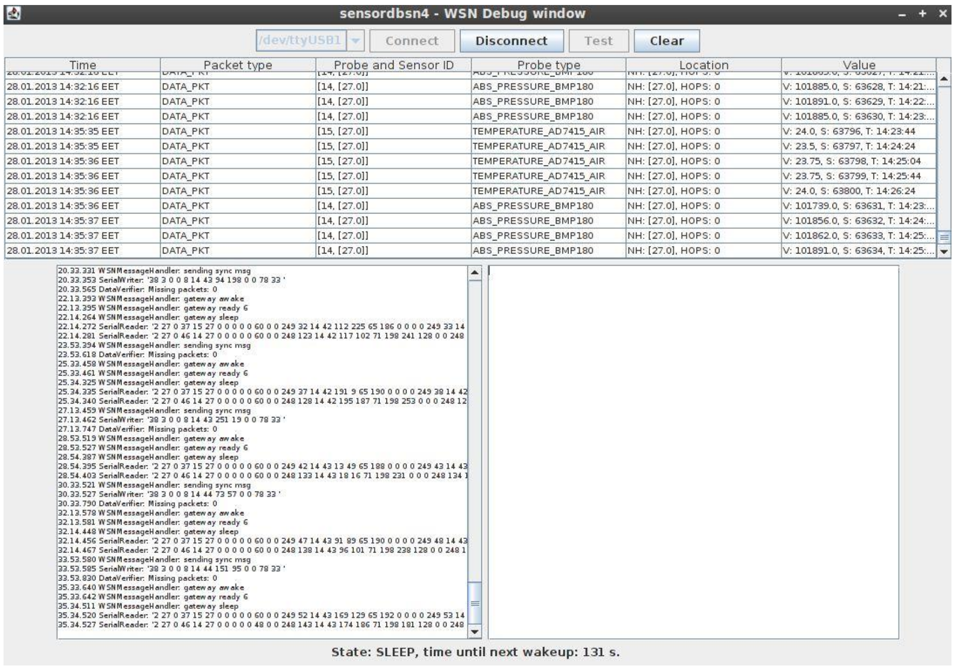
Figure 3. Setup of the SEAM4US environmental monitoring network

There are two types of wireless nodes: *sensor nodes*, which communicate through ZigBee radio (IEEE 802.15.4) with other nodes providing multihop capability; *gateway node*, which communicates with the WSN gateway through RS485 forwarding mote originated measurement data to WSN Gateway and WSN Gateway originated management data to motes. Sensor nodes may contain multiple sensors for environmental measurements and transceivers to communicate with the gateway node as well as with other sensor nodes. The exception to this setting is the *weather station* component, which is a special type of environmental monitoring component based on a single off-the-shelf product. It sends the measurement data through a RS485 link to the gateway server.

4.1.2. Environmental sensor proxies

The sensor network device proxy running in the WSN gateway provides the interface between LinkSmart and the sensors of the environmental monitoring. This component has two main functions: it forwards the sensors' measurements data to the upper layers of the SEAM4US architecture and it interfaces the sensor network management UI (see Sec. 4.1.3) with the sensors. The supporting functions include providing the time synchronization for the sensor nodes and determining the operational status of the sensors based on the received transmissions from the sensor nodes that are stored in a local MySQL database in the Gateway Server. The component is connected to the Gateway Node through a RS485 connection to enable wireless communication to the sensor nodes (see Figure 3). Since the component is written in Java, the Java Runtime Environment is required on the computer in which the device proxy is running, as well as a MySQL Server. The setup of the component is done through two files distributed along with the executable and support libraries.

The sensor network device proxy implements also a GUI, shown in Figure 4, which includes information about the received sensor node transmissions, the state of the sensor network, and some debugging prints within the component.



Time	Packet type	Probe and Sensor ID	Probe type	Location	Value
28.01.2013 14:32:16 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101885.0, S: 63628, T: 14:21:...
28.01.2013 14:32:16 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101891.0, S: 63629, T: 14:22:...
28.01.2013 14:32:16 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101885.0, S: 63630, T: 14:23:...
28.01.2013 14:35:35 EET	DATA_PKT	[15, [27.0]]	TEMPERATURE_AD7415_AIR	NH: [27.0], HOPS: 0	V: 24.0, S: 63796, T: 14:23:44
28.01.2013 14:35:35 EET	DATA_PKT	[15, [27.0]]	TEMPERATURE_AD7415_AIR	NH: [27.0], HOPS: 0	V: 23.5, S: 63797, T: 14:24:24
28.01.2013 14:35:36 EET	DATA_PKT	[15, [27.0]]	TEMPERATURE_AD7415_AIR	NH: [27.0], HOPS: 0	V: 23.75, S: 63798, T: 14:25:04
28.01.2013 14:35:36 EET	DATA_PKT	[15, [27.0]]	TEMPERATURE_AD7415_AIR	NH: [27.0], HOPS: 0	V: 23.75, S: 63799, T: 14:25:44
28.01.2013 14:35:36 EET	DATA_PKT	[15, [27.0]]	TEMPERATURE_AD7415_AIR	NH: [27.0], HOPS: 0	V: 24.0, S: 63800, T: 14:26:24
28.01.2013 14:35:36 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101739.0, S: 63631, T: 14:23:...
28.01.2013 14:35:37 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101856.0, S: 63632, T: 14:24:...
28.01.2013 14:35:37 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101862.0, S: 63633, T: 14:25:...
28.01.2013 14:35:37 EET	DATA_PKT	[14, [27.0]]	ABS_PRESSURE_BMP180	NH: [27.0], HOPS: 0	V: 101891.0, S: 63634, T: 14:25:...

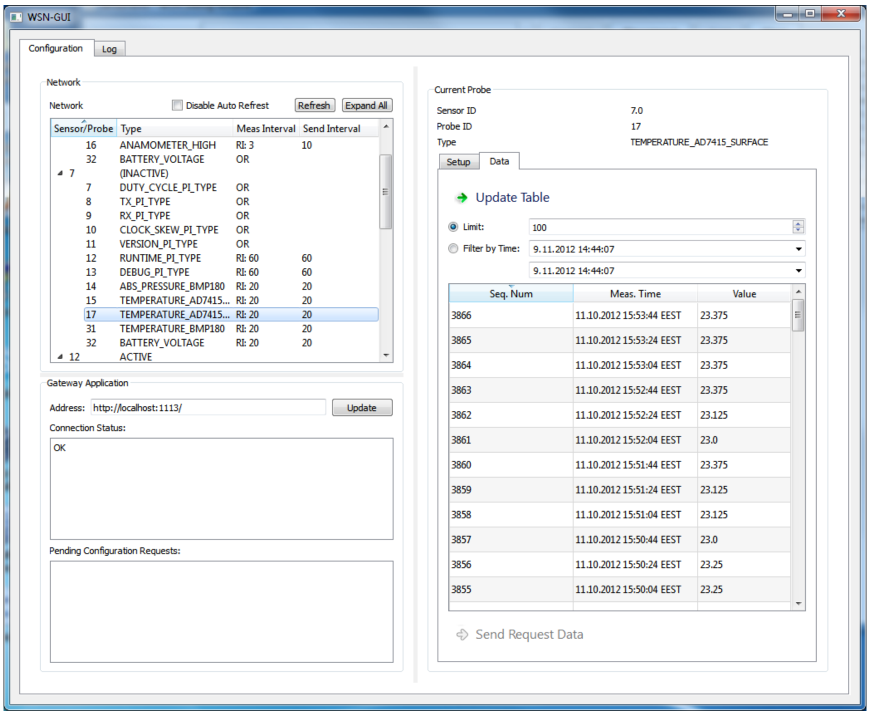
```
20.33.331 WSNMessageHandler: sending sync msg
20.33.353 SerialWriter: '38 3 0 0 8 14 43 94 198 0 0 78 33 '
20.33.565 DataVerifier: Missing packets: 0
22.13.393 WSNMessageHandler: gateway awake
22.13.395 WSNMessageHandler: gateway ready 6
22.14.264 WSNMessageHandler: gateway sleep
22.14.272 SerialReader: '2 27 0 37 15 27 0 0 0 0 0 60 0 0 249 32 14 42 112 225 65 186 0 0 0 0 249 33 14
22.14.281 SerialReader: '2 27 0 46 14 27 0 0 0 0 0 60 0 0 248 123 14 42 117 102 71 198 241 128 0 0 248
23.53.394 WSNMessageHandler: sending sync msg
23.53.618 DataVerifier: Missing packets: 0
25.33.458 WSNMessageHandler: gateway awake
25.33.461 WSNMessageHandler: gateway ready 6
25.34.325 WSNMessageHandler: gateway sleep
25.34.335 SerialReader: '2 27 0 37 15 27 0 0 0 0 0 60 0 0 249 37 14 42 191 9 65 190 0 0 0 0 249 38 14 42
25.34.340 SerialReader: '2 27 0 46 14 27 0 0 0 0 0 60 0 0 248 128 14 42 195 187 71 198 253 0 0 0 248 12
27.13.459 WSNMessageHandler: sending sync msg
27.13.462 SerialWriter: '38 3 0 0 8 14 43 251 19 0 0 78 33 '
27.13.747 DataVerifier: Missing packets: 0
28.53.519 WSNMessageHandler: gateway awake
28.53.527 WSNMessageHandler: gateway ready 6
28.54.387 WSNMessageHandler: gateway sleep
28.54.395 SerialReader: '2 27 0 37 15 27 0 0 0 0 0 60 0 0 249 42 14 43 13 49 65 188 0 0 0 0 249 43 14 43
28.54.403 SerialReader: '2 27 0 46 14 27 0 0 0 0 0 60 0 0 248 133 14 43 18 16 71 198 231 0 0 0 248 134 1
30.33.521 WSNMessageHandler: sending sync msg
30.33.527 SerialWriter: '38 3 0 0 8 14 44 73 57 0 0 78 33 '
30.33.790 DataVerifier: Missing packets: 0
32.13.578 WSNMessageHandler: gateway awake
32.13.581 WSNMessageHandler: gateway ready 6
32.14.448 WSNMessageHandler: gateway sleep
32.14.456 SerialReader: '2 27 0 37 15 27 0 0 0 0 0 60 0 0 249 47 14 43 91 89 65 190 0 0 0 0 249 48 14 43
32.14.467 SerialReader: '2 27 0 46 14 27 0 0 0 0 0 60 0 0 248 138 14 43 96 101 71 198 238 128 0 0 248 1
33.53.580 WSNMessageHandler: sending sync msg
33.53.585 SerialWriter: '38 3 0 0 8 14 44 151 95 0 0 78 33 '
33.53.830 DataVerifier: Missing packets: 0
35.33.640 WSNMessageHandler: gateway awake
35.33.642 WSNMessageHandler: gateway ready 6
35.34.511 WSNMessageHandler: gateway sleep
35.34.520 SerialReader: '2 27 0 37 15 27 0 0 0 0 0 60 0 0 249 52 14 43 169 129 65 192 0 0 0 0 249 53 14
35.34.527 SerialReader: '2 27 0 46 14 27 0 0 0 0 0 48 0 0 248 143 14 43 174 186 71 198 181 128 0 0 248
```

State: SLEEP, time until next wakeup: 131 s.

Figure 4. Sensor network device proxy GUI

4.1.3. Sensor network management UI

A dedicated UI is developed for the sensor network management purposes, depicted in Figure 5. It uses XML-RPC interface provided by the sensor network device proxy to enable control and monitoring of the each sensor and sensor node deployed. It can be used to explore the local sensor database and check the sensor node status. The remote configuration options available through the UI for each sensor independently are: 1) measurement and transmission intervals; 2) measurement types: periodic, threshold event, on-request; 3) reset and over-the-air re-programming. The management UI is written in Qt so the running platform is capable of running Qt applications.



Network

Sensor/Probe	Type	Meas Interval	Send Interval
16	ANAMOMETER_HIGH	RI: 3	10
32	BATTERY_VOLTAGE	OR	
7	(INACTIVE)		
7	DUTY_CYCLE_PI_TYPE	OR	
8	TX_PI_TYPE	OR	
9	RX_PI_TYPE	OR	
10	CLOCK_SKEW_PI_TYPE	OR	
11	VERSION_PI_TYPE	OR	
12	RUNTIME_PI_TYPE	RI: 60	60
13	DEBUG_PI_TYPE	RI: 60	60
14	ABS_PRESSURE_BMP180	RI: 20	20
15	TEMPERATURE_AD7415...	RI: 20	20
17	TEMPERATURE_AD7415...	RI: 20	20
31	TEMPERATURE_BMP180	RI: 20	20
32	BATTERY_VOLTAGE	RI: 20	20
12	ACTIVE		

Address: http://localhost:1113/

Connection Status:

OK

Pending Configuration Requests:

Sensor ID	7.0
Probe ID	17
Type	TEMPERATURE_AD7415_SURFACE

Limit: 100

Filter by Time: 9.11.2012 14:44:07

9.11.2012 14:44:07

Seq. Num	Meas. Time	Value
3866	11.10.2012 15:53:44 EEST	23.375
3865	11.10.2012 15:53:24 EEST	23.375
3864	11.10.2012 15:53:04 EEST	23.375
3863	11.10.2012 15:52:44 EEST	23.375
3862	11.10.2012 15:52:24 EEST	23.125
3861	11.10.2012 15:52:04 EEST	23.0
3860	11.10.2012 15:51:44 EEST	23.375
3859	11.10.2012 15:51:24 EEST	23.125
3858	11.10.2012 15:51:04 EEST	23.125
3857	11.10.2012 15:50:44 EEST	23.0
3856	11.10.2012 15:50:24 EEST	23.25
3855	11.10.2012 15:50:04 EEST	23.25

Figure 5. Environmental monitoring network management UI

4.1.4. Software and protocols

The software in the gateway node and sensor nodes are running on Contiki OS, which is an open-source operating system written in C under development since 2002. It provides functions such as memory handling, event processing, energy usage profiling and protocols for medium access control, multihop communications, and duty cycling for multiple embedded hardware platforms in addition to generic network applications.

The AMPASE platform provides a low level sensoring interface with the sensor nodes. The AMPASE sensoring interface is message based and the messages are described in Table 1 among with Contiki OS based messages. The AMPASE management software provides also XMLRPC interface for optional sensoring user but since the sensor network may be configured statically though MySQL database this interface is not described here.

Table 1. Messages of the AMPASE platform

Message	Origin	Destination	Source	Description
A-PNP	Node	Gateway Server	AMPASE	Register Node and the sensors it contains to the network
A-REQC	Gateway Server	Mote	AMPASE	Set the configuration parameters for each Node
A-REPC	Node	Gateway Server	AMPASE	Acknowledge the configuration parameters received from A-REQC message
A-DATA	Node	Gateway Server	AMPASE	Send measurement data

A-REQDATA	Gateway Server	Mote	AMPASE	Request data from sensors
A-PROG	Gateway Server	Mote	AMPASE	Re-program motes over-the-air
A-ACK	Node	Gateway Server	AMPASE	Acknowledge reception of A-PROG message
A-SYNC	Gateway Server	Mote	AMPASE	Synchronize the clocks in network
C-RREQ	Node	Mote	CONTIKI	Find a multihop route
C-RREP	Node	Mote	CONTIKI	Establish a multihop route

4.1.5. Time synchronization

The WSN gateway, the gateway node, and the sensor nodes are time synchronized based on the WSN gateway clock, which is maintained by a standard Network Time Protocol (NTP). The time synchronization is required in the sensor nodes to timestamp each measurement that may be transmitted later towards the gateway server. Furthermore, the time synchronization allows us to use more energy efficient duty-cycling protocols. The time synchronization is achieved through periodic broadcast transmissions initiated by the gateway node (the gateway node receives the timestamp periodically from gateway server through RS485 connection). The broadcast packet includes a timestamp that indicates the current time. Each node forwards the packet, which contains an updated timestamp. The update is required because of transmission and packet handling delays in the nodes.

The accuracy of the time synchronization protocol is usually about 20ms, which was found convenient because the time resolution of the nodes is 10 ms. However, the accuracy depends on difference of ambient conditions between nodes and the time update interval. For example, 20 ms accuracy can be easily achieved if the update interval is 5 minutes and the ambient conditions do not change drastically in the network because of time-counting 32 KHz crystals included in the nodes.

4.1.6. Duty cycling

Duty cycling is the one of the main techniques to achieve extended battery lifetime in the wireless sensor nodes. The challenge is to enable the node to stay in low-energy consuming mode (i.e., sleep mode) as long as possible. In the sleep mode, the energy consumption is minimized by turning off MCU, radio, and other components, basically enabling only some volatile memory components and the oscillators. However, the sleep mode must often be exited to normal operating mode (i.e., active mode) for actions such as enabling data processing, measurements, data transmissions, and routing of other node’s transmission in case of multihop routing.

AMPASE duty cycling protocols are operating both in medium access control and application layers. The Contiki medium access control implementation (ContikiMAC) enables a continuous duty cycled operation, which puts the node in sleep mode most of the time while ensuring that it is reachable by other nodes in the network. It is based on periodic sampling of the radio interface. In the default configuration, it turns on the node to active mode every 125 ms for few milliseconds to find out if there are transmissions to be received. Therefore, each transmission must be repeated a number of times to be sure that the receiving node is in active mode during the transmissions. In unicast transmissions it is possible to optimize the number of required transmissions for instance by storing the duty cycle phase of the neighbour nodes or timing the transmission based on this information. However, broadcast transmissions must be sent extensive number of times in order to ensure that all reachable nodes will be in active mode during the transmissions.

