



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

SmartSantander – WP4

Working Document

D4.2 Description of implemented IoT services

Contractual Date of Delivery: 30th December 2012

Actual Date of Delivery: 19th December 2012

Editor(s): *The Alexandra Institute (AI)*

Author(s): See Authors list

Participant(s): ALU-IT, UC, CTI, AI, SAN,TID, TTI, UniS, EYU

Work package: WP4

Estimated person months: ALU-IT: 1.5; UC: 9.25; CTI: 3.2; AI: 19.17; SAN: 2.6; TID: 1.1; TTI: 2.64; UniS: 1.5; EYU: 2.0

Security: Public

Version: 1.00

Abstract: This document covers the developed use case scenarios in the SmartSantander project. The main purpose of this document is to describe the development of services and applications, as well as the tools available for further service development on top of the SmartSantander framework.

Keyword list: Software Analysis, Software Development, Services, Applications, UML.

Disclaimer: This document reflects the contribution of the participants of the research project SmartSantander. The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only. This document is provided without any warranty and does not constitute any commitment by any participant as to its content, and specifically excludes any warranty of correctness or fitness for a particular purpose. The user will use this document at the user's sole risk.



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

Authors

| Partner | Name | E-mail |
|---------|--|--|
| AI | João Fernandes | joao.fernandes@alexandra.dk |
| UniS | Michele Nati Alex Gluhak | m.nati@surrey.ac.uk a.gluhak@surrey.ac.uk |
| TID | Demetrio Martinez | dmolano@tid.es |
| UC | Luis Muñoz Javier Casanueva Verónica Gutierrez | luis@tmat.unican.es jcasanueva@tmat.unican.es veronica@tmat.unican.es |
| CTI | Evangelos Theodoridis Georgios Mylonas | theodori@cti.gr mylonasg@cti.gr |
| ALU-IT | Monica Russo Carmine Lausi | monica.russo@alcatel-lucent.com carmine.lausi@alcatel-lucent.com |
| EYU | Srdjan Krco Stevan Jokic | srdjan.krco@ericsson.com stevan.jokic@gmail.com |
| SAN | Tomás Garcia Fresno | tomasgarcia@ayto-santander.es |



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

| Partner | Name | E-mail |
|---------|-----------------|--|
| TTI | Ramón Bezanilla | rbezanilla@ttinorte.es |



Table of Contents

- LIST OF FIGURES6
- LIST OF TABLES10
- ACRONYMS AND ABBREVIATIONS10
- EXECUTIVE SUMMARY13
- 1. INTRODUCTION14
- 2. LIST OF IMPLEMENTED SERVICES AND APPLICATIONS.....16
- 3. METHODOLOGY FOR DESCRIPTION OF SERVICES AND APPLICATIONS18
- 4. IOT SERVICES.....20
 - 4.1. SMART PARKING SERVICE AND APPLICATIONS20
 - 4.1.1. System overview.....20
 - 4.1.2. Applications Description.....23
 - 4.1.2.1. Parking iOS, Android and Windows mobile Application.....23
 - 4.1.2.2. GIS application.....27
 - 4.2. ENVIRONMENTAL MONITORING28
 - 4.2.1. Applications Description.....31
 - 4.2.1.1. Environmental iOS, Android and Windows mobile Application.....31
 - 4.3. PARTICIPATORY SENSING33
 - 4.3.2. Applications Description.....41
 - 4.3.2.1.1. Pace of The City iOS and Android mobile Application.....41
 - 4.3.2.2. Pace of the city: Integration with Incidences management Service in the Santander City Council
45
 - 4.3.3. UNIVERSAL ALERT SYSTEM (UAS) SERVICE46
 - 4.3.3.1. UAS functional diagram46
 - 4.3.3.2. UAS mediator47
 - 4.3.3.3. User Interaction with the Universal Alert System.....48
 - 4.3.3.3.5. Exit56
 - 4.4. AUGMENTED REALITY57
 - 4.3.4. Applications Description.....63
 - 4.4.3.3. SmartSantanderRA IOS and Android mobile Application.....63
 - 4.5. IRRIGATION72



PUBLIC, SMARTSANTANDER PROJECT

4.5.1. Objectives of Service..... 72

4.5.2. System Overview 74

4.5.3. Monitoring and Administration Web Application..... 76

4.5.4. Smartphone Application..... 79

4.6. PUBLIC TRANSPORTATION 81

 4.6.2. Public Transportation application 84

4.7. SMART METERING 86

 4.7.1. System behaviour 90

 4.7.2. SmartMetering applications..... 94

5. REALISATION OF SERVICES ON TOP OF SMARTSANTANDER PLATFORM 98

5.1. USN: WEB SERVICE INTERFACES..... 98

 5.1.1. Protocols used by USN Gateway..... 99

 5.1.1.1. SensorML & O&M (Observations and Measurements) 99

 5.1.1.2. Light Sensor Protocol 99

 5.1.1.3. Ultralight Sensor Protocol 99

 5.1.1.4. Registration request..... 100

 5.1.1.5. Measure notification request 101

 5.1.1.6. Examples using SensorAPI 101

 5.1.1.7. Using LightSensorAPI& UltraLightSensorAPI..... 104

 M2M SOAP API..... 106

 5.1.1.8. Detailed functional description 106

 SOAP CLIENT SKELETON USING AXIS2 & ECLIPSE 113

 5.1.1.9. Implementation..... 113

5.2. REPOSITORY FOR PARTICIPATORY SENSING..... 114

5.3. REPOSITORY FOR AUGMENTED REALITY 117

6. INNOVATION INCUBATOR..... 120

7. CONCLUSIONS 121

 REFERENCES..... 122

 [1] L. Muñoz et al., “Boosting Smart Cities through the Synergies with the Utilities”, submitted to the IEEE Communications Magazine special issue on Smart Cities, January 2013..... 122

APPENDIX..... 123

PUBLIC, SMARTSANTANDER PROJECT

List of Figures

FIGURE 1: SMARTSANTANDER TASK 4.2 OVERVIEW..... 14

FIGURE 2: PARKING USE CASE DIAGRAM – CITIZENS’ PERSPECTIVE 20

FIGURE 3: PARKING - DEPLOYMENT DIAGRAM..... 21

FIGURE 4: PARKING - PUBLISH-SUBSCRIBE CONNECTION PARKING SERVER AND USN 21

FIGURE 5: PARKING SERVICE - INTERACTION BETWEEN SERVICE AND USN - SEQUENCE DIAGRAM..... 22

FIGURE 6: PARKING SERVICE - INTERACTION BETWEEN SERVER AND MOBILE APPLICATION - COMPONENT DIAGRAM..... 22

FIGURE 7: PARKING SERVICE - WEB SERVICE DATA STRUCTURES 23

FIGURE 8: PARKING SERVICE – GETPARKINGLOTS CALL..... 24

FIGURE 9: MAP VIEW (ANDROID,IOS AND WINDOWS MOBILE) 24

FIGURE 10: PARKING LOTS LIST VIEW (ANDROID, IOS AND WINDOWS MOBILE) 25

FIGURE 11: PARKING LOT DETAIL (ANDROID, IOS AND WINDOWS MOBILE) 25

FIGURE 12: MAPS LOADED DISPLAYING ROUTE TO THE PARKING LOT 26

FIGURE 13: ALERT VIEW: EXISTING NUMBER OF PARKING SPACES 26

FIGURE 14: GIS APPLICATION SCREENSHOT..... 27

FIGURE 15: ENVIRONMENTAL MONITORING – USE CASE DIAGRAM..... 28

FIGURE 16: ENVIRONMENTAL MONITORING - DEPLOYMENT DIAGRAM 28

FIGURE 17: ENVIRONMENTAL MONITORING – PUBLISH-SUBSCRIBE CONNECTION ENVIRONMENTAL MONITORING SERVER AND USN 29

FIGURE 18: ENVIRONMENTAL SERVICE - INTERACTION BETWEEN SERVICE AND USN - SEQUENCE DIAGRAM..... 30

FIGURE 19: ENVIRONMENTAL MONITORING SERVICE - INTERACTION BETWEEN SERVER AND MOBILE APPLICATION - COMPONENT DIAGRAM..... 30