The ContikiMAC duty cycling is enhanced in the application layer by turning the ContikiMAC off during network idle periods and keeping the node in sleep mode if no data processing is needed. The idle periods occur when no node in the network is transmitting. They are determined in the application layer from the nodes' transmission interval configuration option, which is the interval the node sends measurement results to the gateway node. The absolute time instant when the interval begins is based on the time synchronization information; therefore, nodes with the same interval have also synchronized interval beginning times. From the transmission intervals, the sensor network device proxy is able to determine the highest common nominator (i.e., network transmission interval of all configured transmission intervals in the nodes) that defines the times when the whole network must be in active mode to enable multihop routing. The network transmission interval is provided to the nodes in the time synchronization packets. During each network transmission interval, the nodes operate in active mode for 15 seconds while remaining in sleep mode rest of the time if no data processing is needed. The following timeline illustrates the functions during the 15 seconds period while the ContikiMAC is on.

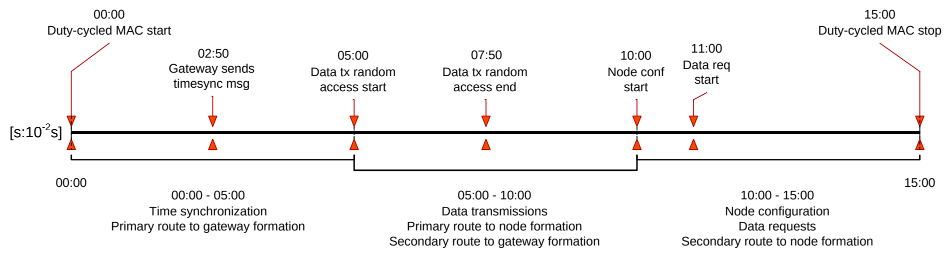


Figure 6. AMPASE main network protocols timeline

When utilizing the described duty cycle protocols, the battery replacement interval depends mainly on the network transmission interval. In practice, each node is configured with a multiple of smallest transmission interval needed in the network to optimize the network transmission interval. The following figure depicts the battery replacement interval based on following parameters neglecting energy consumed during data processing in sleep mode:

- Battery capacity: 2,900 mAh (3 V, assuming linear discharge profile)
- Sleep mode current consumption: 0.02 mA

- Active mode current consumption: 30 mA
- ContikiMAC duty cycle: 5%

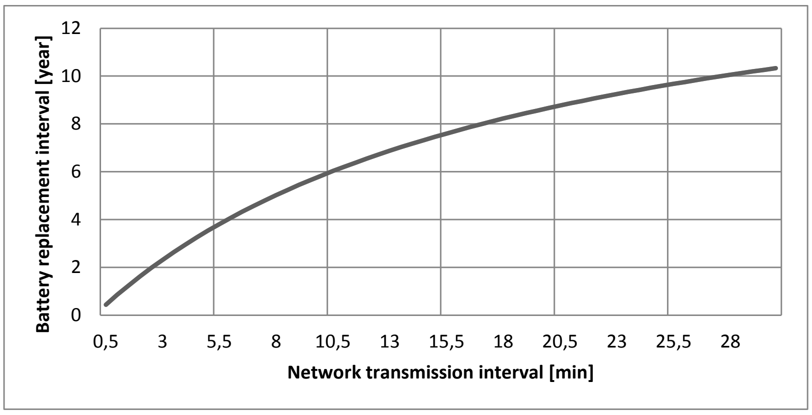


Figure 7. Battery replacement interval of nodes

4.1.7. Routing

The routing is based mainly on time synchronization broadcast packets and data transmissions. The primary method of determining the multihop route towards the gateway node is to monitor the sent time synchronization packets. They include a link quality information – based on received LQI and hop count of the packet, which is updated each time it is rebroadcasted. Therefore, the nodes are able to determine the best up-to-date route towards the gateway node in the beginning of each network transmission interval. The primary method of determining the route toward each sensor node is based on the data transmissions. When the gateway node receives a data transmission or a sensor node is requested to forward a packet towards the gateway node, the last hop information is stored in the node's routing table leading to up-to-date end-to-end multihop path. This path is also best in terms of link quality, if the radio communication between each Node is reciprocal.

The secondary method of determining the multihop routes happens if the Node wants to transmit a packet but a suitable route is not found from the routing table. In this case, Contiki mesh multihop routing is utilized which is based on AODV-type of on-demand routing protocol involving broadcast route-request packets and unicast route-reply packets.

4.2. CCTV proxy and CCTV-based crowd density estimator

The CCTV-based crowd density estimator is the main source of data for modeling the passenger's behaviors and it is based exclusively on the video streams of the CCTV surveillance system already existing at the pilot station, as described in D5.2.2. Thanks to an accurate design of the video processing algorithms, it has been possible to achieve a good accuracy in estimating the crowd (less that 20% of error, which means that in general the density monitored is within ±2 people range from the actual one). This component has been developed using the C++ OpenCV libraries but, as all other SEAM4US architectural monitoring components, it publishes the results of the video processing on the SEAM4US database via the LinkSmart middleware thanks to standard Java-based OSGi modules. The CCTV proxy then is simply an interface towards the crowd density estimator, which are then described in more detail later in the rest of this section.

4.2.1. Working principles

The video streams of the PdG-L3 CCTV suirvellance cameras are combined by a video recorder into one carousel video composed of sections of the individual videos appearing in a predefined order (both the duration of the video sections and the order of appearance of the cameras can be modified). The video recorder is controlled by using the Real Time Streaming Protocol (RTSP) and it can be accessed via the command

```
const std::string videoStreamAddress = "rtsp://127.0.0.1:8554/";
```

The video flow transmission occurs via a widely employed standard (i.e., H.264 over RTP). This carousel video stream is then processed using advanced computer vision algorithms to estimate the amount of people focused by each camera. Such algorithms are described in detail in the following, together with a brief summary of the calibration phase, which is necessary to have a reliable result.

A number of specific actions to minimize the legal/ethical implications of using CCTV data have been taken. In particular, the crowd density estimation algorithms work in real time (i.e., the processing time is less that 1/10 of a second) and hence there is no need to store the CCTV data - since they are processed on the fly - or to show them on screen - since the crowd estimation system does not need any human intervention to work. Also, the CCTV proxy and the whole software running the computer vision algorithms are located on a dedicated computer, which is only accessible via the TMB intranet by using ad hoc credentials; additional software to prevent unwanted intrusions can be easily installed on such computer.

4.2.2. Calibration

The main goal of the calibration is to setup (or retrieve, in case the information is already available) all data concerning the regions of interest (ROI) and the perspective correction of each camera. Also, during the initialization phase the OCR (whose purpose will be specified later in this section) and the background detection are trained. The flowchart of the calibration is shown in Figure 8.

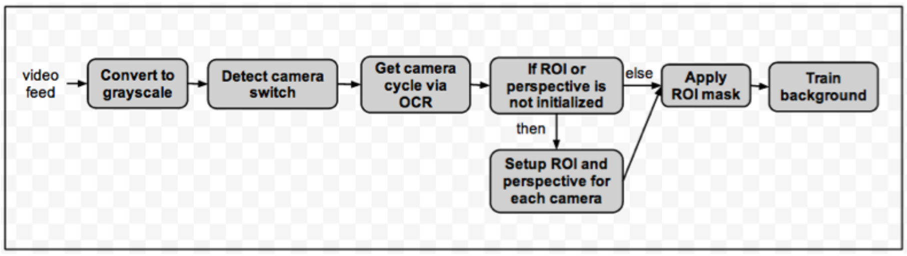
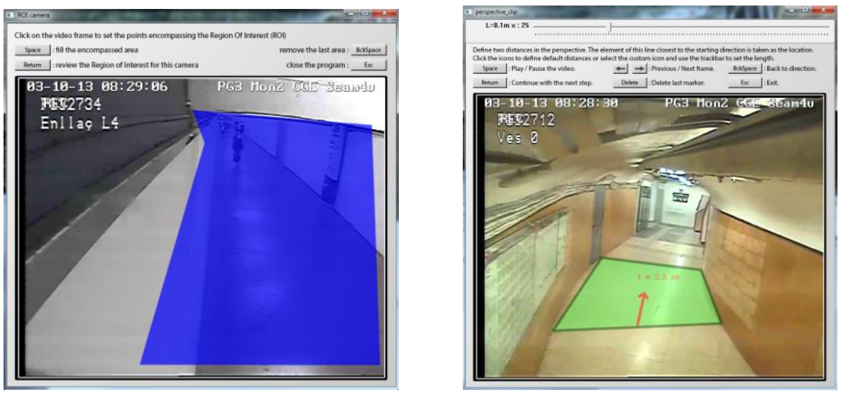


Figure 8. Flowchart of the calibration phase of the crowd density estimation algorithms

By definition, the 'regions of interest' include all the parts of the frame relevant to the passenger detection thus excluding all the areas (walls, tracks etc.) in which human beings are not supposed to be. The setup of the ROI has a two-fold purpose: first, it reduces the amount of data to process, and hence it speeds the algorithm up; second, it prevents the algorithm to be fed with noise coming from uninteresting areas of the frame. In case there are no data available for the ROI of a given camera, the SEAM4US project engineer (see D4.1 for a definition of the SEAM4US roles) can set it via the user interface shown in Figure 9: it is sufficient to select manually the areas to exclude from the image processing algorithms. A similar mechanism has been created for the perspective correction, which of course needs to be different for each camera. The perspective correction is one of the most complex operations in the whole algorithm; considering that cameras may focus a relatively large area, any video processing not considering the perspective information is prone to large errors. In case there are no data available about it, the SEAM4US project engineer can manually set it up via the user interface in Figure 9.


ROI camera

Click on the video frame to set the points encompassing the Region Of Interest (ROI)

Space : fill the encompassed area — remove the last area : BckSpace

Return : review the Region of Interest for this camera — close the program : Esc

perspective_clip

L=0.1m x : 25

Define two distances in the perspective. The element of this line closest to the starting direction is taken as the location.
Click the icons to define default distances or select the custom icon and use the trackbar to set the length.

Space : Play / Pause the video. ← → : Previous / Next frame. BckSpace : Back to direction.

Return : Continue with the next step. Delete : Delete last marker. Esc : Exit.

Figure 9. GUI for the initialization of the ROI (left) and the perspective correction (right).

Besides the ROI and the perspective correction, also the OCR is trained during the calibration phase. Its main purpose is to recognize from what camera the video carousel starts, even though it is also used throughout the execution of the crowd density estimation algorithms in order to verify what frame is currently processed. The last operation of this phase is the background detection, which is simply done by observing what pixels remains unchanged for several frames belonging to different temporal section. Similarly as the OCR, this operation is done once during the initialization of the system, but it is also repeated at the beginning of each new video section to retrain the background since changes (including malfunctioning of the cameras) may occur. An example of the background extraction performed via the standard OpenCV C++ function is shown in Figure 10.

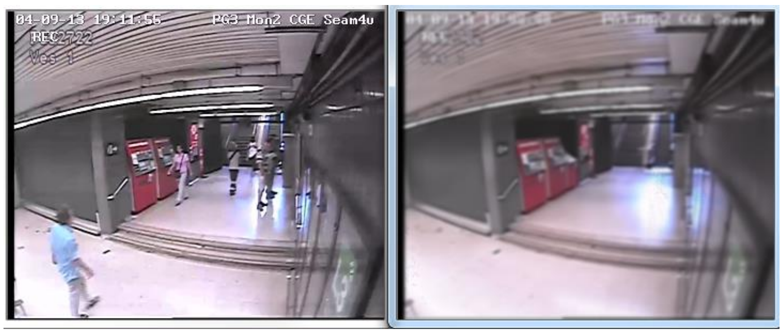


Figure 10. (left) Input frame used as input image in the rest of this section; (right) trained background.

4.2.3. Algorithms for crowd density estimation

In general, motion detection in indoor environments can be performed by using either the optical flow or a combination of edge detection and background subtraction. However, in the first case it is necessary to assume that the image intensity is constant throughout time, which is unfortunately not the case in the case of the SEAM4US project. Comprehensive tests, reported in D4.2.1, have proved that the optical flow approach can fail dramatically in this context resulting in a very noisy image. Therefore, in order to improve the accuracy of the algorithms we have resorted to use a combination of edge detection and background subtraction. The flowchart of the algorithm is shown in Figure 11.

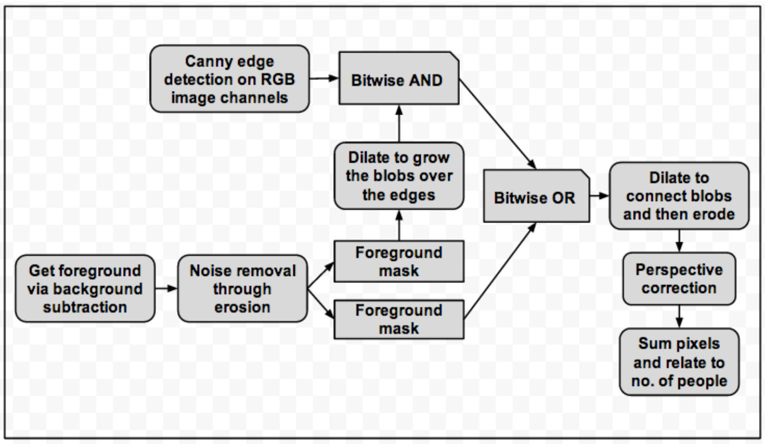

Canny edge detection on RGB image channels
Bitwise AND
Dilate to grow the blobs over the edges
Bitwise OR
Dilate to connect blobs and then erode
Foreground mask
Get foreground via background subtraction
Noise removal through erosion
Foreground mask
Perspective correction
Sum pixels and relate to no. of people

Figure 11. Flowchart of the crowd density estimation algorithm.

The first step of our algorithm consists in retraining the image background with a Gaussian blur which reduces the noise in the image. Several frames (approximately ten) are used during the initialization of the background. The following step consists in extracting the foreground and removing the noise thanks to a simple erosion algorithm (see Figure 12), after which the *foreground mask* is created.

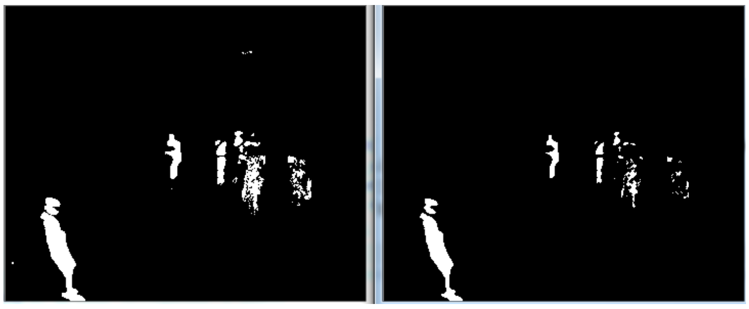


Figure 12. Binary image of the foreground (left) before and (right) after noise removal

In parallel to the process described above, we detect the edge of the whole image by applying the 'Canny edge detection' on the three RGB channels of the original image; the results acquired the different channels are then combined via a simple logic OR. Eventually, the intersection of this image with the dilated *foreground mask* (obtained using a logic AND) allows us to extract the edges of the foreground only (see Figure 13).

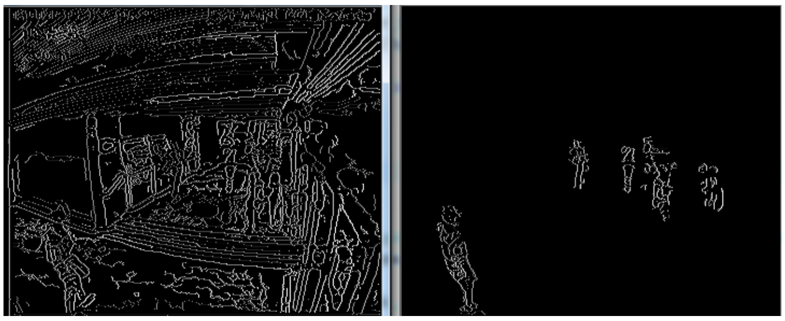


Figure 13. (left) Output of 'Canny edge detection' on the combined RGB channels (right) AND operation of the Canny edge detection and the foreground image after noise removal, see Figure 12.

The last step of the crowd density detection algorithm consists in combining this last image with the foreground mask via a logic OR and to refine the result by dilating (and then eroding) the segmented the blobs (see Figure 14).

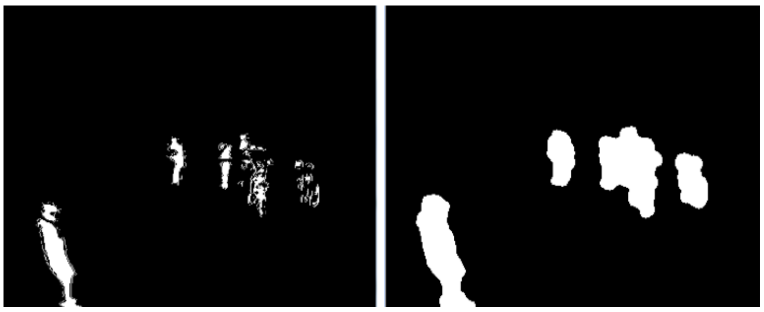


Figure 14. (left) Logic OR between the foreground mask and the edge image; (right) dilated and eroded image in which blobs of people are segmented.

At this point, the actual image processing algorithm is finished, and it remains to apply the perspective correction (whose parameters have been set during the initialization phase) and finally relate the size of the blobs to the actual number of people in them.

It is important to emphasize that robustness is a major issue in the CCTV-based crowd density estimation algorithms, also due to vandalism. Therefore, we have performed a comprehensive analysis of the video frames that correspond to anomalous activities and we have integrated this information into our system. There three emblematic cases (camera/transmission broken, camera occluded, and camera damaged) that our video processing algorithms must be able to handle. While in all cases a warning is issued to the hardware maintenance team, in some cases the algorithms can continue to run, even though performances may deteriorate. Still, in case the camera affected overlaps with others working properly, the overall impact of the malfunctioning component will be mitigated by the compensation mechanisms between cameras of the same areas. In case of a broken transmission or camera, the video recorder sends a ‘blue screen’ to the CCTV proxy. This event is relatively rare; it is handled by discarding the image and issuing a warning to the hardware (HW) maintenance team via the supervisor (see Sec. 7). In case of an occluded camera (see Figure 15, left), part of the screen may be perceived as background and, due to the difference to previously recorded background images for the same camera, the algorithms discard the image and issue a warning to the HW maintenance team. In case of a scratch on the camera (see Figure 15, right), the image can still be processed, but the starch is now considered as part of the background.

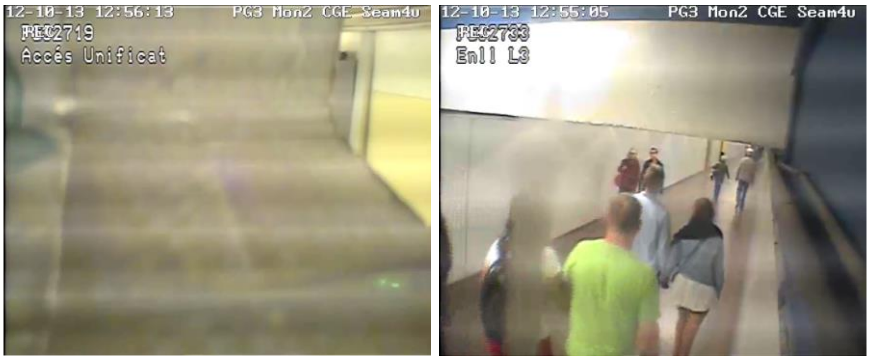


Figure 15. Examples of CCTV mulfunctioning: (left) occluded camera; (right) scratched camera

Thorough tests have showed that the robustness of the algorithm is very close to 100%; a scratche on the camera is the hardest problems to detect since it is in principle undistinguishable from a person standing still within the area focused by the camera (which for instance happens on the platform while waiting for the train). However, we have introduce a buffer time of a few minutes after which an absolute still object becomes part of the background and hence it is ignored by the algorithm.

Before writing the data on the SEAM4US database, it is necessary to perform yet another step to aggregate the crowd density estimations coming from the individual cameras according to zones. For instance, the whole platform in each direction is considered as *one* zone even though it contains multiple cameras. Not all zones are covered by the CCTV system, but this happens only for 13 of them, as evident from Figure 16. The overlap occurs only between cameras 00 and 01 in zone TL3.1, between cameras 04 and 05 in zone TL3.2, and between cameras 32 and 33 in zone SLb.

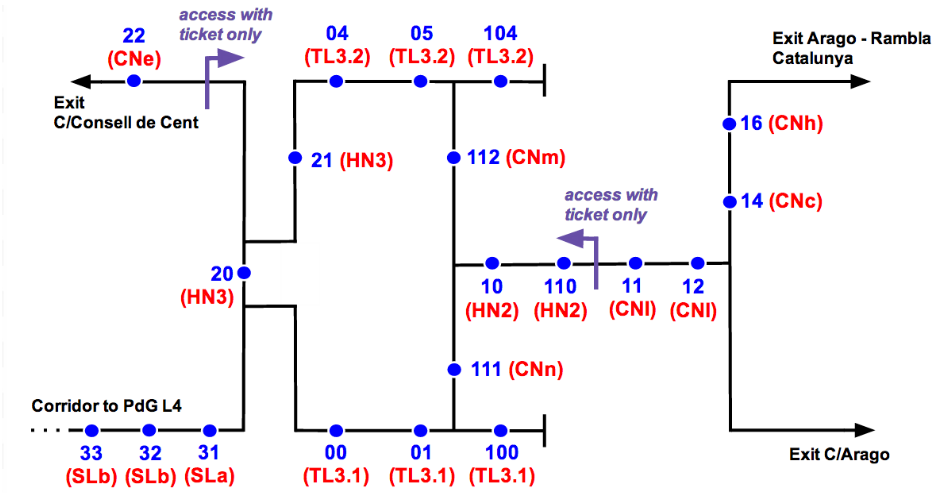


Figure 16. Topology of the existing CCTV system deployed at PdG-L3 station

Finally, the outcome of the CCTV-based crowd density must be written to the SEAM4US database, according to entry described in the appendix. The time and date for the videos are extracted from the local computer operating system and, since the video processing is done in real time, they will correspond to the actual ones of the videos. If needed, the OCR system, whose main goal is to synchronize the carousel video stream, can also be used for this purpose.

4.2.4. Using the crowd density estimator

In general, the crowd density estimator does not need to interact with any user, as it runs in background getting data directly from the video stream and publishes the results of the video processing on the SEAM4US database. However, in this section we give a few notes about the requirements and the setup of the system.

Currently, the crowd density estimator runs on a standard general purpose PC, with a dual-core processor and 4 GB of RAM. From the software point of view, the following software needs to be installed on the local computer:

- MinGW C++ compiler (which provides the libraries used by the software);
- OpenCV 2.4.6 (the main C++ library used in the CV algorithms);
- CMake GUI (GUI used by the C++ program);
- FFMPEG (used by OpenCV to handle videos);
- VLC media player (necessary to Access the video stream).

All software is available for free, no license is required. The necessary programs are included in the folder provided with the crowd density estimator software. In order to be able to run the crowd density estimator, the user needs to make the following steps:

1) Install the VLC, FFMPEG and Cmake using the installers available at:
    a. http://www.videolan.org/vlc/index.html
    b. http://www.ffmpeg.org/
2) Follow steps 1 – 16 on:
    a. http://nenadbulatovic.blogspot.nl/2013/07/configuring-opencv-245-eclipse-cdt-juno.html

The setup is also strightforward, and it includes the following steps:

1) Run VLC media player
    a. Select Stream from file menu
    b. In network, enter the URL rtsp://172.25.44.217/camera-1 and press *Stream*
    c. Set destination to RTSP and uncheck display locally.
    d. Start stream
2) Run seam4us.exe, which contains all code of the crowd density estimator.

4.3. Train arrival proxy

The task of this component is to publish on the LinkSmart middleware the arrival time of the next train based on the information retrieved from an appropriate data source, which can be static (e.g., a .xls file containing the train schedule for the whole day) or dynamic (e.g., a real-time update about the train schedule provided by the TMB operation department). In the current version, the component parses a static .xls file containing the train schedule for each day of the year, and it iteratively publishes a new event (according to the event format defined in Appendix A) every time that a new train arrives.

This component has been implemented in Java as a OSGi-component and it consists of two Java files: `trainSchedule.java` and `TrainArrivalEventPublisherImpl.java`. The first file manages the train schedule and calculates the time of the next train whereas the second publishes the event implements the supervision interface, which enables the supervisor to monitor the status of the train arrival component.

4.4. Weather forecast proxy

Weather forecasts are obtained from the external web-service wunderground.com©, which provides hourly forecasts in JSON format for the weather station closest to the PdG-L3 location (i.e., *Barcelona LBL Airport*). The weather API service (called *Cumulus Plan*) allows up to 10 calls per minute with a maximum of 500 calls per day, which is enough to cover the maximum prediction horizon that we expect to use. The provider currently updates the weather forecast and current conditions every hour.

The weather forecast service is accessed from within the access-restricted TMB network thanks to a proxy, which is configured in *LinkSmart Status* (see Sec. 3). Every time that a new forecast is retrieved, an event is sent via LinkSmart. The parsing of weather data in JSON format is done by JSON Java library working together with Google GSON library for Java. Forecast data are finally accessed by the methods that retrieve the corresponding values of the returned Java object. The Java package that manages the weather forecast service is encapsulated into a *WeatherForecast* Java library that exposes some public classes, properties and methods. In order to stay within the limitations of the service and at the same time to have always updated data, the java bundle does an automatic poll of the web service with a poll rate of 180s.

4.5. Enistic meters proxy

Data from the Enistic meters are relayed to the SEAM4US server through the UDP proxy located in the low voltage room, whose only purpose is to forward broadcast UDP packets as unicast UDP packets towards the SEAM4US Server for further processing. All packages originate from the smart meters and are sent on UDP port 53005.

To receive and process data from the UDP proxy, the program `EnisticUDPReceiver.exe` has to be started. This application is written in C# and processes the data from Enistic smart meters for redistribution and storage.

In short, processing of the Enistic data means parsing UDP packages and converting them into relevant measurement data. The Enistic data format is dependent of the controller version, and the receiver is capable of receiving data on the following Enistic formats:

Enistic new version format:

```
10:54:31.17 Enistic:0050C2F4C21F:
D1:W:13/5/2007/10/54/31;S:000D6F0001A3D676,P=D2,@=23,W=0000,U=34,C=13,M=22
4,V=015:~3B
```

Enistic old version format:

```
Enistic:0004A3BB5D35:EWAPI:000D6F0001A3E7AB:L=0,T=0,M=214,U=290,E=0,W=2290
0,C=12,P=D2
```

All the Enistic sensors of SEAM4US use either the 'new version' or the 'old version' format. Enistic smart meters send voltages, status messages and more, but the important values consist of controller, serial, apparent power and channel. Using the 'old version format', these values represent "`Enistic:`", " `EWAPI:`", "`U=`", "`C=`". Real power is not used because the Enistic smart meters calculate it from the apparent power and the power factor it finds between its energy supply wave and the intensity of the circuit. Since the power supply is not the same for the device and the measured load, the power factor is not correct. Therefore, we use the apparent power and the PF that was measured with the electrical analyser.

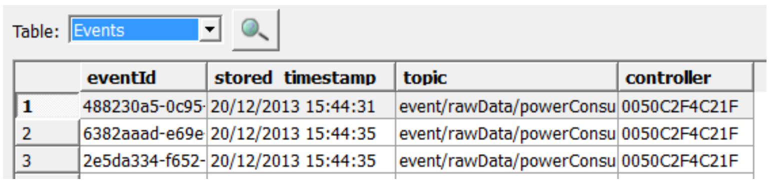
Enistic devices have different clamp sizes that must be taken into account when processing data. This configuration is performed in a file called `EnisticMapping.xml` located in the `EnisticUDPReceiver` folder. In order to correctly read data from the device, each smart meter must have an entry on the following format.

```
smartmeter serial="000D6F0001A3D62E" id="1" panel="P1">
      <powerline code="3NC-3" category="escalator:1" where="access"
powerfactor="0.71">
            <channel id="11" size="15"/>
            <channel id="12" size="15"/>
            <channel id="13" size="15"/>
      </powerline>
</smartmeter>
```

It is evident that each smart meter has its own entry specifying the serial number, the sensor ID within the SEAM4US system, and the cabinet panel where this smart meter is installed. The clamp size for each of the up to 16 channels of the smart meter in use is specified, grouped by the power line they are attached to (according to the number of phases).

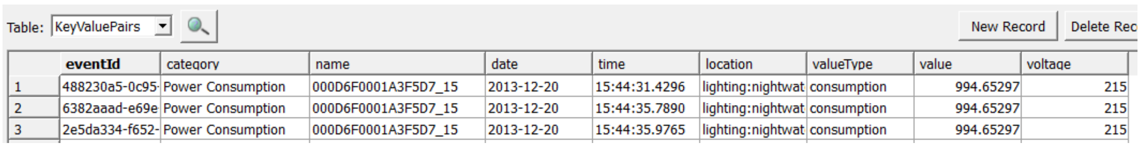
After processing in EnisticUDPReceiver, relevant data are stored locally in the database and then published to *LinkSmart Event Manager* (see Sec. 3.3). Note that the Graphical User Interface (described in detail in Sec. 6) uses the data sent to *LinkSmart Event Manager* and not the local SQLite database.

The local storage in SQLite is being done in separate databases, one for each measuring point/channel of the Enistic energy meters. These databases can be found in `C:\IoTDeviceDBs\Enistic\NewFormat` on the server and follow the name conventions '*SM-4_3.s3db*', where '*SM-4*' stands for *Smart Meter 4* and the '*3*' means *channel 3*. Data are both stored as events (see Figure 17) and key/value pairs (see Figure 18), similar to the SQLite storage of the *LinkSmart Event Manager*.



	eventId	stored timestamp	topic	controller
1	488230a5-0c95-	20/12/2013 15:44:31	event/rawData/powerConsu	0050C2F4C21F
2	6382aaad-e69e	20/12/2013 15:44:35	event/rawData/powerConsu	0050C2F4C21F
3	2e5da334-f652-	20/12/2013 15:44:35	event/rawData/powerConsu	0050C2F4C21F

Figure 17. SQLite event storage



	eventId	category	name	date	time	location	valueType	value	voltage
1	488230a5-0c95-	Power Consumption	000D6F0001A3F5D7_15	2013-12-20	15:44:31.4296	lighting:nightwat	consumption	994.65297	215
2	6382aaad-e69e	Power Consumption	000D6F0001A3F5D7_15	2013-12-20	15:44:35.7890	lighting:nightwat	consumption	994.65297	215
3	2e5da334-f652-	Power Consumption	000D6F0001A3F5D7_15	2013-12-20	15:44:35.9765	lighting:nightwat	consumption	994.65297	215

Figure 18. SQLite Key/Value storage

4.6. SOCOMEC meters proxy

Data from the SOCOMEC meters are received and processed at the SEAM4US Server by `SocomecDevice.exe` (*SOCOMECdevice*, hereafter), which is an application written in C#. The communication of the SOCOMEC meters with the TMB network is done via an Ethernet interface (see Figure 19), which is configured to use Modbus TCP on IP address 172.25.44.220 using default port 502. Multiple meters can be attached to one Ethernet interface, according to standard Modbus systems, each with unique identification numbers; in particular, SEAM4US uses two Socomec Diris A10 meters, one for each escalator.

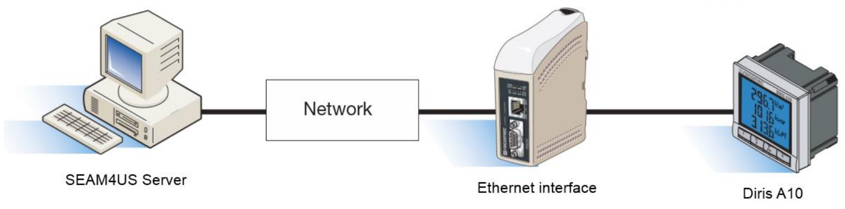


Figure 19. Communication between the SEAM4US Server and a SOCOMEC meter via the TMB network

According to the Modbus TCP protocol, the communication is always initiated by the server and the response is sent by the gateway/device, which forces *SOCOMECdevice* to use polling as primary measurement method. Polling is performed at most each second, but only on active power which can be expanded if needed. Accumulated values are sent to the *LinkSmart Event Manager* (see Sec. 3.3) after 1-2 minutes. *SOCOMECdevice* always waits exactly one second before performing a new Modbus request, which means that another request is not started until the first one finishes plus one second. This is to ensure that the requests are not stacked and that the TMB network is not overloaded. After approximately 1 min, an average of the latest values are calculated and then sent to the *LinkSmart Event Manager* for further processing and storage. *SOCOMECdevice* does not use any SQLite local storage as backup, consumption data is solely sent to the *LinkSmart Event Manager* which ensures that the data is stored correctly. Therefore, *LinkSmart Event Manager* needs to be started in order for *SOCOMECdevice* to successfully save consumption data. *SOCOMECdevice* also handles cases where the network connection is cut off, or single errors in the library, which implies that there is no need to restart the application after a network, malfunction.

The Modbus connection itself is performed through an open-source Modbus TCP library capable of performing all the necessary requests. Modbus TCP is much similar to the regular Modbus, except for a main difference illustrated in Figure 20. A request consists of three fields: i) unitID, ii) function code, iii) address; for instance, `ID:2 FUNCTION:3 ADDRESS:790`.

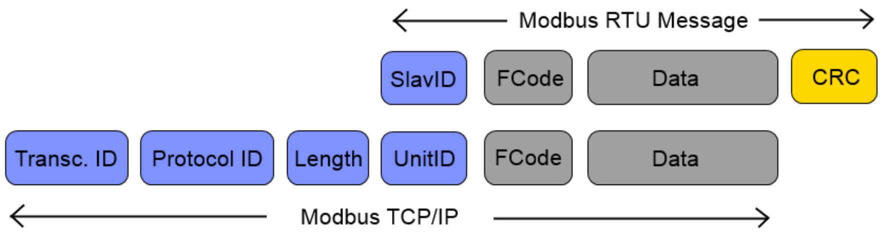


Figure 20. Differences between Modbus TCP and Modbus RTU messages

Figure 21 is an example of the most relevant parameters which can be accessed through Modbus interface and Socomec Diris A10. Address 790 returns the active power across all the phases and is currently the only value needed, and it is thus reported to the *LinkSmart Event Manager*. All of the values presented in Figure 21 uses `FUNCTION 3`.