FIGURE 20: ENVIRONMENTAL MONITORING SERVICE - WEB SERVICE DATA STRUCTURES 31

FIGURE 21: ENVIRONMENTAL MONITORING – GETENVIRONMENTALDATA CALL 31

FIGURE 22: ENVIRONMENTAL MONITORING MAP (ANDROID AND IOS) 32

FIGURE 23: ENVIRONMENTAL MONITORING MEASURED VALUES (ANDROID AND IOS) 32

FIGURE 24: PARTICIPATORY SENSING – CITIZEN USE CASE DIAGRAM 35

FIGURE 25: PARTICIPATORY SENSING SCENARIO - DEPLOYMENT DIAGRAM 36

FIGURE 26: SEQUENCE DIAGRAM: CREATE DEVICE 37

FIGURE 27: SEQUENCE DIAGRAM: PHYSICAL SENSING..... 37

FIGURE 28: SEQUENCE DIAGRAM: EVENT PUBLICATION..... 38

FIGURE 29: SEQUENCE DIAGRAM: NOTIFICATION..... 39

PUBLIC, SMARTSANTANDER PROJECT

FIGURE 30: SEQUENCE DIAGRAM: SUBSCRIBE..... 39

FIGURE 31: SEQUENCE DIAGRAM: USER SUBSCRIPTION ON UAS 39

FIGURE 32: SEQUENCE DIAGRAM: UNSUBSCRIBE 40

FIGURE 33 SEQUENCE DIAGRAM: END USER REVOKES THE SUBSCRIPTION 40

FIGURE 34: SEQUENCE DIAGRAM: DELETE DEVICE..... 41

FIGURE 35: PACE OF THE CITY APP: MAIN VIEW..... 42

FIGURE 36: PACE OF THE CITY APP: ADD EVENT VIEW..... 43

FIGURE 37: PACE OF THE CITY APPLICATION: EVENTS MAP VIEW 43

FIGURE 38: PACE OF THE CITY APP: EVENTS FILTERING VIEW 44

FIGURE 39: PACE OF THE CITY APP: SUBSCRIPTIONS VIEW..... 44

FIGURE 40: PACE OF THE CITY APP: SUBSCRIPTION VIEW SUBSCRIBE 45

FIGURE 41: UAS FUNCTIONAL DIAGRAM 46

FIGURE 42 ACCESS TO THE SYSTEM..... 49

FIGURE 43 USER MENU..... 50

FIGURE 44 PERSONAL DATA 51

FIGURE 45 USER REFERENCES 52

FIGURE 46 LIST OF SUBSCRIBED EVENTS..... 53

FIGURE 47 EVENT SUBSCRIPTION 54

FIGURE 48 DAYS OF THE WEEK AND TIME (DEFAULT VALUES)..... 55

FIGURE 49 SELECTION OF DAYS OF THE WEEK AND TIME..... 56

FIGURE 50: AUGMENTED REALITY – CITIZEN USE CASE DIAGRAM..... 59

FIGURE 51: AUGMENTED REALITY SCENARIO - DEPLOYMENT DIAGRAM..... 60

FIGURE 52: SEQUENCE DIAGRAM: REGISTER DEVICE 61

FIGURE 53: SEQUENCE DIAGRAM: OBSERVATION SENSING..... 62

FIGURE 54: SEQUENCE DIAGRAM: DELETE DEVICE..... 63

FIGURE 55: SMARTSANTANDERRA: HOME VIEW..... 64

FIGURE 56: SMARTSANTANDERRA: SETTINGS VIEW (IOS VS ANDROID)..... 65

FIGURE 57: SMARTSANTANDERRA: SANTANDER VIEW. GETTING BEACH DATA..... 65

FIGURE 58: SMARTSANTANDERRA: SANTANDER VIEW. GETTING TRAFFIC DATA..... 66

FIGURE 59: SMARTSANTANDERRA: SANTANDER VIEW. GETTING NUMBER OF INTEREST DATA..... 66

FIGURE 60: SMARTSANTANDERRA: SANTANDER VIEW. GETTING SPORT FACILITIES DATA..... 67

FIGURE 61: SMARTSANTANDERRA: AR TRANSPORT VIEW. BUS STOP SELECTED 67

FIGURE 62: SMARTSANTANDERRA: EXTRA INFO ACCESSIBLE FROM AR VIEW..... 68

FIGURE 63: SMARTSANTANDERRA: POI LIST VIEW..... 69

FIGURE 64: SMARTSANTANDERRA: POI MAP VIEW..... 69

FIGURE 65: SMARTSANTANDERRA: QR READER VIEW..... 70

PUBLIC, SMARTSANTANDER PROJECT

FIGURE 66: SMARTSANTANDERRA: NFC READER VIEW 71

FIGURE 67 PRECISION IRRIGATION SERVICE USE-CASE DIAGRAM 74

FIGURE 68: IRRIGATION ARCHITECTURE 75

FIGURE 69 SOFTWARE COMPONENTS..... 75

FIGURE 70: CONFIGURATION PAGE OF THE IRRIGATION SERVICE 77

FIGURE 71: REGION MANAGEMENT OF THE IRRIGATION SERVICE 78

FIGURE 72: REGION REPORT 79

FIGURE 73 IRRIGATION MOBILE APP – NODE MONITORING..... 80

FIGURE 74: IRRIGATION MOBILE APPLICATION – NODE REPORT AND SETUP..... 80

FIGURE 75: PUBLIC TRANSPORTATION – USE CASE DIAGRAM..... 82

FIGURE 76: COMPONENTS IN THE PUBLIC TRANSPORTATION SCENARIO 83

FIGURE 77. EcoBUS-SMARTSANTANDER INTEGRATION SEQUENCE..... 84

FIGURE 78. PUBLIC TRANSPORTATION MONITORING APPLICATION: REAL-TIME BUS LOCATION AND ENVIRONMENTAL MEASUREMENTS..... 85

FIGURE 79: SMARTMETERING – DIFFERENT INVOLVED ACTORS..... 87

FIGURE 80: SMARTMETERING – USE CASE DIAGRAM USER PERSPECTIVE 88

FIGURE 81: SMARTMETERING – USE CASE DIAGRAM AUTHORITIES PERSPECTIVE..... 88

FIGURE 82: SMARTMETERING – USE CASE DIAGRAM TECHNICAL USER PERSPECTIVE..... 89

FIGURE 83 ARCHITECTURE COMPONENTS 90

FIGURE 84 SEQUENCE DIAGRAM: SUBSCRIPTION 92

FIGURE 85: SEQUENCE DIAGRAM: NOTIFICATION..... 93

FIGURE 86 SEQUENCE DIAGRAM: SHARING AND STORAGE..... 94

FIGURE 87 MYECOFOOTPRINT GADGET AND DASHBOARD 95

FIGURE 88 SMARTENERGY APP: REGISTRATION AND INEFFICIENCY NOTIFICATION..... 96

FIGURE 89 SMARTENERGY APP: REAL-TIME ENERGY CONSUMPTION NOTIFICATION 97

FIGURE 90: SENSOR REGISTRATION FLOW..... 100

FIGURE 91: MEASURE FLOW..... 101

FIGURE 92: REGISTER_TEST.REQ.XML TEMPLATE REGISTER EXAMPLE USING SENSORML 103

FIGURE 93: OBSERVATION_TEST.REQ.XML” TEMPLATE OBSERVATION EXAMPLE USING SENSORML..... 104

FIGURE 94: REGISTER_TEST.REQ.XML” TEMPLATE REGISTER EXAMPLE USING LIGHTSENSORAPI..... 105

FIGURE 95: OBSERVATION_TEST.REQ.XML” TEMPLATE OBSERVATION EXAMPLE U USING LIGHTSENSORAPI 106

FIGURE 96: PARAMETERS OF THE GETDATA BY SERVICE OPERATION 110

FIGURE 97: PACE OF THE CITY ARCHITECTURE..... 114

FIGURE 98: CLASS DIAGRAM - DEVICEDTO..... 115

FIGURE 99: CLASS DIAGRAM - OBSERVATION 116

FIGURE 100: AUGMENTED REALITY ARCHITECTURE 117



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

| | |
|---|-----|
| FIGURE 101: CLASS DIAGRAM - DEVICEDTO | 118 |
| FIGURE 102: URBAN PLATFORM..... | 120 |

List of Tables

TABLE 1: LIST OF SERVICES AND APPLICATIONS 17

Acronyms and Abbreviations

| | |
|------|-------------------------------|
| KPI | Key Performance Indicator |
| UCD | Use Case Diagram |
| UML | Unified Modeling Language |
| PDA | Personal Digital Assistant |
| PC | Personal Computer |
| USN | Ubiquitous Sensor Network |
| SOAP | Simple Object Access Protocol |
| GIS | Geographic Information System |
| GPS | Global Positioning System |
| SMS | Short Message Service |
| UUID | Universally Unique Identifier |
| UAS | Universal Alert System |
| GZ | Geographical Zones |



PUBLIC, SMARTSANTANDER PROJECT

| | |
|------|---------------------------------|
| MT | Message Text |
| MN | Message Number |
| DB | Database |
| RA | Recorded Announcement |
| JAR | Java Archive |
| M2M | Machine to Machine |
| SW | Software |
| Pol | Point of Interest |
| AR | Augmented Reality |
| OS | Operating System |
| NFC | Near Field Communication |
| CS | Computer Science |
| REST | Representational State Transfer |
| HTML | HyperText Markup Language |
| JSON | Javascript Object Notation |
| RDF | Resource Description Framework |
| IoT | Internet of Things |
| RD | Resource Directory |
| LCD | Liquid Crystal Display |
| GUI | Graphical User Interface |

PUBLIC, SMARTSANTANDER PROJECT

| | |
|---------|------------------------------------|
| HTTP | Hypertext Transfer Protocol |
| O&M | Observation and Measurements |
| OGC | Open Geospatial Consortium |
| URL | Uniform Resource Locator |
| QR Code | Quick Response Code |
| XML | Extensible Markup Language |
| GW | Gateway |
| IP | Internet Protocol |
| IDAS | Intelligence Data Advance Solution |
| IDE | Integrated Development Environment |
| DTO | Data Transfer Object |
| WSDL | Web Services Description Language |
| XSD | XML Schema |
| JDK | Java Development Kit |
| URI | Uniform Resource Identifier |

Table 1: Acronyms and Abbreviations



EXECUTIVE SUMMARY

The smart city paradigm covers many disciplines from both technological and societal perspectives. Citizens play a major role in this paradigm as the final recipients of the services supported by the associated infrastructure.