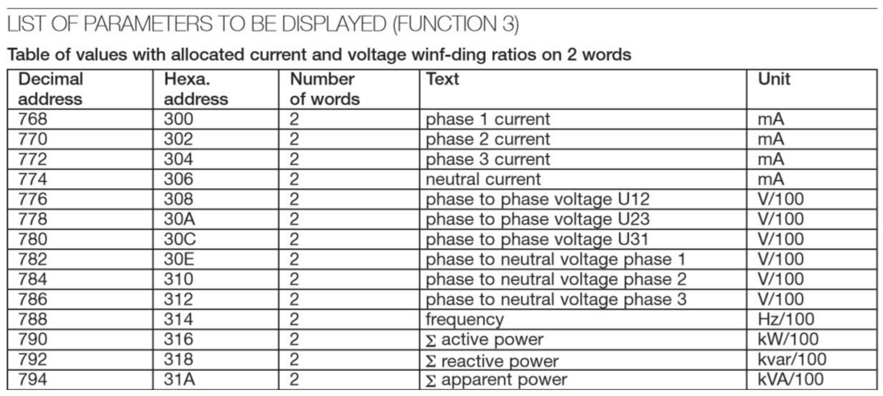

LIST OF PARAMETERS TO BE DISPLAYED (FUNCTION 3)

Table of values with allocated current and voltage winf-ding ratios on 2 words

Decimal address	Hexa. address	Number of words	Text	Unit
768	300	2	phase 1 current	mA
770	302	2	phase 2 current	mA
772	304	2	phase 3 current	mA
774	306	2	neutral current	mA
776	308	2	phase to phase voltage U12	V/100
778	30A	2	phase to phase voltage U23	V/100
780	30C	2	phase to phase voltage U31	V/100
782	30E	2	phase to neutral voltage phase 1	V/100
784	310	2	phase to neutral voltage phase 2	V/100
786	312	2	phase to neutral voltage phase 3	V/100
788	314	2	frequency	Hz/100
790	316	2	Σ active power	kW/100
792	318	2	Σ reactive power	kvar/100
794	31A	2	Σ apparent power	kVA/100

Figure 21. SOCOMEC address map

With help of Figure 20 and Figure 21, it is then easy to conclude that the previously mentioned request (`ID:2 FUNCTION:3 ADDRESS:790`), reads (`FUNCTION:3`) active power (`ADDRESS:790`) from the device 2 (`ID:2`).

4.7. SACI meters proxy

Data from the SACI meters are received and processed at the SEAM4US Server by `SACIDevice.exe`, similarly as for the SOCOMEC meters (see Sec. 4.6). Also the access to the TMB network, which is shown in Figure 22, resembles the one of the SOCOMEC meters, since it is done via an Ethernet interface. The SACI Ethernet interface (etherGATE) is configured to use Modbus TCP with IP address 172.25.44.216 and port 502. Multiple SACI meters can be attached to the etherGATE, but SEAM4US uses only a single SACI MAR 144.

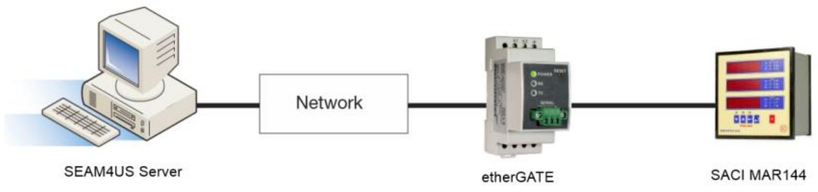


Figure 22. Communication between the SEAM4US Server and a SACI meter via the TMB network

The communication occurs exactly as in SOCOMEC implementation: accumulated values are sent to *LinkSmart Event Manager* each minute, and the underlying Modbus TCP connection is performed using the same library. As explained above, no SQLite backup storage is used and the *LinkSmart Event Manager* must be started in order for values to be processed further.

5. ACTUATOR PROXIES

The systems driving the three actuators (fans, escalators, lights) are described in detail in D5.2.2. Therefore, in the following we will discuss about actuators proxies, which run on the SEAM4US server, and give a brief introduction about the functionalities of each actuator within the SEAM4US pilot.

5.1. Fan proxy

The SEAM4US fan pilot consists of a modification of the fans PLC so that each fan (two located at the platforms and two located in the tunnels, in the pilot project) can be driven continuously in the ranges [-45 Hz,-25 Hz] and [25 Hz, 45 Hz] or stopped (0 Hz). In the original system, each fan could only be driven off (i.e., at 0 Hz) or on (i.e., at 45 Hz). Fans are driven by Beckhoff BK9050 Ethernet couplers with input and output modules connected to the existing fan frequency variators as well as to the TMB SCADA system (also known via the TMB Spanish acronym CCIF) through a switching logic as shown in Figure 23. Besides controlling the fans, this system is also used to forward the values measured by the anemometers in each fan's duct.

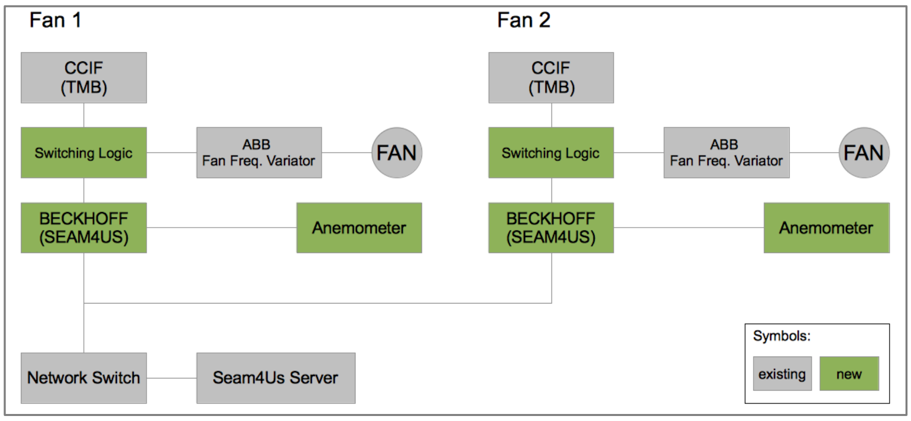


Figure 23. Fan proxy communication between the server and the new devices

The switching logic build on physical relays allows us to switch between the SEAM4US control and the 'standard' control of the TMB CCIF through the new Beckhoff modules. It is important to emphasize that only the TMB CCIF control can toggle between the two modes whereas the SEAM4US control can indicate to the TMB CCIF that it is ready to take over the control and without activating itself. From the practical point of view, the operator has to explicitly allow SEAM4US to take over the control each time that it has been returned or deactivated. If the communication is interrupted for more than 10 seconds, then the hardware automatically returns to a default state triggered by the watchdog in which the control is automatically returned to the TMB CCIF.

The fan control is implemented as an OSGi bundle running in the LinkSmart environment. Within this bundle, each fan to be controlled is addressed separately and their outputs can be driven between 0 V and 10 V, mapping the whole range of operational frequencies of the fans. The bundle polls each Beckhoff device at least two times per second using the MODBUS TCP protocol, which has been described in Sec. 4.6.

5.2. Light proxy

The SEAM4US lighting pilot consists of a deployment, in some parts of the station as described in D5.1.2, of LED lamps that are controllable through the DALI protocol, which assigns an address to each light (also called, *node*) and allows us to control each light individually, e.g., switching it on and off or setting the dimming level. Each node can be assigned to one or multiple groups in order to control the lighting by zones.

A DALI gateway from Orama Inc. has been installed in the station and it is integrated to the SEAM4US system thanks to a LinkSmart device proxy, which provides the following functionalities: i) request a list of nodes; ii) request a list of zones; iii) request the assignment of nodes to zones; iv) request the current switching status and dimming level for each node; v) set switching status and dimming level for individual nodes, or entire zones. The assignment of nodes to groups is done manually or programmatically through the web interface of the DALI Gateway. The DALI proxy is implemented in Java based on the LinkSmart/OSGi-framework. Within the framework, it provides the service called 'DALIProxy'.

5.3. Escalator proxy

The SEAM4US escalator pilot consists of a modification of the escalator, which allows the SEAM4US controller to select dynamically the escalator speed. Before SEAM4US, the escalator was always running at 0.5 m/s (unless, of course, it was switched off). Thanks to SEAM4US, the escalator speed can be set to 0.5 m/s where there is a intense passenger load or 0.4 m/s when there is a moderate passenger load). Additionally, radars systems have been installed to detect when no passenger is using the escalator, which is then stopped altogether. This allows the escalator to stop altogether when no passengers are on it whereas when the radars are not active the escalator is running at 0.2m/s (see D3.2.2 for more details). As explained in D5.1.2, a dedicated Beckhoff PLC provides the signal about the speed to the escalator actuator, which actually changes the speed only when there are no people on the escalator, due to safety issues.

The escalator LinkSmart proxy is implemented in Java based on the LinkSmart/OSGi-framework and it communicates with the Beckhoff PLC using the Modbus/TCP protocol, which has been already discussed in Sec. 4.6. Within the framework, it provides the service called 'EscalatorProxy'. The main functions of the proxy are to set the escalator speed selection signal, to retrieve the current escalator speed from the PLC, and to publish as a LinkSmart event.

6. GRAPHICAL USER INTERFACE

All SEAM4US users (i.e., the project engineer and the station operator, as defined in D4.1) can interact with the system via a Graphical User Interface (GUI) which has been developed in .Net, C#, HTML5, CSS3 and several JavaScript libraries within the Microsoft Visual Studio IDE. Due to privacy and data sensitivity issues, the GUI runs on the SEAM4US server, which is accessible only from the TMB computer network, at the address http://localhost/Signin.aspx. In order to protect sensitive data and prevent accidental changes of the system, access to the GUI requires user identification (see Figure 24).

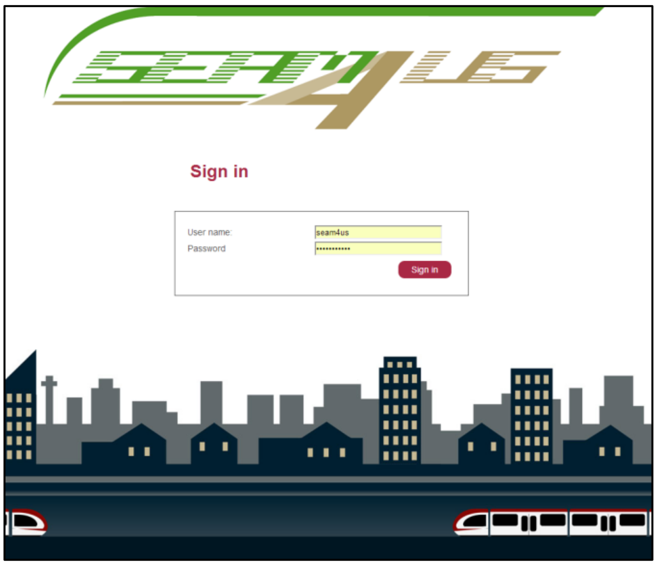
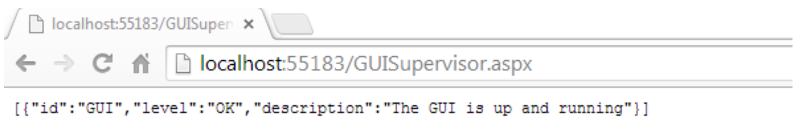


Figure 24. User identification screen to access the GUI: each SEAM4US role has different credentials

Each SEAM4US user has different credentials, which allow the user to access only the tabs s/he is allowed to see.

The SEAM4US architecture shown in Figure 1 gets its data from the *data storage* component and from the *control* component. The communication with the former is one directional since the GUI displays data but the user cannot manipulate them; in contrast, the communication with the former (which occurs via web services) is bidirectional since the user can interact with such component via the GUI. Also, the GUI communicates the *representation* component and with the supervisor; the status of the GUI can be checked by accessing the address localhost:55183/GUISupervisor.aspx, as shown in Figure 25.



```
[{"id":"GUI","level":"OK","description":"The GUI is up and running"}]
```

Figure 25. Example of check on the GUI functionality

The GUI has three tabs: the 'Station map' tab (see Sec. 6.1) and the 'Monitoring data' tab (see Sec. 6.2) refer to data retrieved from the data storage; the 'Control' tab (see Sec. 6.3) refers to status of the controller.

6.1. Station map tab

The 'Station map tab' retrieves the sensors location and the recorded measurements from the data storage and shows them on a map of the station (see Figure 26). This tab is shown only in the project engineer GUI, as it contains detailed information about the status of the system.

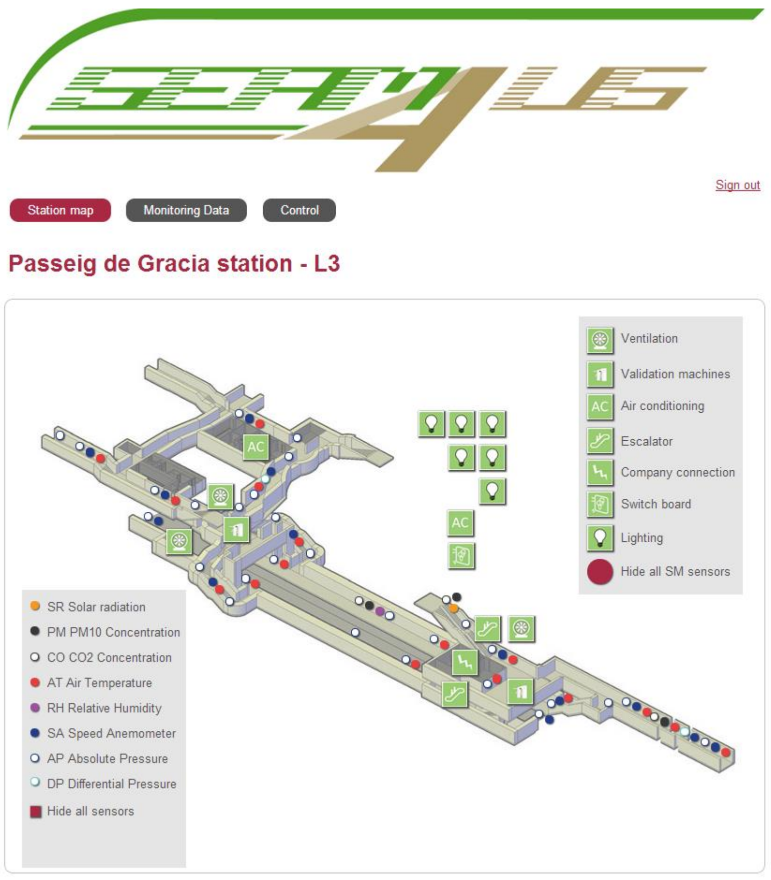


Figure 26. 'Station map' tab showing the position and values of all sensors

The environmental sensors (e.g., temperature, pressure, CO2) are visualized as circles and they are color coded according to the legend on the left-bottom corner of Figure 26; the consumption sensors (e.g., escalator, lightning) are visualized as squares and they are icon coded according to the legend on the right-top corner of Figure 26. The legend is interactive: clicking on a legend item toggles the visualization of all monitoring sensors of that kind from the station map. For instance, in Figure 27 only four kinds of sensors are visible: temperature and speed anemometer, as environmental monitoring, and air conditioning and escalator, as consumption monitoring.

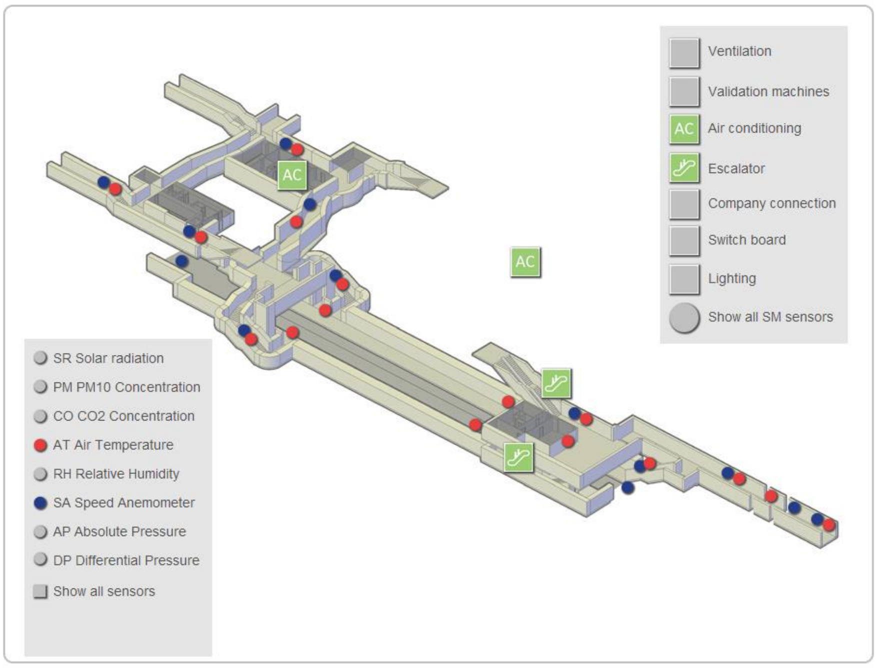


Figure 27:. 'Station map' tab showing the position and values of selected sensors

In the station map, the user can visualize on the tooltip the latest value and timestamp of each sensor by hovering on it with the mouse. Also, the user can access the historic values of each sensor by selecting it, which will prompt a pop up window showing the values of the last month in a chart (see Figure 28). Different months and years can be selected by using the dropdown list.

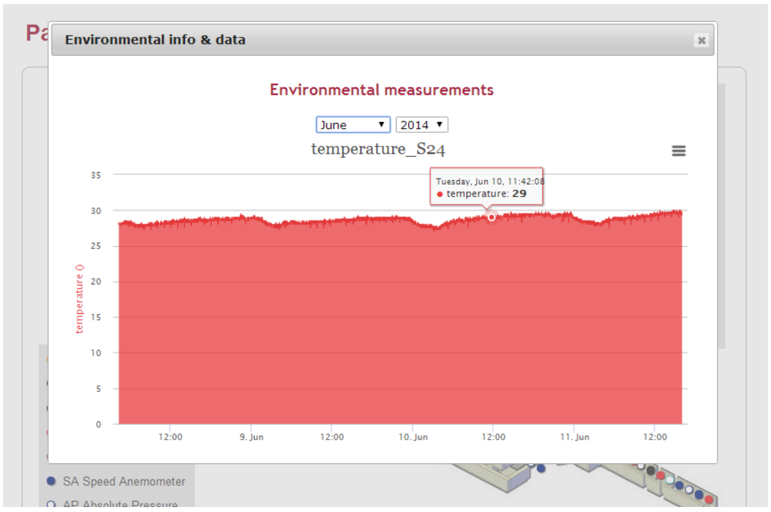


Figure 28. Historical values of a selected sensor

The chart can be zoomed in by selecting a time span and it can be reset to the original format by clicking on the 'reset zoom' button, which appears automatically. Also, it is possible to download or print the chart by clicking on the icon in the upper-right corner (see Figure 29).

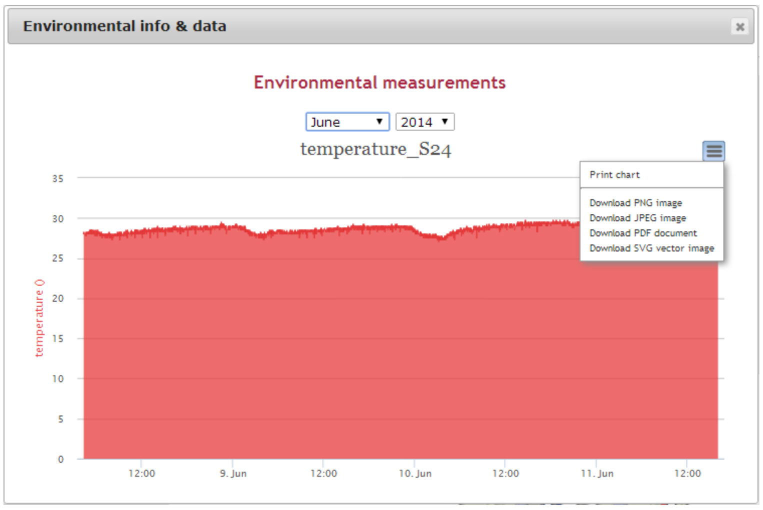


Figure 29. The chart of the historical values can also be printed or downloaded in multiple formats

The chart for the consumption monitoring has an additional function, since it is possible select a specific circuit from a dropdown list in the upper-left corner (see Figure 30).

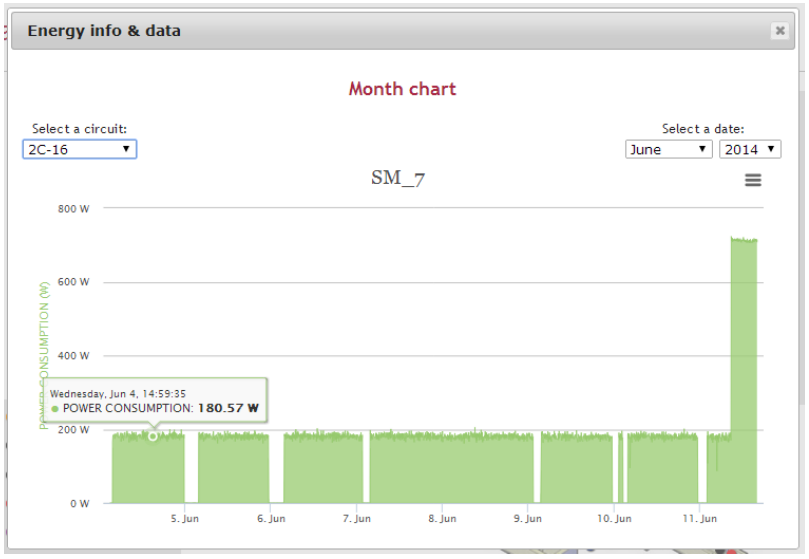
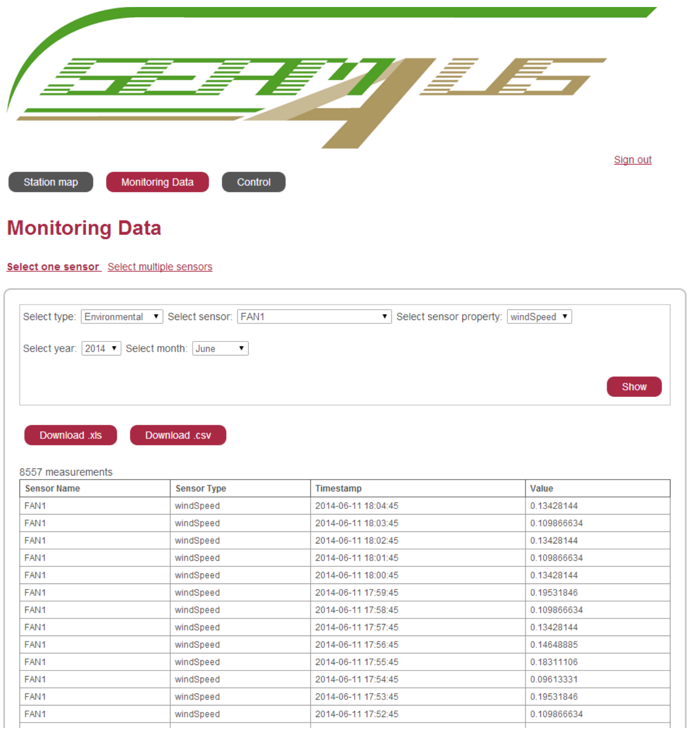


Figure 30. Example of historical values for a consumption sensor

6.2. Monitoring data tab

Monitoring data can also be visualized in a tabular way within the 'Data' tab, which contains two options: 'select one sensor" and 'select multiple sensors'. In the first case, the user can visualize a table containing the values of a single property of one sensor for a specific month, and download it either as an XLS or as a CSV file (see Figure 31).



Monitoring Data

Select one sensor Select multiple sensors

Select type: Environmental Select sensor: FAN1 Select sensor property: windSpeed

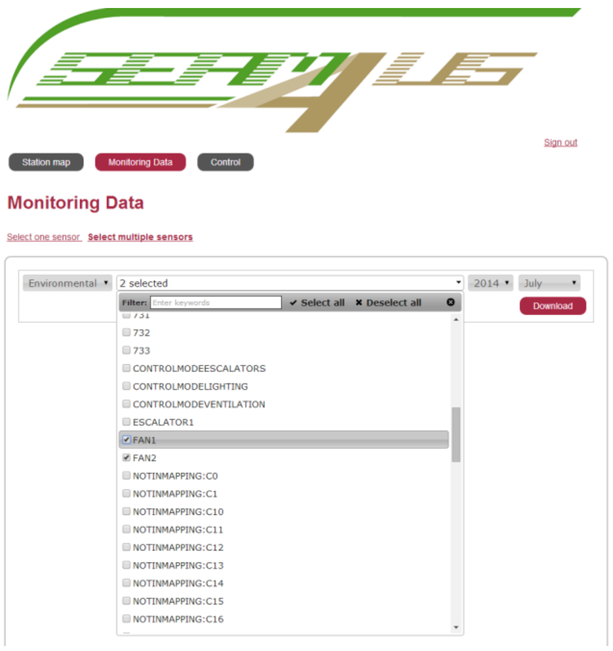
Select year: 2014 Select month: June

8557 measurements

Sensor Name	Sensor Type	Timestamp	Value
FAN1	windSpeed	2014-06-11 18:04:45	0.13428144
FAN1	windSpeed	2014-06-11 18:03:45	0.109866634
FAN1	windSpeed	2014-06-11 18:02:45	0.13428144
FAN1	windSpeed	2014-06-11 18:01:45	0.109866634
FAN1	windSpeed	2014-06-11 18:00:45	0.13428144
FAN1	windSpeed	2014-06-11 17:59:45	0.19531846
FAN1	windSpeed	2014-06-11 17:58:45	0.109866634
FAN1	windSpeed	2014-06-11 17:57:45	0.13428144
FAN1	windSpeed	2014-06-11 17:56:45	0.14648885
FAN1	windSpeed	2014-06-11 17:55:45	0.18311106
FAN1	windSpeed	2014-06-11 17:54:45	0.09613331
FAN1	windSpeed	2014-06-11 17:53:45	0.19531846
FAN1	windSpeed	2014-06-11 17:52:45	0.109866634

Figure 31. Values of one sensor in the 'monitoring data' tab

In the second case, all properties of multiple sensors for a specific month can be selected at once (see Figure 32) though, for obvious reasons, this large amount of data cannot be previewed but only downloaded as a CSV file. The list where the user can select sensors from has built-in functions such as a filter among sensor names and a toggle to select/deselect all sensors.



Monitoring Data

Select one sensor Select multiple sensors

Environmental 2 selected 2014 July

- [ ] 731
- [ ] 732
- [ ] 733
- [ ] CONTROLMODEESCALATORS
- [ ] CONTROLMODELIGHTING
- [ ] CONTROLMODEVENTILATION
- [ ] ESCALATOR1
- [x] FAN1
- [x] FAN2
- [ ] NOTINMAPPING:C0
- [ ] NOTINMAPPING:C1
- [ ] NOTINMAPPING:C10
- [ ] NOTINMAPPING:C11
- [ ] NOTINMAPPING:C12
- [ ] NOTINMAPPING:C13
- [ ] NOTINMAPPING:C14
- [ ] NOTINMAPPING:C15
- [ ] NOTINMAPPING:C16

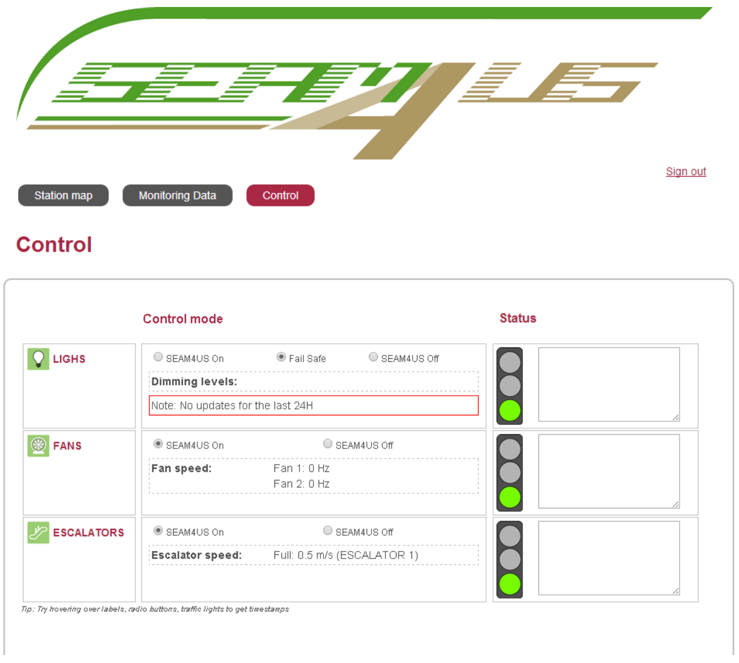
Figure 32. Values of multiple sensors in the 'monitoring data' tab

6.3. Control tab

The 'Control' tab is shown both in the both engineer and station operator GUIs, and it opens a page that is divided into two vertical parts (see Figure 33): the 'control mode' and the 'subsystem status'. They are both divided into three horizontal parts, one for each actuator.

The 'control mode' shows the general status of the SEAM4US control, which can be ON or OFF (there are *fail safe* modes for the lights and for the fans, which is further documented in D3.2.2). Additionally, the GUI displays the dimming levels for the lighting subsystem, the fan speed for ventilation subsystem, and the escalator speed for the escalator subsystem. If any of the subsystem has a 'warning' or a 'failure', then a label appears as well. Also, if a subsystem fails, the corresponding control mode radio buttons are disabled to prevent errors.

The 'subsystem status' consists of a traffic light (where green, yellow and red correspond to OK, WARN and FAIL, respectively) and a box, where further information about the subsystem status may be provided. The meaning of this terminology will be better explained in Sec. 7. Hovering on the traffic lights, radio buttons and values will show the timestamp of when the value (event) was generated. For instance, in Figure 33, the SEAM4US fan control has been stopped because no relevant environmental sensor values have been received in the last 24 hours.



Control

	Control mode	Status
LIGHS	SEAM4US On [✓ Fail Safe] SEAM4US Off **Dimming levels:** Note: No updates for the last 24H	🟩
FANS	[✓ SEAM4US On] SEAM4US Off **Fan speed:** Fan 1: 0 Hz Fan 2: 0 Hz	🟩
ESCALATORS	[✓ SEAM4US On] SEAM4US Off **Escalator speed:** Full: 0.5 m/s (ESCALATOR 1)	🟩

Tip: Try hovering over labels, radio buttons, traffic lights to get timestamps

Figure 33. Example of the SEAM4US GUI 'control' tab

7. SUPERVISION

The SEAM4US architecture includes a supervision component – called 'system supervisor' or simply 'supervisor' – that notifies the user about the status of every component and, in case of errors, provides details about the issue by emailing a list of subscribers. Therefore, the purpose of the supervisor is twofold: it serves as a maintenance tool during the operational phase and as a bug tracker during the development phase.

7.1. Architecture

The architecture of the supervisor is shown in Figure 34.

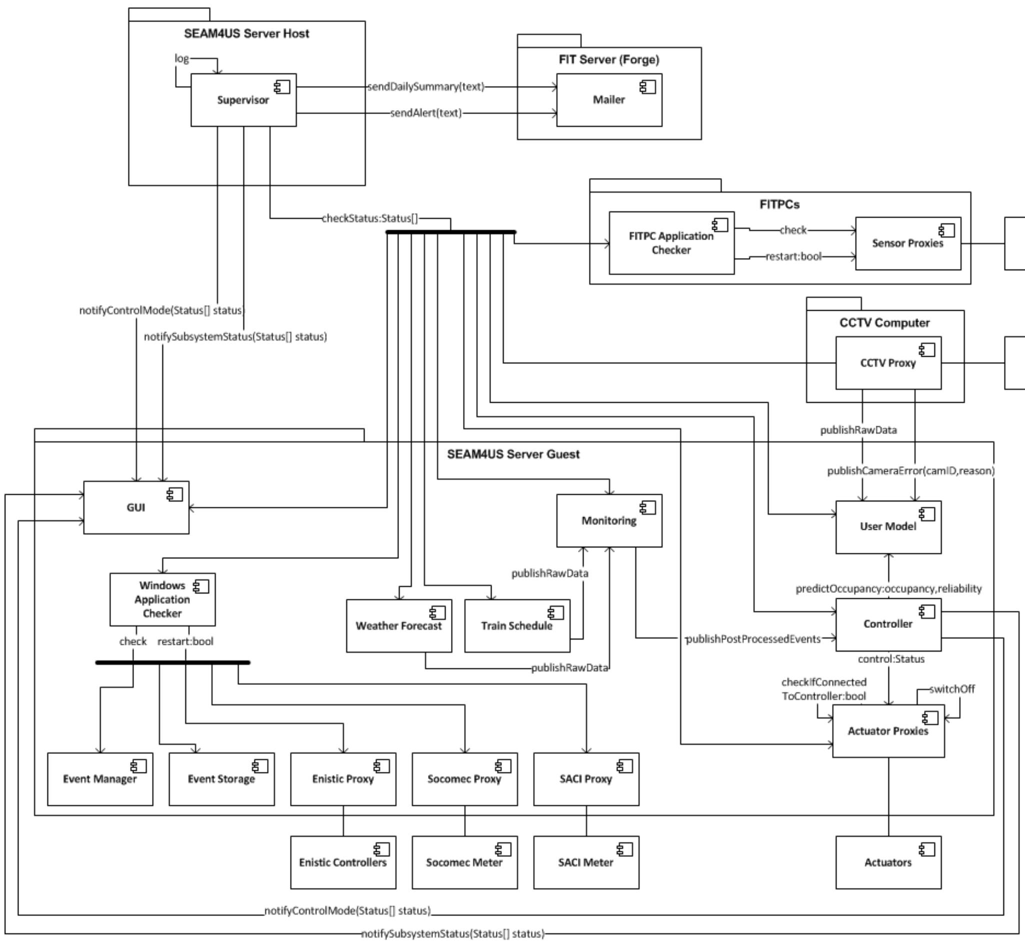


Figure 34. Architecture of the SEAM4US *system supervisor* component

The supervisor is a Java application running on the SEAM4US Server, and it has been exported to a runnable `.jar` so that the cronjob can easily execute it by `java -jar /home/admini/Supervisor.jar`. It expects the parameters `summary` and `alerts` for the two different running modes, which are described in Sec. 7.3. For debugging purposes, it is possible to provide further parameters that allow he execution of a cycle without generating a mail (parameter `nomail`) or with adding the prefix [Test] to the mail subject (parameter `test`). Furthermore, the use of a proxy to connect to the mail server can be disabled (parameter `noproxy`), so that the same code can be run on the SEAM4US Server or on the developer machines. In general, the supervision call should not only be used to report malfunctions but to automate corrective action as well.

The *mailer* subcomponent is a PHP script running on an external web server, and it is necessary because it is not possible to send mails from within the restricted TMB network. It can be reached by an HTTP call under `https://forge.fit.fraunhofer.de/seam4us/mailer.php` and it is secured by user+password authentication to avoid misuse. The *mailer* expects the subject and mail body as parameters for creating the mail.

7.2. The *Supervisable* interface

Every supervised component must implement an interface, called *supervisable*, consisting of two methods: `getName()` and `checkStatus()`. The first method returns the name of the component as a string, which will be used by the supervisor as an identifier; the second method is called in every supervision cycle and it returns a list of `Status` elements, thanks to which a component can detail the status of different subcomponents. Each `Status` element consists of three fields: i) `id`, which indicates what subcomponent the status is about; ii) `level`, which indicates whether the subcomponent is running correctly (`OK`), it has a problem that does not completely disturb the execution of the component (`WARN`), or it has a critical problem (`FAIL`); iii) `description`, which gives further details about the problem, if any, or in general about the status of the subcomponent. The static data structure of the *supervisable* interface described above is summarized in Figure 35.