In WP4 we assess the service umbrella of the SmartSantander initiative. In the first 18 months of the project we analysed and designed use cases that were selected and prioritised in consultation with the local authorities, regional government and according to user/citizen preferences.

In task 4.2 we realise these use cases by initially elaborating a more detailed technical specification of the solutions. In these specifications we describe the important functionalities of such use case and the envisioned software components to be implemented. All this information supports the next phase of implementation where we develop and integrate the software components. The development of the services and applications is done using a user-driven methodology where users help design and validate the implemented services (also part of task 4.3 services and applications evaluation).

Over a duration 20 months (month 8 to 28) of the project, we have realised various use cases that utilise not only the Santander deployment but also the Belgrade and Guilford testbed facilities. The covered use cases were parking control, environmental monitoring, participatory sensing, augmented reality, irrigation, public transportation, smart metering, among others. In some of the use cases, we have developed prototype applications/services whilst complete service and application(s) have been produced for a number of use cases. These have been made available for download and use by the citizens.

The objective of this deliverable is to describe the implemented services and applications with a technical perspective, this also includes the architecture of the developed components and specification of their integration with the SmartSantander platform. In order to facilitate the improvement of the existing services and the development of new and innovative services we describe the implemented tools the service/application developers can make use of in order to build their solutions. We also provide an extended description of the developed applications, as well as guidance on how to use them.

1. INTRODUCTION

This document reports on the development work carried out in WP4 for the period from M8 to M28 in Task 4.2. The objectives of this task were to develop and deploy the IoT based applications and services analysed and selected in task 4.1. The following figure shows the workflow between the tasks in WP4:

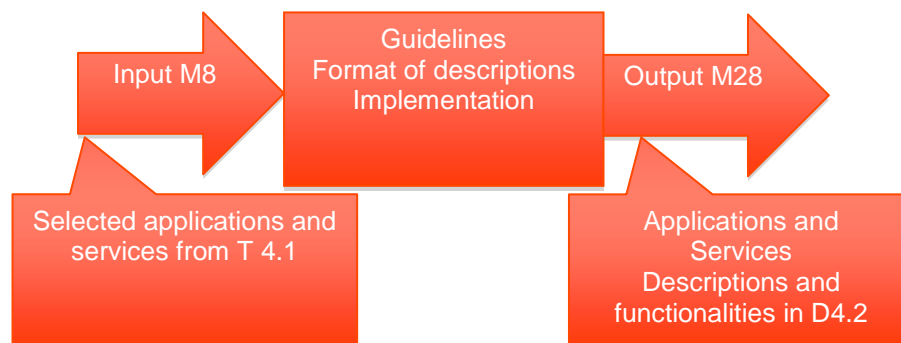


Figure 1: SmartSantander Task 4.2 overview

A ranked list of use cases and its descriptions served as input for task 4.2, where we further analysed and described with in deeper technical details the use cases selected for implementation. These technical specifications provided the level of detail required to support the implementation phase. The outputs of this task were the developed applications and services which are now described in this report deliverable. This deliverable will be used by task 4.3 in the evaluation process of the implemented services and applications, i.e., calculation and matching of KPIs (key performance indicators) described in the D 4.1.

This report is structured as follows:

- Section 2 enumerates the use cases selected for realisation in terms of their corresponding services and the applications that fulfil them.
- Section 3 describes the methodology used to design the different components that support the implemented applications. Basically a user-driven methodology has been followed in order to involve users throughout the development cycle, from a use case's conception through its realisation and validation.



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

- Section 4 provides detailed full description of the different applications which make use of the facilities and services provided by the SmartSantander platform. Those applications have been developed the aforementioned methodology.
- Next, Section 5 proposes a description of the USN component which provides the storage of the information that is sent by the devices connected to SmartSantander facilities and it notifies that information to the related subscribed applications. Also this section explains how the functionalities provided by the SmartSantander platform can assist to developers interested in building new services and/or applications following either a participatory sensing or augmented reality approach.
- In Section 6 we introduce the concept of innovation incubator, which introduces how a smart city should provide mechanisms to integrate and link services and applications beyond mere vertical solutions.
- Finally the section 7 presents the conclusions about this deliverable.

PUBLIC, SMARTSANTANDER PROJECT
2. LIST OF IMPLEMENTED SERVICES AND APPLICATIONS

In this section we present a list of the selected use cases from task 4.1 that were developed in task 4.2, this includes a list of services and applications realised per use case, they are as follows:

| <u>Use Case</u> | <u>Service(s)</u> | <u>Application(s)</u> |
|--|---|---|
| WP4_UC1 – Parking | WP4_Serv8 - Parking Service | WP4_AP1a - Parking iPhone app WP4_AP1b - Parking Android app WP4_AP1c – Parking Windows mobile app WP4_AP1c - GIS system app |
| WP4_UC5 - Environmental Monitoring | WP4_Serv9 - Environmental Monitoring service | WP4_AP2a - Environmental Monitoring iPhone app WP4_AP2b - Environmental Monitoring Android app |
| WP4_UC11 – Participatory Sensing | WP4_Serv10 – Pace Of The City Server WP4_Serv11 - PSens Server WP4_Serv16 – Universal Alert System | WP4_AP3a – Pace Of The City iPhone app WP4_AP3b – Pace of The City Android app |
| WP4_UC3 – Cultural Activities – Augmented Reality | WP4_Serv12 – Augmented Reality Server | WP4_AP4a – Augmented Reality iPhone app WP4_AP4b – Augmented Reality Android app |
| WP4_UC6 – Public Transportation | WP4_Serv13 – Public Transportation Server | WP4_AP5a – Ekobus Android app WP4_AP5b – Ekobus Web App |
| WP4_UC10 – Precision Irrigation Management System | WP4_Serv14 – Irrigation Server | WP4_AP6a – Irrigation Web app WP4_AP6b – Irrigation Android app |
| WP4_UC4 – Smart Metering | WP4_Serv15 – Smart Metering Server | WP4_AP7a – Smart Metering desktop |



PUBLIC, SMARTSANTANDER PROJECT

| | | |
|--|--|--|
| | | app WP4_AP7b – Smart Metering Android app |
|--|--|--|

Table 1: List of Services and Applications

PUBLIC, SMARTSANTANDER PROJECT

3. METHODOLOGY FOR DESCRIPTION OF SERVICES AND APPLICATIONS

The methodology used within each Service/Application specification process in Task 4.2 is hereby described.

The UML [ref. 2] standard specification defines a language for software modelling and can universally be applied in many spheres of software development. However, it can still be adapted for custom environments as it includes mechanisms to accommodate user or environment specific extensions.

The Use Case-Driven nature of modelling with UML ensures that all the different models trace back to elements of the original functional requirements. This traceability threading through the models comes without the extra effort of creating and maintaining a 'traceability matrix' which can be complex and as well as time consuming.

The benefit of this approach is that the impact of a requested change can quickly be estimated. A change in requirements can be traced through analysis and design models into the components affected and even up to the lines of code that implement the affected functionality. The affected code can then be traced back to the requirements and total regression testing effort calculated.

In most of the services and applications, we utilised an incremental software engineering process where only those parts of the model that were required to fulfil the functionality expected in the current phase of the project were implemented. In this way, only the work needed to fulfil the current requirement is done and components are designed in such a way as to maximise reuse, maintainability and extensibility.

Moreover, a large complex project, as SmartSantander, take advantage from UML models because in this way the project can be decomposed in a totally user-definable way so that different parts of the model can be developed independently by different people or groups. Furthermore, given the right tools and change management infrastructure, each sub-model can be independently change managed. This facility enables independent, but yet controlled and coordinated, development effort by different groups in the project. Each group, however, still has full visibility of the activity of other groups and is kept fully aware of all changes that affect its own work.

1. We have utilised a software development process that is underpinned by Object Oriented Design principles and the UML specification language. Within this scheme of things. the next section presents in such a way as to reflect the different stages in our OO Design process: A description of the use case considered by each services has been provided;



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

2. Starting from the use case description a set of “UML Use Case Diagram” have been derived in order to describe the functions and the services provided by the system and their interaction with the actors involved in the services utilizations (such as service providers, users and potential stakeholders);
3. From these UCD a set of functional and non-functional system requirements have been then identified;
4. Based on the identified requirements a Block Diagram of the implemented service is then proposed;
5. The interaction between different architecture components has been then visualized by means of “UML Sequence Diagram” in order to show how the system processes and components interact with each and in which order and employing which message.

4. IOT SERVICES

In this section we describe the developed applications and services in terms of the software development method outlined in the previous section..

4.1. SMART PARKING SERVICE AND APPLICATIONS

In this scenario the user needs an overview of the realtime occupancy of certain parking areas in the city of Santander; the viewpoints of the different stakeholders of this scenario are as follows:

- Citizens perspective
 - From their Smart Phones, PDAs, PCs the citizens can check the availability of parking spaces in the city (both standard and for people with disabilities).
 - Citizens can also get a route to a specific parking area; so they can easily reach a certain parking area.

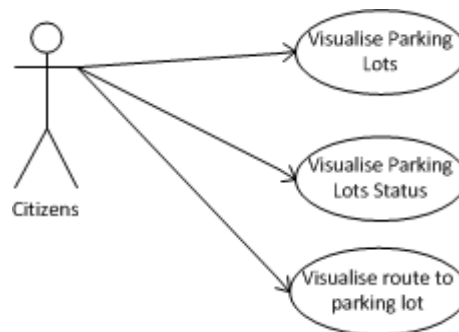


Figure 2: Parking use case diagram – Citizens' perspective

4.1.1. System overview

In order to get access to the parking spaces availability information at real-time we need to connect to the USN component of the SmartSantander platform. This information must be stored locally in a server component that communicates with the applications. The following diagram illustrates in a coarse-grained manner the components deployed in the system.

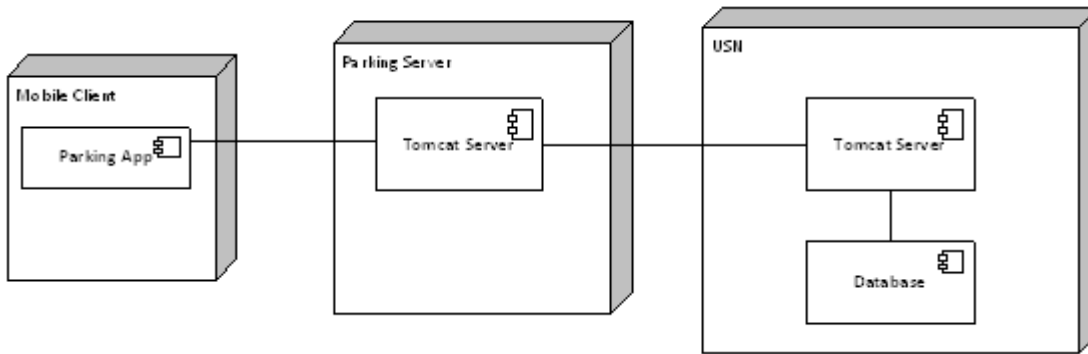


Figure 3: Parking - deployment diagram

The Parking Server is the component that holds the information about the parking availability. It interacts with the USN component through an asynchronous publish-subscribe mechanism; i.e. it subscribes to the USN for events regarding parking sensors. From these subscriptions, it receives notifications from the USN whenever there is a change of availability of parking spaces. On consuming this event, the parking server updates its data by setting the availability of a parking space to free or occupied according to the event received.

The following diagram shows in more details the connection between the Parking Server and the USN component.

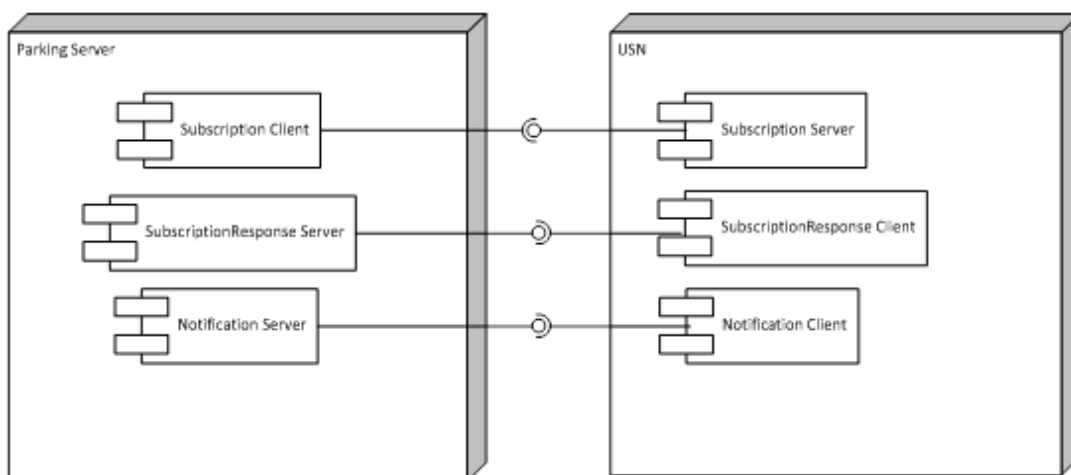


Figure 4: Parking - Publish-Subscribe connection Parking Server and USN

The Parking Server includes a Subscription client component, whereas the USN contains a Subscription server component. The Parking Server uses the subscription interface to subscribe to the USN to events regarding

PUBLIC, SMARTSANTANDER PROJECT

parking sensors. To receive a response from the subscription call the Parking Server contains a SubscriptionResponse server component which the USN calls to send the response of the subscription call. For receiving notifications the Parking Server contains a Notification server component that the USN calls whenever there are changes to parking sensors.

The interaction between USN and the Parking Service is illustrated in the following sequence diagram:

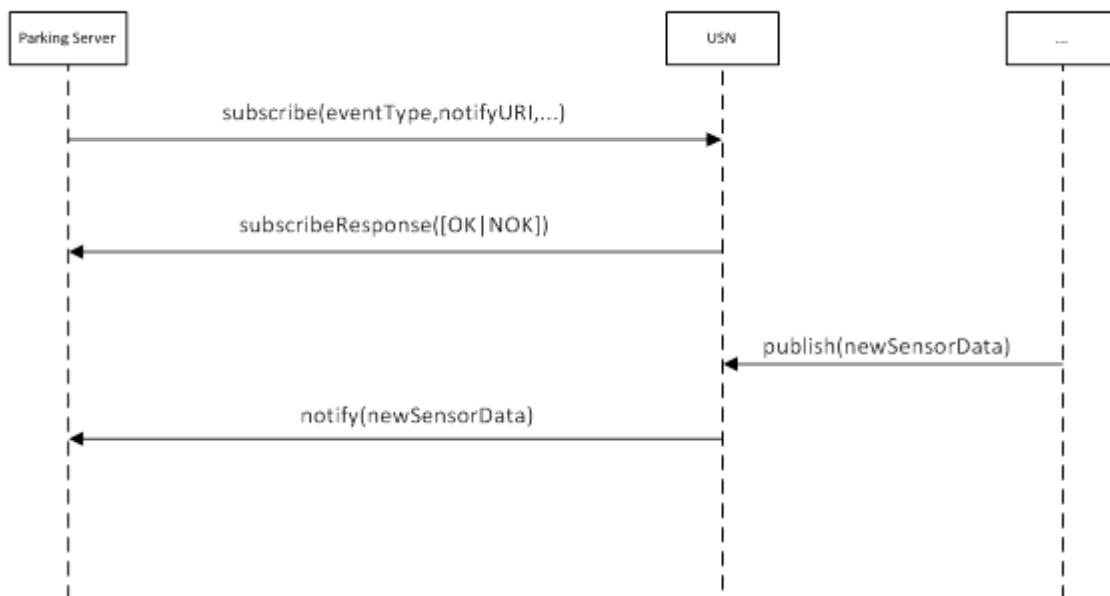


Figure 5: Parking Service - Interaction between service and USN - Sequence Diagram

The connection between the Parking Server and the Mobile Client is done through a Web Service interface the Parking Server provides. By making a SOAP based invocation the mobile client can request the specified parking lots availability/occupancy. The following figure shows the connection between mobile client and Parking Server.

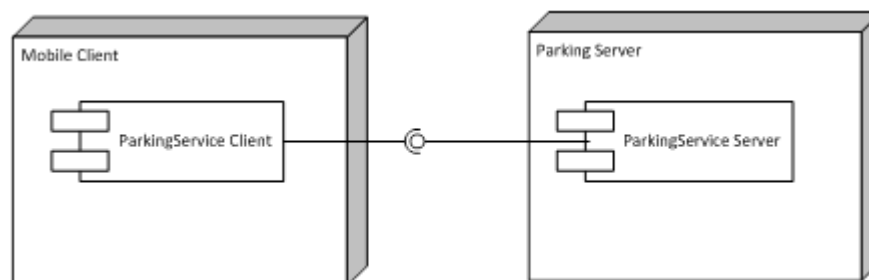


Figure 6: Parking Service - Interaction between server and mobile application - Component Diagram

PUBLIC, SMARTSANTANDER PROJECT

By connecting to the USN platform our service is able to get valuable information about sensor data. The communication between these two components is done using XML Web Services (with a SOAP transport protocol).

The Parking Server provides a very simple Web Service interface, providing a method which the Mobile Client calls in order to obtain information about the parking lots available in Santander, as well as, its occupancy. The Web Service Description and XML Schema files for this service can be found in the Appendix One of this deliverable.

As can be seen in the Parking Service web service description the service provides a simple method `getParkingLots()`, this method returns a sequence of `ParkingLot` data type. A `ParkingLot` data type is defined by its address and a sequence of `ParkingSpace` data type. A `ParkingSpace` data type is defined by a `ParkingSpaceType` (ex: Parking space for people with disabilities, time limited parking space, etc), a `SpaceCurrentStatusType` (ex: Free or Occupied) and `GPSCoordinates` of the `ParkingSpace`. See figure 7.