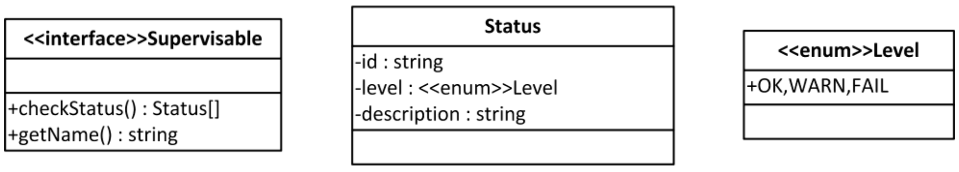


Figure 35: Static data structure of the *supervisable* interface

The supervisor can request the status of a component via a servlet provided by a separate OSGi bundle. The URL for the servlet is individualized via the component's name, which is returned by the `getName()` method. Calling the servlet triggers a call of the `checkStatus()` method; the returned list of `Status` elements is then converted into a JSON array. For OSGi components, the developer simply has to reference the supervisable bundle and implement the `Supervisable` interface. Also non-OSGi components can be supervised but they must provide the servlet and construct the JSON array themselves.

In general, the supervisable interface is integrated into the supervised component. However, in some cases, *supervisables* are new components that supervise other components. For example, we implemented the *Windows Application Checker* subcomponent that checks whether the processes of the energy proxies (i.e., *Enistic proxy*, *SOCOMEC proxy* and *SACI proxy*) and the *LinkSmart Event Manager* and *LinkSmart Event Storage Manager* run correctly (see Figure 34). If not, the *Windows Application Checker* tries to restart them and returns the success value.

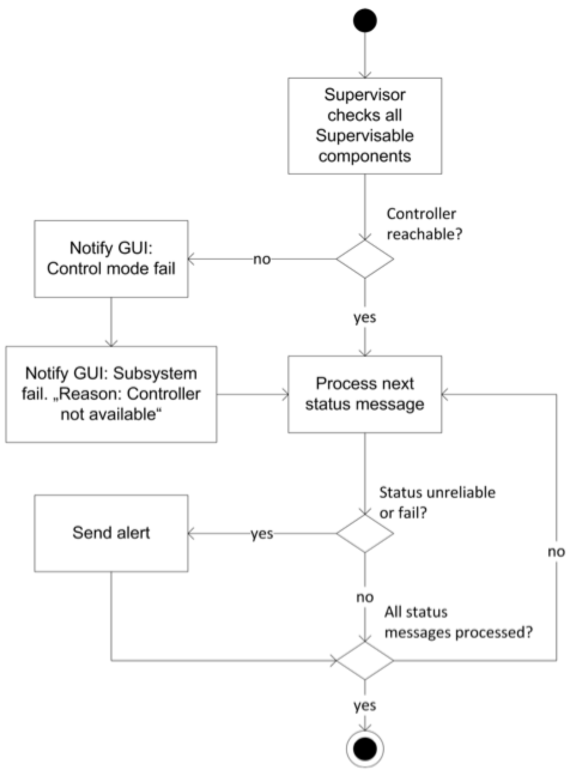
As an example, the following listing shows the return value of the *controller* supervisable servlet that includes three status elements, one for each subcomponent (escalator controller, light controller, and fan controller).

```
[{"id":"Escalator controller status","level":"OK","description":""},
{"id":"Light controller status","level":"WARN","description":"Light
control has been manually disabled"}, {"id":"Fan controller
status","level":"FAIL","description":"Sensor feed from
topic:event/postProcessed/temperature/passeigDeGracia/line3/Cni has a
confidence below the threshold. Fan control stopped."}]
```

In this case, both the escalator and the light controllers run correctly even though there is a warning about the fact that light control was switched off manually. The fan control has been stopped because the confidence level of one critical input value (i.e., the temperature for the SEAM4US spatial zone *Cni*, see Sec. 8.1) was below the desired threshold; the status level FAIL prompts the user to find out the cause of the problem in order to restart the fan control.

7.3. Working principles

This supervisor workflow, which is shown in Figure 36, is triggered every 10 minutes through a *cronjob*, which has to be configured in `/etc/crontab`.

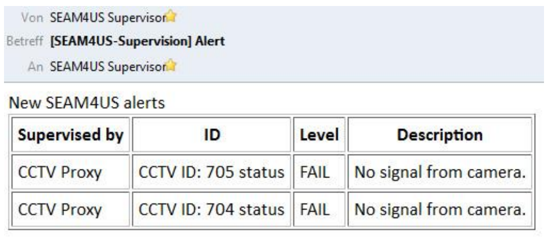


Supervisor checks all Supervisable components
Controller reachable?
no
Notify GUI: Control mode fail
yes
Notify GUI: Subsystem fail. „Reason: Controller not available"
Process next status message
Status unreliable or fail?
yes
Send alert
no
no
All status messages processed?
yes

Figure 36: Workflow of a supervision cycle

First of all, the controller reads a configuration file storing the URL of each supervised component. Modifying this configuration file allows us to disable, enable and change the supervision of one or more components even during operation. For each component, the supervisor requests the current status by calling the correspondent servlet; the response is logged. Then, the supervisor checks if a status has changed to level WARN or FAIL; if so, it creates an alert message. When all the supervised components have been processed, an email combining new alerts is sent to a list of subscribers if there is at least one of them. Note that the rationale behind this mechanism is to minimize the number of messages sent to the subscribers. However, a human operator can check the supervisor's log files at any time to monitor the alerts issued, if needed. In order to provide a daily overview of the current system status, a summary of all current statuses, independently of their levels, is sent once a day. This mechanism also serves as supervision for the supervisor: if the daily summary is missing, the human operator knows that there is a problem with the supervisor itself. The first component to be supervised is always the controller, which usually updates also the GUI. If the controller is not reachable, then the supervisor notifies the GUI that all control modes fail thus ensuring that the GUI does not show unreliable data.

All emails are sent to the distribution list `seam4us-supervision@fit.fraunhofer.de` (in order to be able to update the list of subscribers) and also archived. The supervisor constructs the mail body for alerts and daily summary mails as HTML tables so that all mail clients can present the format. Figure 37 shows an example of an alert email.


Von SEAM4US Supervisor
Betreff **[SEAM4US-Supervision] Alert**
An SEAM4US Supervisor

New SEAM4US alerts

Supervised by	ID	Level	Description
CCTV Proxy	CCTV ID: 705 status	FAIL	No signal from camera.
CCTV Proxy	CCTV ID: 704 status	FAIL	No signal from camera.

Figure 37. Example of a supervisor alert mail

8. REPRESENTATIONS

All SEAM4US components must share some knowledge regarding the representation of the PdG-L3 station and regarding the event format, which is used to exchange data. The two dedicated components that carry out these tasks are described in the following sections.

8.1. Spatial representation

The 'spatial representation' of the PdG-L3 provides a unique identification code of each area of the pilot (including portions of the corridors, for instance). Of course, TMB already uses an internal nomenclature for the spaces of the station, but it is too coarse and not comprehensive enough to be used within SEAM4US. A map of the PdG-L3 station including all 29 SEAM4US spatial zones is shown in Figure 38.

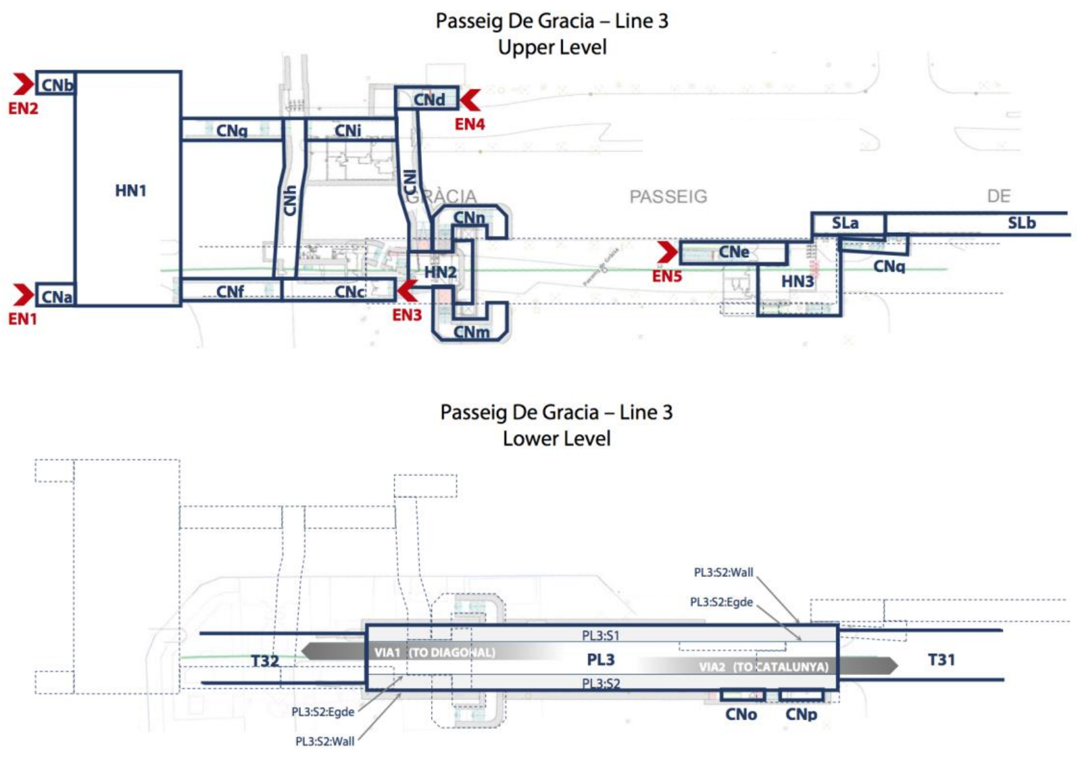


Figure 38. SEAM4US nomenclature applied over the station model

The details of the nomenclature of the SEAM4US spatial zones have been fully documented in D3.1.1. In short, each label is composed of three positions. In the first position, there is one or two capital letters characterizing the function of the zone ('E' for the entrances, 'C' for the corridors, 'H' for the halls, 'PL' for the platforms, 'SL' for the station links, and 'TL' for the tunnels); in the second position (which exists only for the corridors, the entrances, and the halls), there is a single capital letter characterizing the position of the part within the PdG station ('N' for North, which means PdG-L3, and 'S' for South, which means PdG-L2 and PdG-L4)[2]; in the third position, there is a progressive letter for the corridors, the suffixes 3:S1 and

[2] Since the pilot refers only to PdG-L3, the letter 'S' is never used in the nomenclature, but it has been included in order to create an extendable nomenclature.

3:S2 for the platforms and the tunnels, and a number for all other areas. The topology of the PdG-L3 station represented as a graph where the nodes correspond to the Pd3-L3 spatial zones is shown in Figure 39.

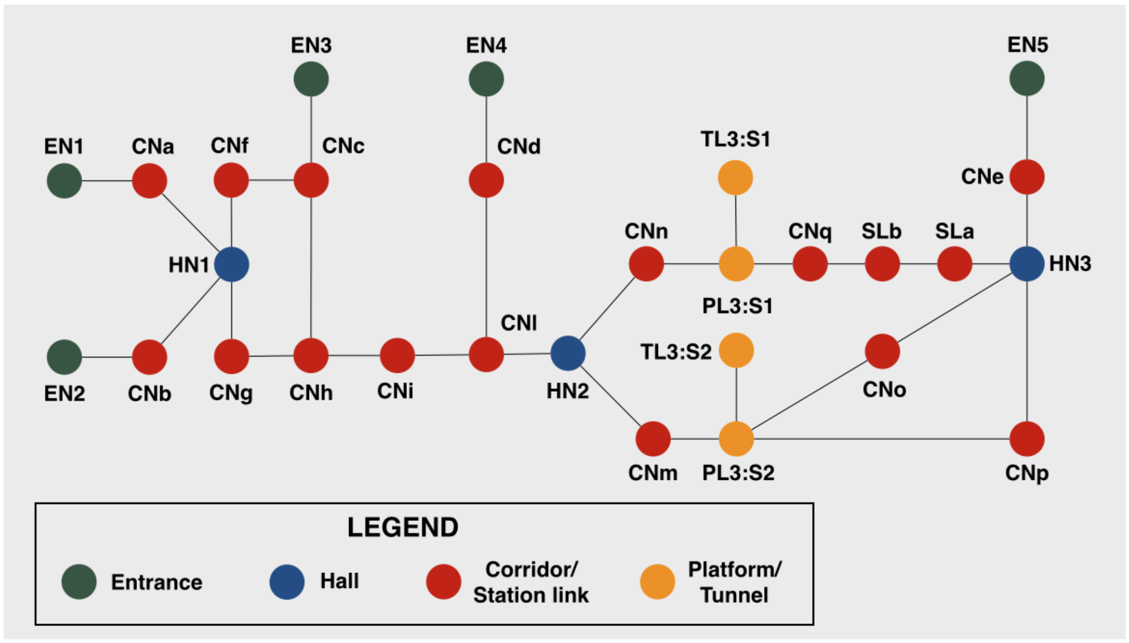


Figure 39. Topology of the PdG-L3 station represented with the SEAM4US spatial zones

The spatial representation has been coded in a JSON file which is then used by all components that need it. If such representation changes (because of maintenance in some areas or because of extensions in the pilot, for instance), it is sufficient to modify this file to keep all components running with the updated representation.

8.2. Event representation

SEAM4US components exchange information as events published via the LinkSmart middleware. There are two kinds of events: *raw data*, which are published by the monitoring proxies components and are generated to translate the proprietary format used by all monitoring devices and sensors into a standard SEAM4US format; *postprocessed data*, which are published by the monitoring components and aggregate/filter raw data, possibly coming from multiple sources. In addition to these two categories, there are other kinds of events, for instance related to the database access or to the control flow. A thorough description of all events is given in Appendix A, but in general raw data follow the format of Table 2, whereas in the postprocessed data there is an additional row regarding the confidence level of the data.

Similarly to the spatial representation, the event representation has been coded in a .JSON file which is then used by all components that need it. If such representation changes (because of maintenance in some areas or because of extensions in the pilot, for instance), it is sufficient to modify this file to keep all components running with the updated representation.

Table 2. Template for the event format of SEAM4US raw data

Key	Value type	Value	Comment
category	string		type of the sensor/device
name	string		name of the sensor/device
value	float		appropriate unit of measurement
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location of the sensor/device

9. DATABASE AND DATABASE MANAGEMENT

As shown in Figure 40, the storage database consists of four tables: the *EventItem* table (see Figure 41) stores all unmodified events, and it is then distributed into the other three tables the *Context* table (see Figure 42) stores the event name and types; the *ObservableProperties* table (see Figure 43) stores all event properties such as timestamp, speed, and location; the *Observation* table (see Figure 44) stores one value for each ObservableProperty.

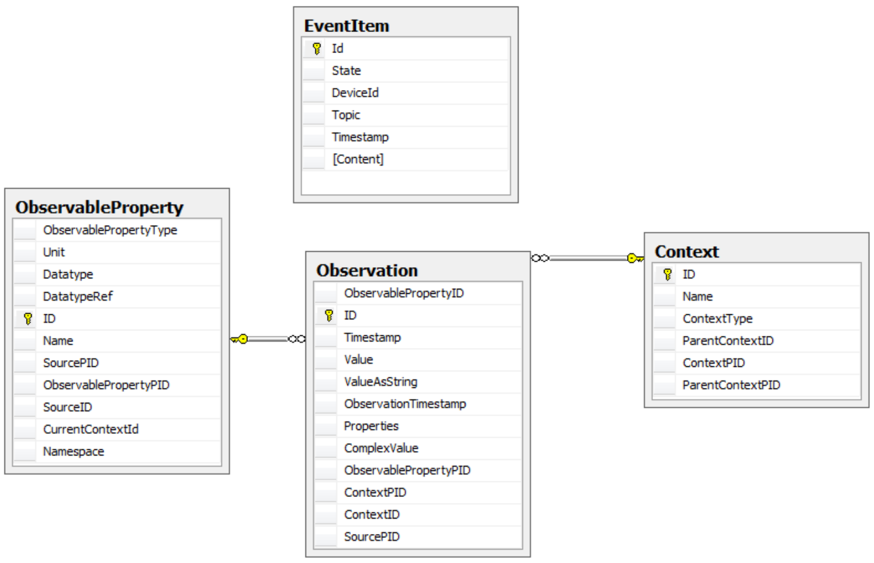
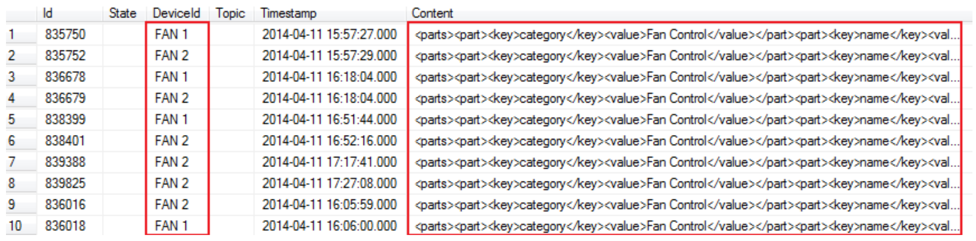
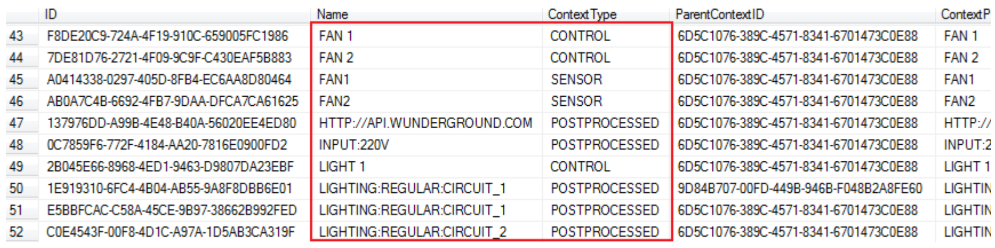


Figure 40. Structure of the SEAM4US storage DB



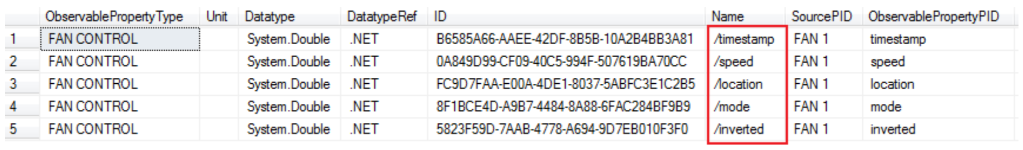
	Id	State	DeviceId	Topic	Timestamp	Content
1	835750		FAN 1		2014-04-11 15:57:27.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
2	835752		FAN 2		2014-04-11 15:57:29.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
3	836678		FAN 1		2014-04-11 16:18:04.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
4	836679		FAN 2		2014-04-11 16:18:04.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
5	838399		FAN 1		2014-04-11 16:51:44.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
6	838401		FAN 2		2014-04-11 16:52:16.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
7	839388		FAN 2		2014-04-11 17:17:41.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
8	839825		FAN 2		2014-04-11 17:27:08.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
9	836016		FAN 2		2014-04-11 16:05:59.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...
10	836018		FAN 1		2014-04-11 16:06:00.000	<parts><part><key>category</key><value>Fan Control</value></part><part><key>name</key><val...