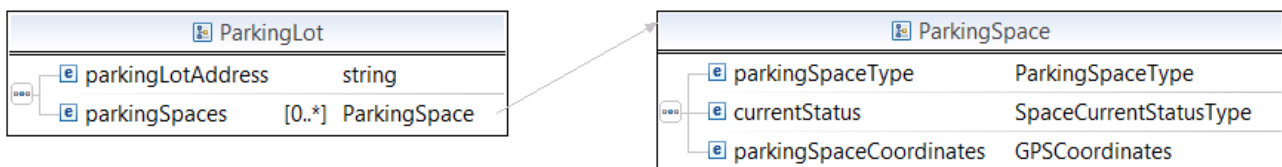


Figure 7: Parking Service - web service data structures

4.1.2. Applications Description

In this section we describe the applications developed for the Parking use case. Three mobile applications have been developed; one for the iOS platform; for the Android platform and finally for the Windows mobile platform. The three applications have similar functionalities and usability.

4.1.2.1. Parking iOS, Android and Windows mobile Application

When the “SmartParking” application starts it sends a `getParkingLots()` request to the server and the Parking Server returns the list of parking lots and its availability. The application displays in a map the information received. In section 6 we describe in detail how to use the applications. The following sequence diagram shows the interaction between application and parking server:

PUBLIC, SMARTSANTANDER PROJECT

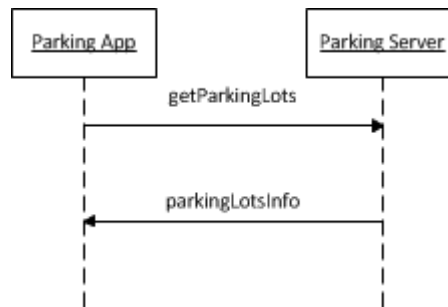


Figure 8: Parking Service – getParkingLots call

When the application starts, it provides the user with a visualisation in a map the parking lots and its locations in the city. Each parking lot is denoted with a P icon. The following figures show the map view:



Figure 9: Map View (Android,iOS and Windows mobile)

The user can also see the parking lots organized in a list ordered by current distance. To view this option in the Android application the user can click on the target button on the top right of the screen; as for the iOS application on the bottom right the target button with “Parking Lots” written. As for the Windows mobile app, on the bottom center the user can click on the button in the middle; the list will then be displayed as follows:

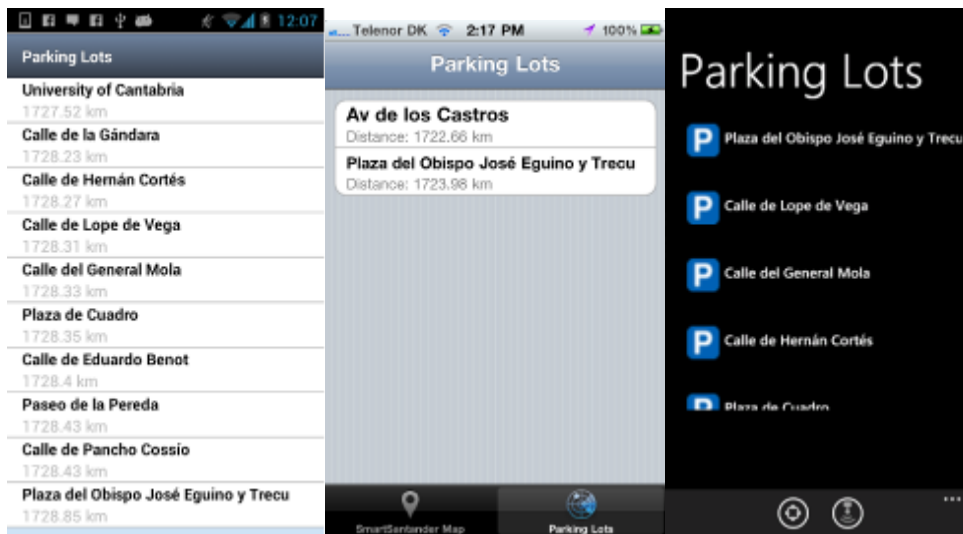


Figure 10: Parking Lots list view (Android, iOS and Windows Mobile)

For visualising the current occupancy status of a certain parking lot the user can click on the parking lot he/she wants. A detail view of the parking lot is loaded as we can see in the following screenshots:

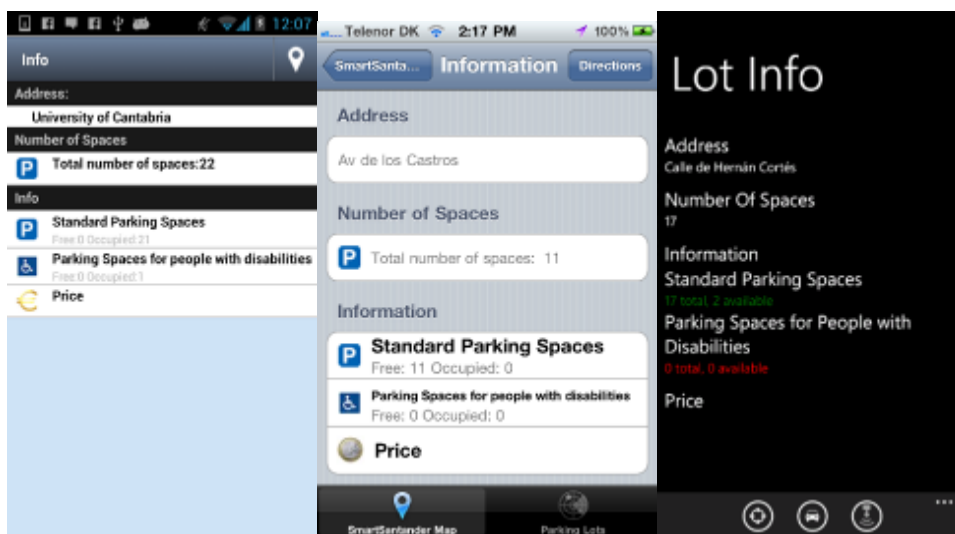


Figure 11: Parking Lot detail (Android, iOS and Windows Mobile)

To get a route from the user's current location to the parking lot the user can click on the button on the top right of the screen. On clicking on this button, the maps application of the phone is loaded with the route(s) to the parking lot; this functionality is very useful for the users that are for instance visiting the city and do not know the roads. The loaded maps are as follows:

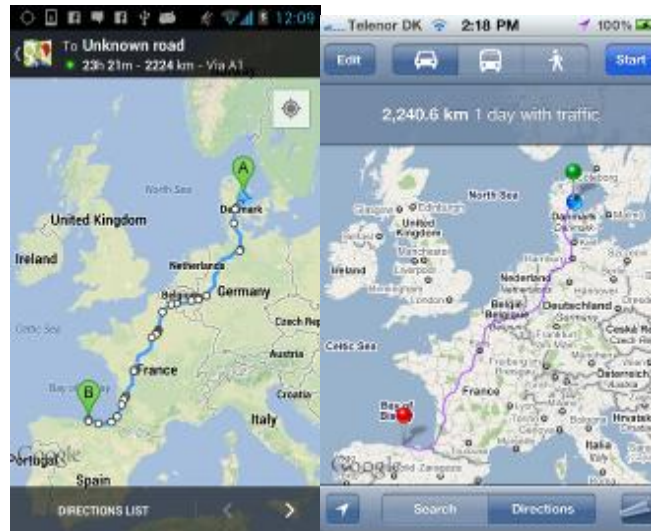


Figure 12: Maps loaded displaying route to the parking lot

Back to the map view of the application the user can also click on the icons of a parking lot this will open an alert displaying the number of parking spaces the lot has. It is seen as follows:

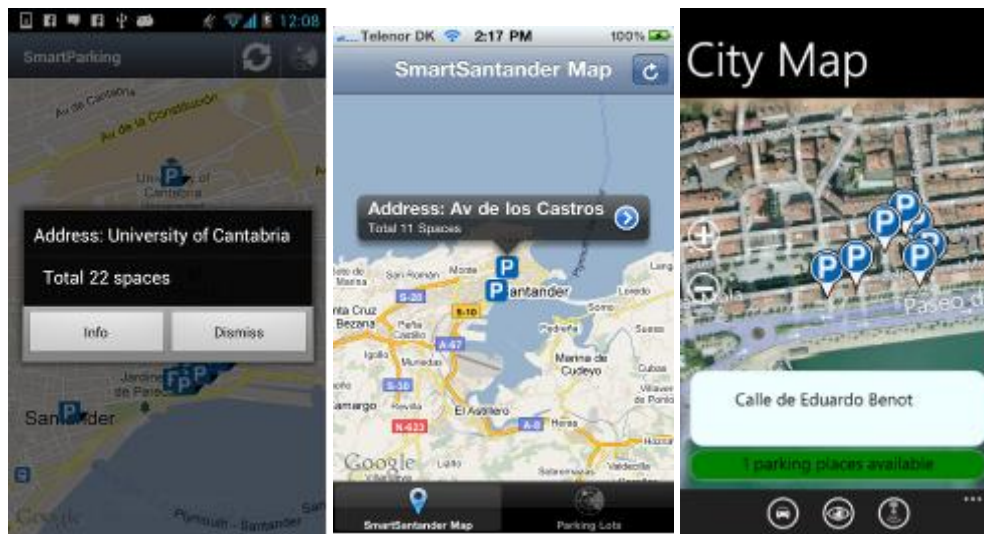


Figure 13: Alert view: Existing number of parking spaces

To visualise the details of the parking lot the user can click on more info; the view loaded is the detail parking lot one mentioned above.