Figure 41. *EventItem* DB table



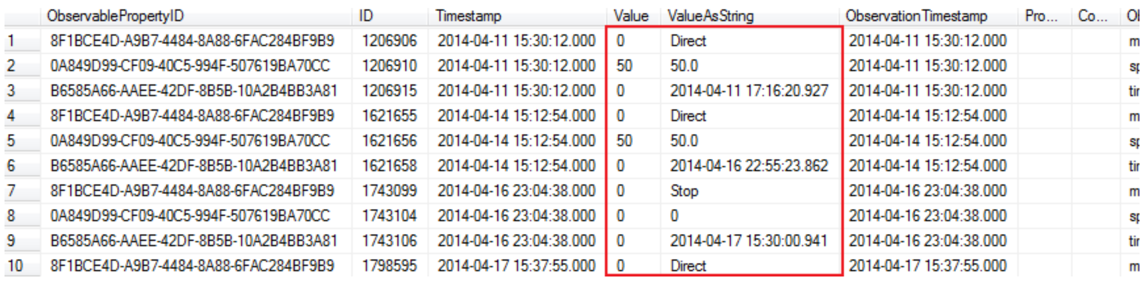
	ID	Name	ContextType	ParentContextID	ContextP
43	F8DE20C9-724A-4F19-910C-659005FC1986	FAN 1	CONTROL	6D5C1076-389C-4571-8341-6701473C0E88	FAN 1
44	7DE81D76-2721-4F09-9C9F-C430EAF5B883	FAN 2	CONTROL	6D5C1076-389C-4571-8341-6701473C0E88	FAN 2
45	A0414338-0297-405D-8FB4-EC6AA8D80464	FAN1	SENSOR	6D5C1076-389C-4571-8341-6701473C0E88	FAN1
46	AB0A7C4B-6692-4FB7-9DAA-DFCA7CA61625	FAN2	SENSOR	6D5C1076-389C-4571-8341-6701473C0E88	FAN2
47	137976DD-A99B-4E48-B40A-56020EE4ED80	HTTP://API.WUNDERGROUND.COM	POSTPROCESSED	6D5C1076-389C-4571-8341-6701473C0E88	HTTP://
48	0C7859F6-772F-4184-AA20-7816E0900FD2	INPUT:220V	POSTPROCESSED	6D5C1076-389C-4571-8341-6701473C0E88	INPUT:2
49	2B045E66-8968-4ED1-9463-D9807DA23EBF	LIGHT 1	CONTROL	6D5C1076-389C-4571-8341-6701473C0E88	LIGHT 1
50	1E919310-6FC4-4B04-AB55-9A8F8DBB6E01	LIGHTING:REGULAR:CIRCUIT_1	POSTPROCESSED	9D84B707-00FD-449B-946B-F048B2A8FE60	LIGHTIN
51	E5BBFCAC-C58A-45CE-9B97-38662B992FED	LIGHTING:REGULAR:CIRCUIT_1	POSTPROCESSED	6D5C1076-389C-4571-8341-6701473C0E88	LIGHTIN
52	C0E4543F-00F8-4D1C-A97A-1D5AB3CA319F	LIGHTING:REGULAR:CIRCUIT_2	POSTPROCESSED	6D5C1076-389C-4571-8341-6701473C0E88	LIGHTIN

Figure 42. *Context* DB table



	ObservablePropertyType	Unit	Datatype	DatatypeRef	ID	Name	SourcePID	ObservablePropertyPID
1	FAN CONTROL		System.Double	.NET	B6585A66-AAEE-42DF-8B5B-10A2B4BB3A81	/timestamp	FAN 1	timestamp
2	FAN CONTROL		System.Double	.NET	0A849D99-CF09-40C5-994F-507619BA70CC	/speed	FAN 1	speed
3	FAN CONTROL		System.Double	.NET	FC9D7FAA-E00A-4DE1-8037-5ABFC3E1C2B5	/location	FAN 1	location
4	FAN CONTROL		System.Double	.NET	8F1BCE4D-A9B7-4484-8A88-6FAC284BF9B9	/mode	FAN 1	mode
5	FAN CONTROL		System.Double	.NET	5823F59D-7AAB-4778-A694-9D7EB010F3F0	/inverted	FAN 1	inverted

Figure 43. *ObservableProperty* DB table



	ObservablePropertyID	ID	Timestamp	Value	ValueAsString	ObservationTimestamp	Pro...	Co...	Ol
1	8F1BCE4D-A9B7-4484-8A88-6FAC284BF9B9	1206906	2014-04-11 15:30:12.000	0	Direct	2014-04-11 15:30:12.000			m
2	0A849D99-CF09-40C5-994F-507619BA70CC	1206910	2014-04-11 15:30:12.000	50	50.0	2014-04-11 15:30:12.000			sp
3	B6585A66-AAEE-42DF-8B5B-10A2B4BB3A81	1206915	2014-04-11 15:30:12.000	0	2014-04-11 17:16:20.927	2014-04-11 15:30:12.000			tir
4	8F1BCE4D-A9B7-4484-8A88-6FAC284BF9B9	1621655	2014-04-14 15:12:54.000	0	Direct	2014-04-14 15:12:54.000			m
5	0A849D99-CF09-40C5-994F-507619BA70CC	1621656	2014-04-14 15:12:54.000	50	50.0	2014-04-14 15:12:54.000			sp
6	B6585A66-AAEE-42DF-8B5B-10A2B4BB3A81	1621658	2014-04-14 15:12:54.000	0	2014-04-16 22:55:23.862	2014-04-14 15:12:54.000			tir
7	8F1BCE4D-A9B7-4484-8A88-6FAC284BF9B9	1743099	2014-04-16 23:04:38.000	0	Stop	2014-04-16 23:04:38.000			m
8	0A849D99-CF09-40C5-994F-507619BA70CC	1743104	2014-04-16 23:04:38.000	0	0	2014-04-16 23:04:38.000			sp
9	B6585A66-AAEE-42DF-8B5B-10A2B4BB3A81	1743106	2014-04-16 23:04:38.000	0	2014-04-17 15:30:00.941	2014-04-16 23:04:38.000			tir
10	8F1BCE4D-A9B7-4484-8A88-6FAC284BF9B9	1798595	2014-04-17 15:37:55.000	0	Direct	2014-04-17 15:37:55.000			m

Figure 44. *Observation* DB table

Figure 45 the mechanisms to store SEAM4US events in the DBs are shown. As also depicted in the SEAM4US system architecture of Figure 1, there are currently two databases, an MSSQL and an SQLLite DB. Note that MSSQL is the primary source of database retrieval and there are currently no applications retrieving data from the SQLite DB, which is solely used for backup purposes. Each DB has its own 'event storage manager', which subscribes to all events published to the *LinkSmart Event manager* (see Sec. 3.3) so that they can be accessed later on as well. The manager for the SQLite DB is called `Seam4usStorage.exe` (located in the directory `C:\Seam4us\Seam4UsStorage\Debug`) and it is an executable file that needs to be started by the system administrator. The manager for the MSSQL DB equivalent is called *NotifyWS* and is installed in *Microsoft IIS*, and for this reason it does not need to be restarted in case of a server reboot. To stop, restart or change this application, one must start Microsoft IIS Manager and navigate to *NotifyWS*; once selected, it will be possible to administer the service application.

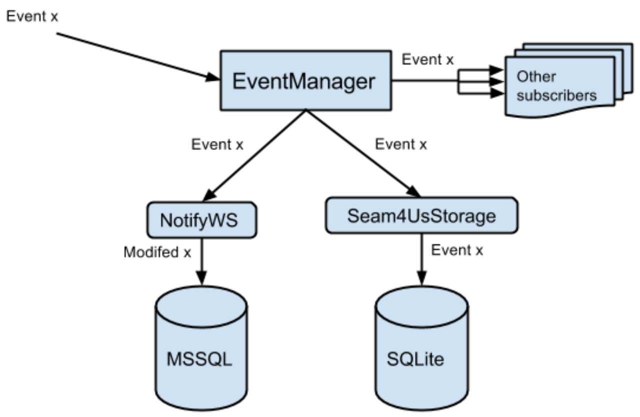


Figure 45. Event flow through the EventManager

In addition to the different storage destination, there is another substantial difference between the two managers: *NotifyWS* processes and converts key value pairs received from the *LinkSmart Event Manager* into the database structure illustrated above, so that all applications connecting to MSSQL (and especially the GUI) have an easier access to the information stored.

Accessing the database can be done with either via *Windows Authentication* (used only by the GUI) or via *SQL Authentication* (used by all other components). The rationale behind it is that using *Windows Authentication* is in general simpler in MS applications whereas using *SQL Authentication* is in general simpler in non-MS applications. The *Windows Authentication* account is the same as the user on the server. For the *SQL Authentication*, an account called *PortalReader* has been created; it can be accessed through environments such as JAVA and even though it is currently limited to data reading, it can be easily be modified to include data writing too.

10. CONCLUSIONS

This deliverable has described in detail the different components of the final SEAM4US system. The work done during the definition of the system architecture – which was documented in D4.1 – and the first version of the prototype - which was documented in D4.2.1 - served as starting points for this document. The fact that there are very minimal discrepancies between the original architecture and its actual deployment proves that all the phases of the project (from the definition of requirements to the specification of the interfaces) have been successfully executed. All component descriptions have been updated in order to reflect what is currently run in the SEAM4US servers.

The main conclusion of this deliverable is that *all* critical and non-critical components have been integrated into a prototype, which is now functional and under evaluation. A particular emphasis has been given in this deliverable to the interoperability of the components, which have been developed synergistically even though they started from different maturity levels. Also, the loose coupling among components makes it possible to use some subsystems individually (as also reported in D6.1.2) to easily create spin-off 'products' of the SEAM4US system.

It is important to emphasize that this deliverable contains thorough instructions to interact with the system via the Graphical User Interface. Also, it includes important information regarding the setup of the CCTV monitoring system, the LinkSmart middleware, and the database, which are beyond the original scopes of this deliverable but have been deemed relevant to be reported, in case the SEAM4US system will find applications besides the pilot project.

APPENDIX A - EVENT FORMAT

This appendix details the format of the events published by the 'Proxies' components (generally called 'raw data', see Sec. A.1), those published by the 'Monitoring' components (generally called 'postprocessed data', see Sec. A.2), and those to access the database (see Sec. A.3).

The names of the monitoring event follow a clear convention. The names of all environmental monitoring network events are composed of 'S' followed by a progressive number to identify the specific sensor (e.g., “S28”). The names of the cameras of the CCTV system map the official ones given by TMB. The names of the energy consumption network events (also called smart meters) are composed of 'SM' followed by a progressive number to identify the specific meter and the corresponding channel used for a certain measure separated by ':' (e.g., “SM-4:C12” identifies the smart meter "SM-4" and channel "C12"); when different power meters are metering the same equipment but for different ranges of power (i.e. different clamps), a suffix High/Low is used to identify the range. The names of the fans are composed of 'SF' followed by a progressive number to identify the specific fan (e.g., “SF1”) followed by a ":" and the "High/Low" suffix for identifying the power range for which it provides reliable values (for instance SF1:High, SF2:Low).

A.1 Raw data

A.1.1. Raw data for monitoring

Name: *Solar radiation*

Topic: *event/rawData/solarRadiation/.**

Note: *there may be different events for every location. The location can be specified in the topic in the form "event/rawData/solarRadiation/passeigDeGracia/line3"*

Table 3. Raw data format for the solar radiation

Key	Value type	Value	Comment
category	string	“Solar radiation”	
name	string		name of the sensor
solarRadiation	float		expressed in W/m^2
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *Anenometers*

Topic: *event/rawData/airSpeed/.**

Note: *there are different events for every location. The location can be specified in the topic in the form "event/rawData/airSpeed/passeigDeGracia/line3/CNq"*

Table 4. Raw data format for the anenometers

Key	Value type	Value	Comment
category	string	"Low Speed Anemometer" or "High Speed Anemometer"	the type of anemometer identifies the range of windSpeed
name	string		name of the sensor
windSpeed	float		expressed in m/s
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *Differential pressure*

Topic: *event/rawData/pressureDrop/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/rawData/pressureDrop/passeigDeGracia/line3/CNq"*

Table 5. Raw data format for the differential pressure

Key	Value type	Value	Comment
category	string	"Pressure Drop"	
name	string		name of the sensor
pressureDrop	float		expressed in Pa
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *Absolute pressure*

Topic: *event/rawData/absolutePressure/.**

Note: *there are different events for every location. The location can be specified in the topic in the form "event/rawData/absolutePressure/passeigDeGracia/line3/CNq"*

Table 6. Raw data format for the absolute pressure

Key	Value type	Value	Comment
category	string	“Absolute pressure”	
name	string		name of the sensor
absolutePressure	float		expressed in Pa
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *PM10 for lower concentrations (also called 'outdoor PM10')*

Topic: *event/rawData/OutPM10/.**

Note: *there are different events for every location. The location can be specified in the topic in the form "event/rawData/OutPM10/passeigDeGracia/line3/PL3"*

Table 7. Raw data format for the PM10 (lower concentrations)

Key	Value type	Value	Comment
category	string	“PM10”	
name	string		name of the sensor
PM10	float		expressed in mg/m^3
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	String		

Name: *PM10 for higher concentrations (also called 'indoor PM10')*

Topic: *event/rawData/IndPM10/.**

Note: *there are different events for every location. The location can be specified in the topic in the form "event/rawData/IndPM10/passeigDeGracia/line3/PL3"*

Table 8. Raw data format for the PM10 (higher concentrations)

Key	Value Type	Value	Comment
category	string	“PM10”	
name	string		name of the sensor

numOfParticles	long		number of particles
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *CO2*

Topic: *event/rawData/CO2/.**

Note: *there are different events for every location. The location can be specified in the topic in the form "event/rawData/CO2/passeigDeGracia/line3/PL3"*

Table 9. Raw data format for the CO2

Key	Value type	Value	Comment
category	string	“CO2”	
name	string		name of the sensor
concentration	float		CO2 concentration in ppm
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *Temperature*

Topic: *event/rawData/temperature/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/rawData/temperature/passeigDeGracia/line3/PL3"*

Table 10. Raw data format for the temperature

Key	Value type	Value	Comment
category	string	"Air Temperature" or "Surface Temperature"	
name	string		name of the sensor
temperature	float		expressed in ºC
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *Humidity*

Topic: *event/rawData/humidity/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/rawData/humidity/passeigDeGracia/line3/PL3"*

Table 11. Raw data format for the humidity

Key	Value Type	Value	Comment
category	string	“Humidity”	
name	string		name of the sensor
relativeHumidity	float		expressed in %
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *Power consumption*

Topic: *event/rawData/powerConsumption/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/rawData/powerConsumption/passeigDeGracia/line3/SM-4:C12"*

Table 12. Raw data format for the power consumption

Key	Value type	Value	Comment
category	string	“Power Consumption”	
name	string		
consumption	float		Apparent power expressed in VA
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		

Name: *CCTV cameras*

Topic: *event/rawData/occupancyLevel/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/rawData/occupancyLevel/passeigDeGracia/line3/PL3"*

Table 13. Raw data format for the occupancy

Key	Value type	Value	Comment
category	string	"Occupancy Level"	
name	string		name of the camera
numberOfPeople	int		no. of people focused by the camera
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
standardDeviation	int		standard deviation from the no. of people detected

Name: *Train arrivals*

Topic: *event/rawData/arrivedTrain/.**

Note: *This event is generated by a train proxy when a train arrives in platform. Just one event is generated for each train in order to allow post-processing to count arrived trains.*

Table 14. Raw data format for the train arrivals

Key	Value Type	Value	Comment
category	string	"Arrived Train"	
name	string		not used
value	float		not used
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location of platform at which the train arrives (e.g., PL3:S1)

A.1.2. Raw data for actuators

Note that the raw data for the actuators are used to monitor the state of the devices (only fans and escalators), also when the SEAM4US control is switched off.