4.1.2.2. GIS application

One of the first developments in this task 4.2 was the integration between a legacy application with the information collected by sensors deployed in SmartSantander, mapping the different devices and their information in the Geographic Information System (GIS) owned by Hall of Santander.

A software component was developed which acts as a proxy between the Santander's GIS and some web services provided by SmartSantander. The software component was developed using Microsoft .NET C# programming language. As this application was conceived as a pilot then the only functionality that has been developed is the mapping of sensors and repeaters. Currently the application is only for internal use for municipal technicians. The GIS is expected to be made public to citizens in the near future because it is a project still under development.



Figure 14: GIS application screenshot

4.2. ENVIRONMENTAL MONITORING

For the environmental monitoring scenario, the user needs an overview of the environmental conditions in Santander such as temperature, CO level, luminosity and noise; the overview of the stakeholders of this scenario is:

- Citizens/experimenter perspective
 - Visualise in a map the latest environmental measurements of a sensor from their mobile phones.



Figure 15: Environmental Monitoring – Use case diagram

4.2.1. System Overview

In order to access environmental data in real-time a connection to the USN component of the SmartSantander platform must be done. This data must be stored locally in a server component that communicates with the applications. The following deployment diagram shows the components that compose the system.

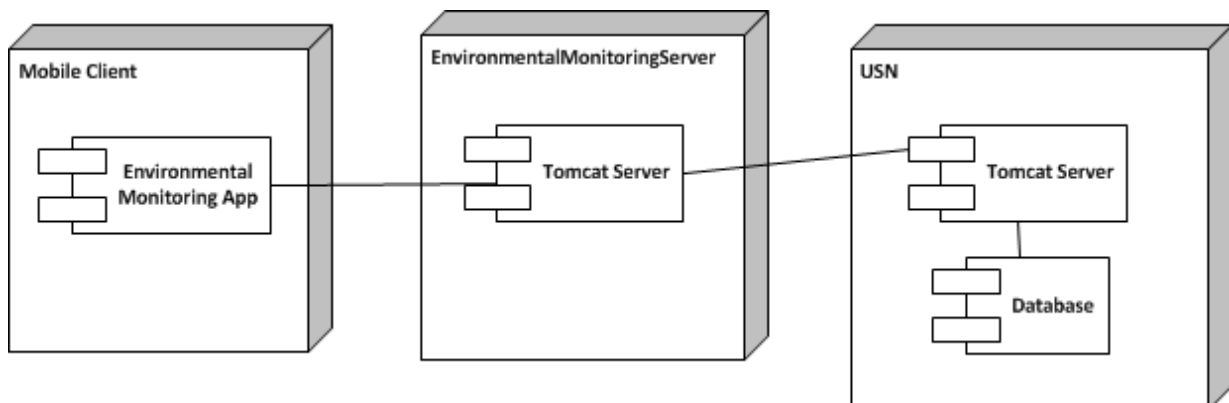


Figure 16: Environmental Monitoring - deployment diagram

The Environmental Monitoring Server is the component that receives and holds the the environmental sensor data. It is connected to the USN component through an asynchronous publish-subscribe mechanism, i.e. it

PUBLIC, SMARTSANTANDER PROJECT

subscribes to the USN for events regarding environmental sensors. The USN notifies the Environmental Monitoring Server whenever there is a new measurement of an environmental sensor, after receiving this event the environmental monitoring server updates its data setting the new measured value for the sensor in cause.

The following diagram shows in detail the connection between the Environmental Server and the USN component.

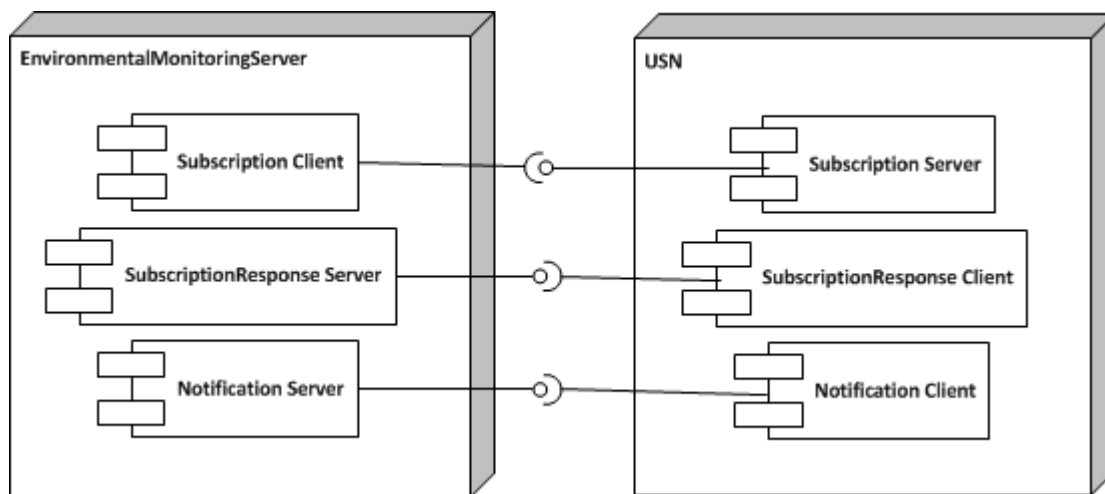


Figure 17: Environmental Monitoring – Publish-Subscribe connection Environmental Monitoring Server and USN

The Environmental Monitoring Server contains a Subscription client component, whereas the USN contains a Subscription server component. The Environmental Server uses the subscription interface to subscribe to the USN to events regarding environmental sensors. To receive a response from the subscription call the Environmental Monitoring Server contains a SubscriptionResponse server component which the USN calls to send the response of the subscription call. For receiving notifications the Environmental Monitoring Server contains a Notification server component that the USN calls whenever there are changes to environmental sensors.

The interaction between USN and the Environmental Monitoring Service is illustrated in the following sequence diagram:

PUBLIC, SMARTSANTANDER PROJECT

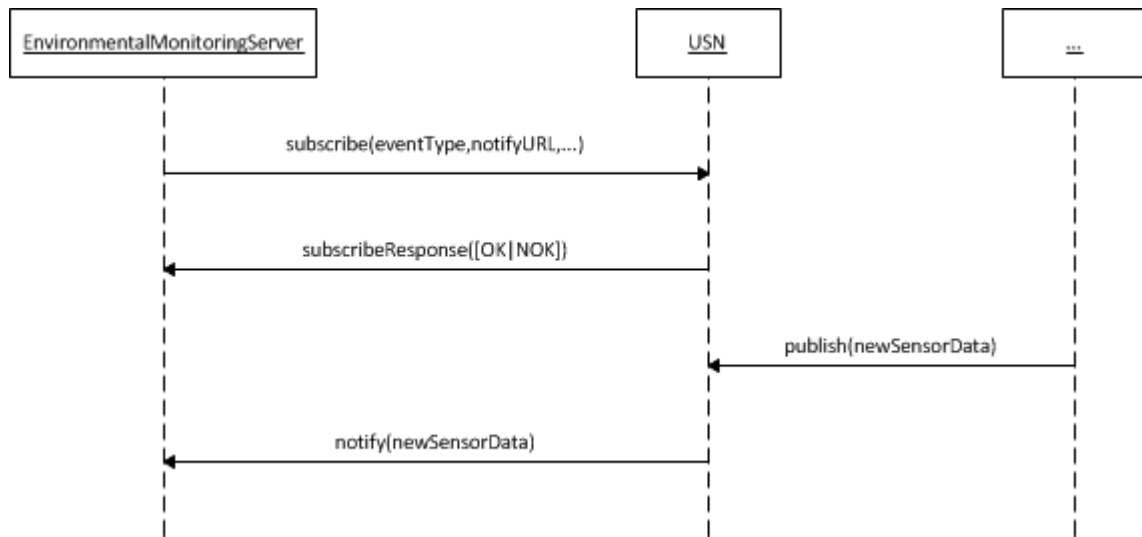


Figure 18: Environmental Service - Interaction between service and USN - Sequence Diagram

The connection between the Environmental Monitoring Server and the Mobile Client is done through a Web Service interface that the Environmental Monitoring Server provides. By making a SOAP based invocation call the mobile client can request the latest environmental sensor measurements. The following figure shows the connection between mobile client and Environmental Monitoring Server.

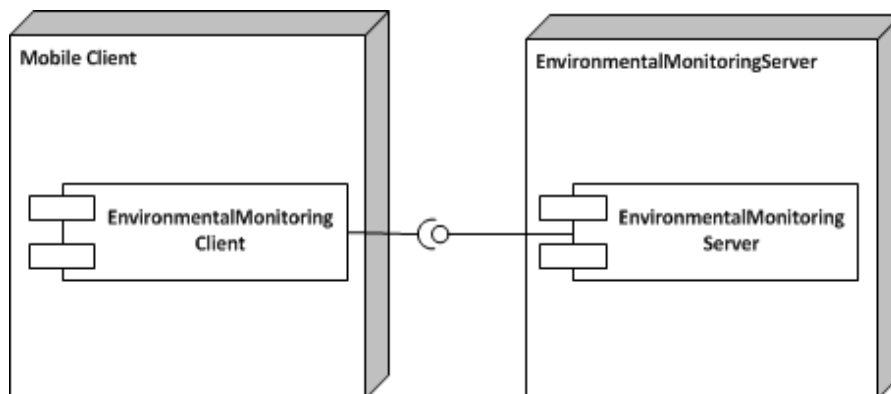


Figure 19: Environmental Monitoring Service - Interaction between server and mobile application - Component Diagram

By connecting to the USN platform our service is able to get valuable information about sensor data. The communication between these two components is done using web services (with SOAP as the transport protocol).

PUBLIC, SMARTSANTANDER PROJECT

The Environmental Monitoring Server provides a very simple Web Service interface it exports a method which the Mobile Client calls in order to obtain the environmental data of Santander. The Web Service description and XML Schema files for this service can be found in the Appendix Two of this deliverable.

As we can see in the Environmental Monitoring Service web service description the service provides a simple method called `getEnvironmentalData()`, this method returns a sequence of `EnvironmentalSensor` data types. An `EnvironmentalSensor` data type, as depicted by Figure 20, is defined by a String `SensorType`, a `CurrentMeasuredValue`, a `CurrentMeasuredValueUnits` and the `GPSCoordinates` of the sensor.

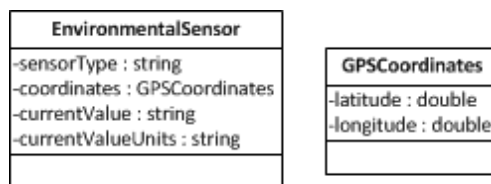


Figure 20: Environmental Monitoring Service - web service data structures

4.2.1. Applications Description

This section describes in detail the applications developed for the Environmental scenario. For this use case we have developed two mobile applications: one for the iOS platform and another for the Android platform. The two applications are similar in term of functionality and usability.

4.2.1.1. Environmental iOS, Android and Windows mobile Application

The environmental mobile applications were developed for the above mentioned platforms. When the application opens, it sends a `getEnvironmentalData()` request to the server and displays the received information in a map. Section 6 provides guidelines on how to use the applications. The interaction between environmental monitoring app and environmental monitoring server can be seen in the following diagram:

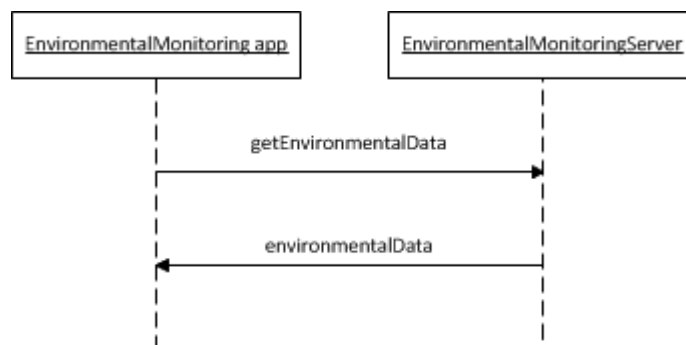


Figure 21: Environmental Monitoring – `getEnvironmentalData` call

PUBLIC, SMARTSANTANDER PROJECT

For the environmental monitoring use case we have developed two simple applications for the Android and iOS platforms. The users can visualise all the environmental sensors in a map, as shown in the following figure:



Figure 22: Environmental Monitoring map (Android and iOS)

As depicted in the following figure, if the user clicks on an icon of an environmental sensor he/she can visualise the last reading of this sensor.

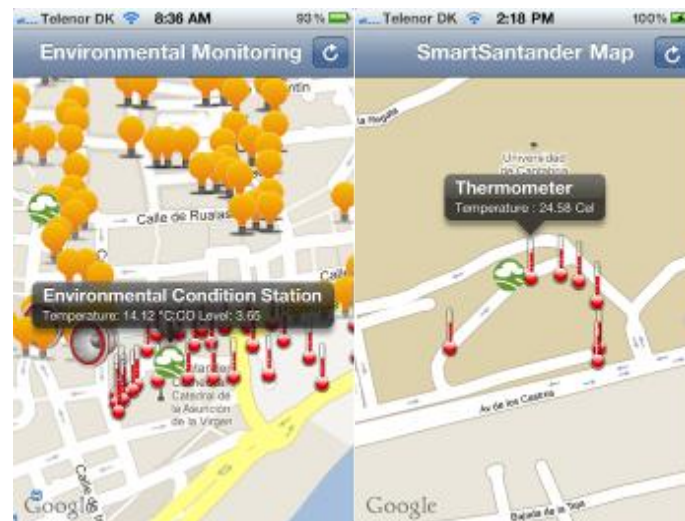


Figure 23: Environmental Monitoring measured values (Android and iOS)

PUBLIC, SMARTSANTANDER PROJECT

4.3. PARTICIPATORY SENSING

In this scenario, users utilize their mobile phones to send physical sensing information, e.g. GPS coordinates, compass, environmental data such as noise, temperature, etc. This information is fed to the SmartSantander platform. Users can also subscribe to services such as “the pace of the city”, where they can get alerts for specific types of events currently occurring in the city. Users can also themselves report the occurrence of such events, which will subsequently be propagated to other users who have subscribed to the respective type of events, etc.

The users receive the notifications on the occurred events via a smartphone application, SMS and e-mails in the preferred language.

All users interested in receiving the notifications have to register with the service, thereby their personal profile (including e.g. the preferred language) and selecting the information they are interested to. This subscription can be done via web interface. If the Council wants to provide the service also to users without web access, it can provide a phone number of a help desk to be called in order to subscribe to the service with operator support.

Examples:

- Maria has finished working for the day and is going home by bus. She catches the bus in front of her workplace, while the trip normally takes around 20 minutes. Unfortunately today there is a lot of traffic for unknown reasons. After 10 minutes stuck in traffic, she finds that the cause for the traffic congestion is a road accident. She uses her smartphone to report the road accident as well as its location. Her colleague Carla that uses also this application receives an alert in her smartphone that there is an accident in that area and decides then to take another route home.
- Antonio and his wife Carmen don't have a smartphone to use the application, so they register themselves with the notification service using their home PC to receive information on traffic situation in the area they daily pass through. When Maria reports the accident, Antonio is at his office and receives the information on his mobile with a call or an SMS. So Antonio comes back home taking another route. At the same time, Carmen stays at home and receives a call on fixed phone just a few minutes before her usual turn to go to the gym, so she decides to use the next turn instead.
- José is a municipal guard in Santander. At the beginning of his shift time, he subscribes to the notification service indicating the area where he works. Ana is subscribed to PSens service and is driving her car near a school. She observes that in front of the school, there is a tree that is falling down due to a windstorm, so there is a serious danger for the students that will go out from the school



PUBLIC, SMARTSANTANDER PROJECT

in a short time. She uses her smartphone to report the need of an urgent intervention. Since the school is in José's working area, he receives the notification. He immediately goes to the school and blocks the passage under the tree just before the exit of the students. Afterwards, he calls firemen that arrive and remove the tree.

- Luis is close to his home. He is walking on the sidewalk and sees that there is a big hole in the middle of the street; this is maybe due to the heavy rain in the last months. Using his smartphone he reports the anomaly giving the details and sending a picture. The city council then receives an alert about this anomaly and decides to take a quick action by sending someone responsible to fix this as soon as possible, because this street has heavy traffic.
- Pedro is a sports fan. He is on vacation for the first time in Santander and not sure what to do this afternoon so he decides to check what sports events are in the city during the afternoon. Using his mobile phone he can see that there will be a football match between Santander and Barcelona at 18:00 and since he does not know where the stadium is located, the phone assists him on getting there by displaying a route from the place he is located currently to the stadium.
- Jorge is an active member of environmental groups and a cyclist as well; he decides to take part in SmartSantander activities and help in reporting environmental data with the help of his smartphone. He mounts his smartphone on his bicycle and uses it to record noise levels, road and traffic condition data through a SmartSantander application and then uploads it to the system. Similarly, other members of the cyclist community upload their own data to the system and help to build a map of the noisiest places in the urban area.
- Irene frequently passes through the areas where part of the SmartSantander stationary infrastructure is situated. She is aware of the project goals and decides to help out; she uses her smartphone to communicate with the infrastructure and offload the data recorded in her device.

The following use case diagram shows the functionalities or system services the user can interact with:

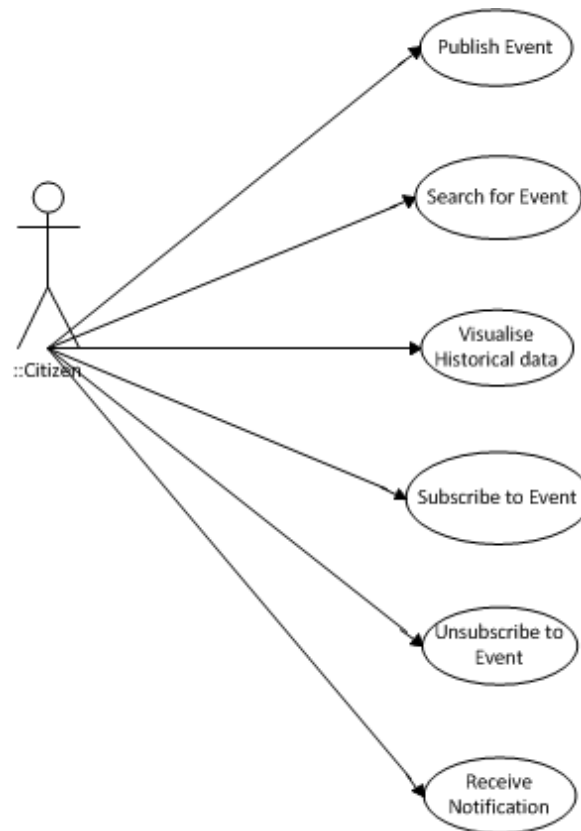


Figure 24: Participatory Sensing – Citizen Use Case Diagram

4.3.1. System Overview

For this scenario three software components were developed: 1) a mobile client application for the users; 2) a server that runs the service business logic and bridges the application with the SmartSantander platform which we call Pace Of The City Server; and 3) a server component that allows the mobile devices to register themselves to the SmartSantander platform called PSens Server. The UAS System was also integrated with the system in order to trigger the user’s notifications via SMS and/or e-mail.