Name: *Escalator speed*

Topic: *event/rawData/escalatorSpeed/.**

Note: *This event is generated by the escalator proxy when the speed of the escalator is changed. The escalator proxy publishes events about the current speed of the escalator using* *the LinkSmart EventManager whenever the speed of the escalator is changed. This event is sent onChange every 10 minutes.*

Table 15. Raw data format for the escalator speed

Key	Value type	Value	Comment
category	string	"Escalator Speed"	
name	string		
speed	float	{0, 0.2, 0.4, 0.5}	actual speed in m/s
status	boolean	{true, false}	true if the connection is established
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location of the escalator

Name: *Fan frequency*

Topic: *event/rawData/fanFrequency /.**

Note: *The fan frequency event is periodically generated by the controller at each control step. The output is implemented using the Event Manager.*

Table 16. Raw data format for the fan frequency

Key	Value type	Value	Comment
category	string	"Fan Frequency"	
name	string		name of the fan
frequency	float		current frequency in Hz
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location of the fan

Name: *Fan control status*

Topic: *event/rawData/fanControlStatus/.**

Note: *The Fan Control Status event is generated each time the status of the control changes from CCIF to Seam4US or vice versa. The output is implemented using the Event Manager. If the fan frequency variator is controlled manually by a technician on site for maintenance the control commands are not forwarded. If the maintenance mode is deactivated, the variator returns to the last frequency and operation mode received.*

Table 17. Raw data format for the fan control status

Key	Value type	Value	Comment
category	string	“Fan Control Status”	
name	string		name of the fan
maintanence	boolean	{True, False}	
frequency	float		current frequency in Hz
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location of the fan

A.2 Postprocessed data

Postprocessed data are computed from raw data, and they always include a field to state the confidence level of the data. Postprocessing actions includes filtering, estimation of indirect measures and resampling (synchronization).

A.2.1. Postprocessed data for monitoring

Name: *Air change rate*

Topic: *event/postProcessed/airChangeRate/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/airChangeRate/passeigDeGracia/line3"*

Table 18. Postprocessed data format for the air change rate

Key	Value type	Value	Comment
category	string	“Air Change Rate”	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
airChangeRate	float		expressed in m^3/s
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Air flow rate*

Topic: *event/postProcessed/airFlowRate/.**

Note: *there are different events for every location. The location can be specified in the topic in the form "event/postProcessed/airFlowRate/passeigDeGracia/line3/CNl"*

Table 19. Postprocessed data format for the air flow rate

Key	Value type	Value	Comment
category	string	"Air Flow Rate"	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
airFlowRate	float		expressed in m^3/s
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Air speed*

Topic: *event/postProcessed/airSpeed/.**

Note: There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/airSpeed/passeigDeGracia/line3/CNq"

Table 20. Postprocessed data format for the air speed

Key	Value type	Value	Comment
category	string	"Low Speed Anemometer" or "High Speed Anemometer"	Type of the anemometer is required to identify out of range windSpeeds
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
windSpeed	float		expressed in m/s
date	string	yyyy-mm-dd	date of acquisition; format yyyy-mm-dd
time	string	hh:mm:ss	time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *PM10*

Topic: *event/postProcessed/PM10/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/PM10/passeigDeGracia/line3/PL3"*

Table 21. Postprocessed data format for the PM10

Key	Value type	Value	Comment
category	string	“PM10”	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
PM10	float		expressed in µg/m^3
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *CO2*

Topic: *event/postProcessed/CO2/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/CO2/passeigDeGracia/line3/PL3"*

Table 22. Postprocessed data format for the CO2

Key	Value type	Value	Comment
category	string	“CO2”	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
CO2	float		expressed in ppm
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Absolute pressure*

Topic: *event/postProcessed/absolutePressure/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/absolutePressure/passeigDeGracia/line3/CNq"*

Table 23. Postprocessed data format for the absolute pressure

Key	Value type	Value	Comment
category	string	“Absolute Pressure”	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
absolutePressure	float		expressed in Pa
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Temperature*

Topic: *event/postProcessed/temperature/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/temperature/passeigDeGracia/line3/PL3"*

Table 24. Postprocessed data format for the temperature

Key	Value type	Value	Comment
category	string	"Air Temperature" or "Surface Temperature"	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
temperature	float		expressed in ºC
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Humidity*

Topic: *event/postProcessed/humidity/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/humidity/passeigDeGracia/line3/PL3"*

Table 25. Postprocessed data format for the humidity

Key	Value type	Value	Comment
category	string	"Humidity"	
name	string		corresponding real sensor at which the post-processed data refers to (if existing)
relativeHumidity	float		
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Power consumption*

Topic: *event/postProcessed/powerConsumption/.**

Note: *There are different events for every location. The location can be specified in the topic in the form "event/postProcessed/powerConsumption/passeigDeGracia/line3/PL3"*

Table 26. Postprocessed data format for the power consumption

Key	Value type	Value	Comment
category	string	"Power consumption"	
name	string		Name of equipment or lighting zone at which the consumption refers to (format).
consumption	float		active power expressed in W
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		Name of equipment or lighting zone at which the consumption refers to.
confidence	float	[0,1]	level of confidence

Name: Passenger *occupancy*

Topic: *event/postProcessed/occupancyLevel/.**

Table 27. Postprocessed data format for the passenger occupancy

Key	Value type	Value	Comment
category	string	" Occupancy Passenger Level "	
name	string		It is left empty
numberOfPeople	float		number of people in the specified location
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location of occupancy level
confidence	float	[0,1]	level of confidence of resampled data

Name: *Frequency of the trains*

Topic: *event/postProcessed/trainFrequency/.**

Table 28. Postprocessed data format for the frequency of the trains

Key	Value type	Value	Comment
category	string	"Train Frequency"	
name	string		
frequency	float		number of the trains arrived in the last 10 minutes
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		platform at which the trains arrives (e.g., "PL3:S1")

Name: *Weather station*

Topic: *event/postProcessed/weatherStation/.**

Note: *There may be different events for every location. The location can be specified in the topic in the form "event/postProcessed/weatherStation/passeigDeGracia/line3"*

Table 29. Postprocessed data format for the weather station

Key	Value type	Value	Comment
category	string	“Weather Station”	

name	string	"W11"	name of the weather station (there is only one in the pilot)
temperature	float		expressed in °C; it can be empty ("NaN")
pressure	float		expressed in Pa; it can be empty ("NaN")
relativeHumidity	float		expressed in %; it can be empty ("NaN")
windSpeed	float		expressed in m/s; it can be empty ("NaN")
windDirection	float	[0,360]	expressed in degrees; it can be empty ("NaN")
rainAmount	float		expressed in mm; it can be empty ("NaN")
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

Name: *Weather forecast*

Topic: *event/postProcessed/weatherForecast/.**

Note: *Every hour there are N new events created, each for one hour in the future. So, modules that subscribe to the event "event/postProcessed/weatherForcast/passeigDeGracia/line3/2hourahead" will get notified of the weather forecast for two hours from that moment on.*

Table 30. Postprocessed data format for the weather forecast

Key	Value type	Value	Comment
category	string	"Weather Forecast"	
name	string	"http://api.wunderground.com"	Name of the weather station
temperature	float		expressed in °C, it can be empty ("NaN")
pressure	float		expressed in Pa, it can be empty ("NaN")
relativeHumidity	float		expressed in %, it can be empty ("NaN")
windSpeed	float		expressed in m/s, it can be empty ("NaN")
windDirection	float		expressed in degrees, it can be empty ("NaN")
windDirectionString	string		e.g., "north-east", it can be empty ("NaN")
skyCondition	float		e.g., "partly cloudly", it can be

			empty ("NaN")
icon	string	"http://icons-ak.wxug.com/i/c/k/partlycloudy.gif"	
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
forecastDate	string		format yyyy-mm-dd
forecastTime	string		format hh:mm:ss
location	string	"Barcelona (LEBL)"	
confidence	float		level of confidence

A.2.2. Postprocessed data for control

These data are generated for updating information available at control level but are not stored in the database.

Name: *Fan frequency*

Topic: *event/postProcessed/fanFrequency/.**

Note: *This event is generated by Controller at each control step. When the Seam4US controller is active, the frequency value generated by the control algorithm is provided. When the Seam4US controller is inactive and the CCIF directly control fans, this event provides CCIF scheduled frequencies (this is always the case for the uncontrolled fans).*

Table 31. Postprocessed data format for the fan frequency

Key	Value type	Value	Comment
category	string	" Fan Frequency "	
name	string	{SF1,SF2, TF1, TF2}	name of the fan
frequency	float		expressed in Hz. If the fan is inverted this value must be negative.
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location at which the fan refers to

Name: *Fan control status*

Topic: *event/postProcessed/fanControlStatus/.**

Note: *This event is generated by Controller at each control step*

Table 32. Postprocessed data format for the fan control status

Key	Value type	Value	Comment
category	string	" Fan Control Status "	
name	string	{SF1,SF2, TF1, TF2}	name of the fan
status	boolean	{True, False}	SEAM4US active (true) or not (false)
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		location at which the fan refers to

Name: *Update of the passenger occupancy*

Topic: *event/postProcessed/occupancyLevelUpdate/.**

Table 33. Postprocessed data format for the update of the passenger occupancy

Key	Value type	Value	Comment
category	string	"Train Frequency"	
name	string		
numberOfPeople	float		number of people in the specified location, updated by the 'passenger model' component component with the latest CCTV camera available
date	string		date of acquisition; format yyyy-mm-dd
time	string		time of acquisition; format hh:mm:ss
location	string		
confidence	float	[0,1]	level of confidence

A.3. DB access events

Name: *DB request*

Topic: *event/db/getPostProcessed/passeigDeGracia/line3/.**

Table 34. Format for the DB request

Key	Value Type	Value	Comment
category	String		CSV string

location	String		CSV string
name	String		CSV string (optional)
dateStart	String		start date of acquisition; format yyyy-mm-dd
timeStart	String		start time of acquisition; format hh:mm:ss
dateEnd	String		end date of acquisition; format yyyy-mm-dd
timeEnd	String		end time of acquisition; format hh:mm:ss

Name: *DB response*

Topic: *event/db/postProcessed/passeigDeGracia/line3/.**

Table 35. Format for the DB response

Key	Value Type	Value	Comment
category	String		CSV string
location	String		CSV string
name	String		CSV string (optional)
value	String(float)		CSV string
date	String		date of acquisition; format yyyy-mm-dd
time	String		time of acquisition; format hh:mm:ss
confidence	String(float)		CSV string

APPENDIX B - METHODS

B.1 Actuators

Escalators

- `public boolean setDesiredSpeed(boolean lowSpeed);`
    - Function: it sets the desired speed.
    - Parameters: lowSpeed - If true, the desired speed is low speed (0.4 m/s). If false, the desired speed is full speed (0.5 m/s).
    - Returns: Whether the command has been successfully executed. If not connected false is returned.
- `public boolean isConnected();`
    - Function: it checks the connection state of the proxy.
    - Returns: Whether the PLC is connected and reachable from the server.
- `public boolean isSeam4UsActive();`
    - Function: it checks whether SEAM4US mode is activated in the switch on the cabinet door. We do not actually know the status of this switch, so this value is just a guess from observing the state of the PLC.
    - Returns: Whether SEAM4US mode is activated. If not connected false is returned.
- `public void switchToDefaultOperation();`
    - Function: it switches the actuator to default operation. This will set the "Desired speed = 0.4 m/s" output off. The setting remains until the setDesiredSpeed() method is called.

Fans

- `public double setVariatorFrequencyInHz(int id, double value, boolean inverted)`

  This methods allows to set the ventilator's speed. If the proxy cannot control the fans due to CCIF policy, maintenance or technical issues, the value is stored and set as soon as the control is given to the SEAM4US system. Frequency in Hz (ranging from 0.0 Hz = full stop to 50.0Hz = full speed) Frequency levels between 0.0 and 10.0 Hz are set to 0.0 Hz. Frequency levels between 10.0 Hz and 20.0 Hz are set to 20.0 Hz. If "inverted" is set to true the direction of the fans is inverted. In addition to setting the frequency of the variator with every call of the setVariatorFrequencyInHz method the system is set to ready as described in the setSystemReady method.

- `public boolean isSeam4UsActive(int id)`

  This methods returns the actual mode of control. If true is returned, the switching logic has been set by the CCIF operator to SEAM4US and the SEAM4US system is controlling the fans at the moment.

- `public boolean isConnected(int id)`

This methods returns the state of connection between the server and the installed hardware in the ventilation room. During runtime the connection state is monitored continuously. If the connection is established, true is returned, otherwise false is returned.

- `public boolean setSystemReady(int id, boolean ready)`

  This method sets the system state indicator connected to the CCIF system. If set to false the system returns into the default CCIF mode. This means for reactivating SEAM4US Ready, the method has to be called with the ready parameter set to true, and then the local CCIF operator has to acknowledge the SEAM4US mode. If the PLC is not connected or reachable, the command will be set as soon as the connection is established. The method returns true if the value has successfully be sent to the controller, or return false if the value could not be transmitted.

- `public void switchToDefaultOperation(int id)`

  With this method the actuator is switched to default operation which is CCIF mode. This will set the "SEAM4US Ready" to false for the selected fan. This setting remains until either the `setSystemReady()` or the `setVariatorFrequencyInHz()` method is called for this particular fan.

Lights

- `public String[] getNodes();`
  - Function: it gets a list of all nodes.
  - Returns: the addresses of all nodes, as an array of Strings.
- `public String[] getZones();`
  - Function: it gets a list of all zones.
  - Returns: the identifiers of all zones, as an array of Strings.
- `public String[] getNodesInZone(String zone);`
  - Function: it gets a list of all nodes in a zone.
  - Parameters: zone - the identifier of the zone to be queried, as a String.
  - Returns: the addresses of all nodes in the zone, as an array of Strings.
- `public String[] getZonesForNode(String node);`
  - Function: it gets a list of the zones assigned to the node.
  - Parameters: node - the address of the node to be queried, as a String.
  - Returns: the addresses of all zones of the node, as an array of Strings.
- `public boolean getSwitchingStatus(String node);`
  - Function: it gets the switching status (on/off) for the given node.
  - Parameters: node - the address of the node to be queried, as a String.
  - Returns: true if the node is currently switched on, false otherwise
- `public int getDimmingLevel(String node);`
  - Function: it gets the dimming level for the given node.
  - Parameters: node - the address of the node to be queried, as a String.
  - Returns: the current dimming level, as an int from the range [0,100]
- `public void setSwitchingStatus(String node, boolean status);`

    - Function: it sets the switching status (on/off) for the given node.
    - Parameters: node - the address of the node to be actuated, as a String; status - whether to switch the node on (true) or off (false), as a boolean
- `public void setDimmingLevel(String node, int level);`
    - Function: it sets the dimming level for the given node.
    - Parameters: node - the address of the node to be actuated, as a String; level: the desired dimming level, as an int from the range [0,100]
- `public void setSwitchingStatusForZone(String zone, boolean status);`
    - Function: it sets the switching status (on/off) for the entire zone given.
    - Parameters: zone - the identifier of the zone to be actuated, as a String; status - whether to switch the zone on (true) or off (false), as a boolean
- `public void setDimmingLevelForZone(String zone, int level);`
    - Function: it sets the dimming level for the entire zone given.
    - Parameters: zone - the identifier of the zone to be actuated, as a String; level: the desired dimming level, as an int from the range [0,100]
- `public void switchToDefaultOperation();`
    - Function: it switches the actuator to default operation. This will set all lights to dimming level 100 (full brightness). The setting remains until any other method is called to set the dimming level or switching status.

Notes: *In the SEAM4US pilot, the controllable lights are located in the zones 'HN2' (PdG-L3 hall with ticket validation machines), 'CNm' (corridor/staircase between platform and hall), 'PL3_Wall' and 'PL3_Edge' (platform in southward direction, allowing to control the two lines of lights separately). For the pilot operation, the method `setDimmingLevelForZone()`, which is called by the controller, also implements a limit of the dimming levels corresponding to the illuminance required by TMB's company policies.*

B.3 LinkSmart

Event manager

- public List<Event> GetEventData(string topic, string[] timespan)
    - Function: it gets a list of all events with a specific topic and during a set time period. Set timespan to null to get all events.
    - Returns: events in same format as received, as a list of Event.
- public List<RawEvent> GetRawEventData(string topic, string[] timespan)
    - Function: it gets a list of all events with a specific topic and during a set time period. Set timespan to null to get all events.
    - Returns: more structured events, as a list of RawEvent.
- public string GetChannelData(string smartMeterID, string channel, string[] timespan)
    - Function: it gets all consumption observations for a single channel given a specific timespan. Set timespan to null to get all data.
    - Returns: XML representation of consumption data
- public string GetPowerLineData(string powerLine, string[] timespan)
    - Function: it gets all consumption data associated with a specific power line during a set time period. Set timespan to null to get all data.

- Returns: XML representation of consumption data associated with a power line.

GLOSSARY AND ABBREVIATIONS

- **CCTV**: Closed-Circuit Television
- **CSV**: Comma-Separated Values, which is a file storing tabular data in plain-text form
- **DB**: Database
- **GUI**: Graphical User Interface
- **IIS**: Internet Information Server
- **JDK**: Java Development Kit
- **JRE**: Java Runtime Environment
- **MS**: Microsoft
- **NTP**: Network Time Protocol
- **OSGi**: Open Service Gateway initiative, describes a modular system and a service platform for the Java programming language that implements a complete and dynamic component model
- **PdG-L3**: Passeig de Gracia - L3, which is the location of the pilor project
- **PM10**: Particulate Matter smaller than about 10 micrometers
- **TMB CCIF**: Spanish acronym for the TMB SCADA
- **TMB SCADA:** Main control center of TMB infrustructures in the pilot project
- **WSN**: Wireless Sensor Network