The following diagram shows the various components involved in this scenario and the interactions between them:

PUBLIC, SMARTSANTANDER PROJECT

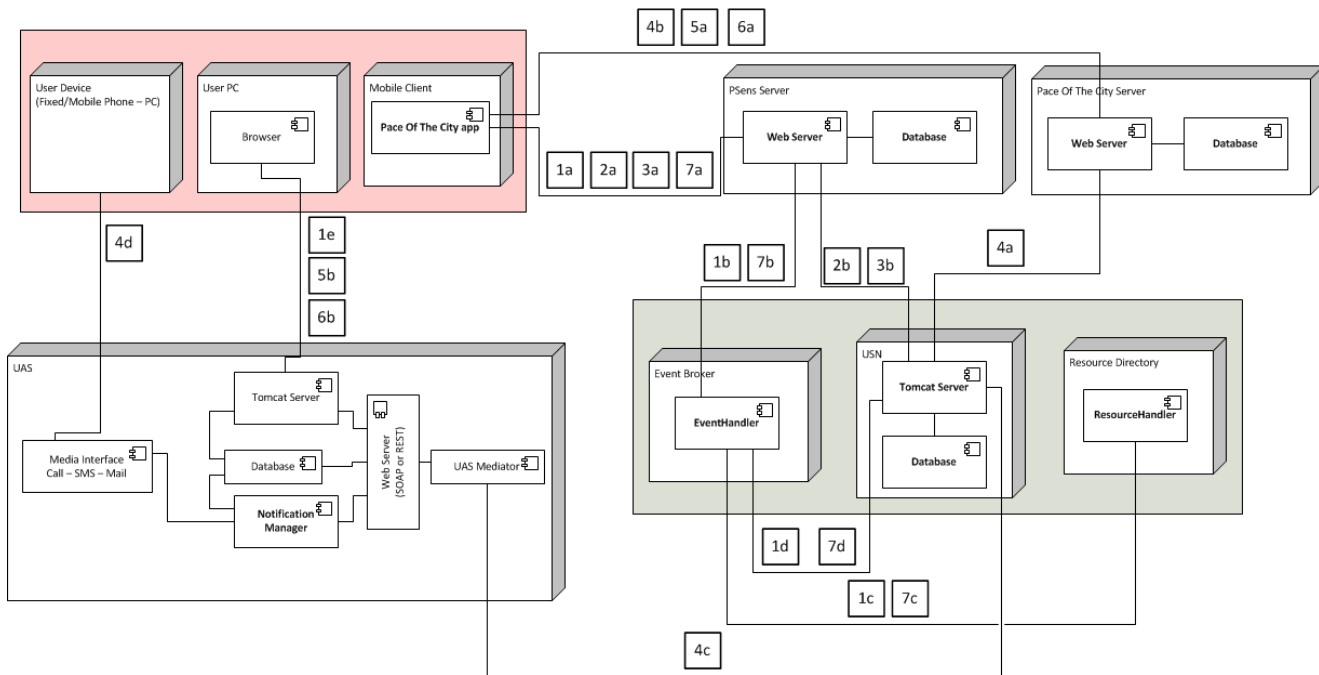


Figure 25: Participatory Sensing scenario - Deployment diagram

We hereby describe the functionality required for this scenario and the interactions between the components to realize this functionality.

1. Register Device

- a. Devices get registered in the SmartSantander platform through the PSens Server. This registration is anonymous. From the mobile client information like phone model and manufacturer is sent to the PSens Server which stores this information locally and generates a unique identifier for the mobile client. This UUID will be used by the mobile client in all the interactions with the PSens Server. If the registration of the device to the SmartSantander platform is successful then the PSens Server returns the generated UUID to the Mobile Client.
- b. A new device must be registered in the SmartSantander platform. Therefore the PSens Server sends a registration message containing device information to the Event Broker component.
- c. The Event Broker component sends the registration information to the Resource Directory so the device can be registered in the SmartSantander platform. The Resource Directory returns a device Id.
- d. The Event Broker also sends a registration message to the USN containing the device information and the device Id.
- e. Users who do not have a smartphone can register themselves to the UAS from their browser using their computer.

The following diagram shows the interaction between the components for a device creation:

PUBLIC, SMARTSANTANDER PROJECT

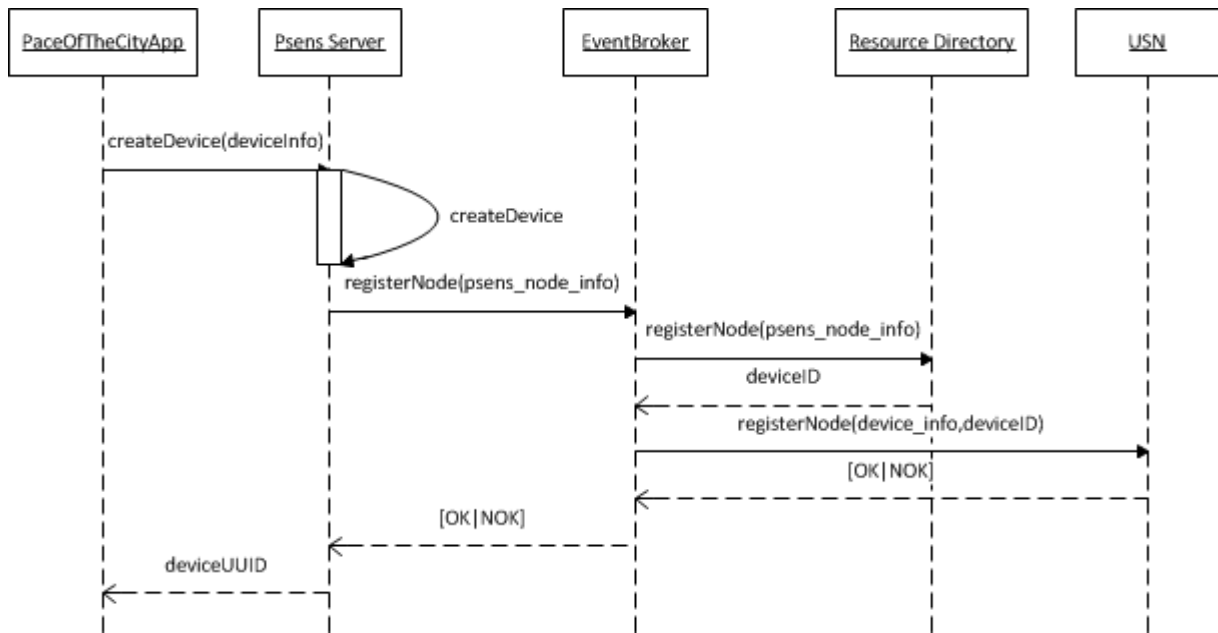


Figure 26: Sequence Diagram: Create Device

2. Physical Sensing

- a. Periodically the mobile client communicates with the PSens server to send information about physical sensing. This physical sensing includes, e.g GPS, compass, humidity, etc. This physical sensing information is forwarded from the PSens server to the USN. The message sent to the USN is an “Observation Message” containing the measured values.

The following sequence diagram shows the interaction between the components involved:

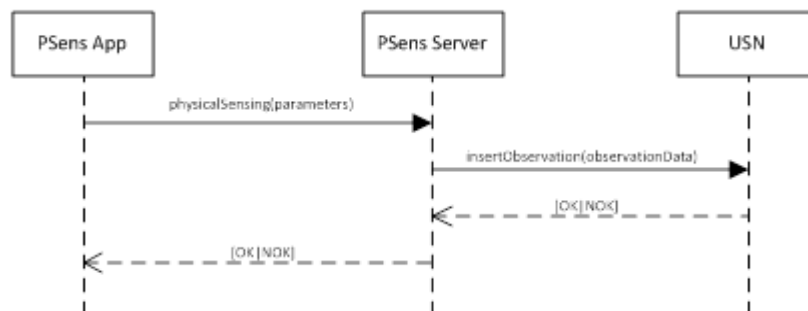


Figure 27: Sequence Diagram: Physical Sensing

3. Publishing an Event observation

PUBLIC, SMARTSANTANDER PROJECT

- a. When the user publishes an event the information about the event data is sent from the PSens App to the PSens server.
- b. The publication data is forwarded from the PSens server to the USN for storing it.

The following sequence diagram shows the interaction between the components when the user publishes a new event:

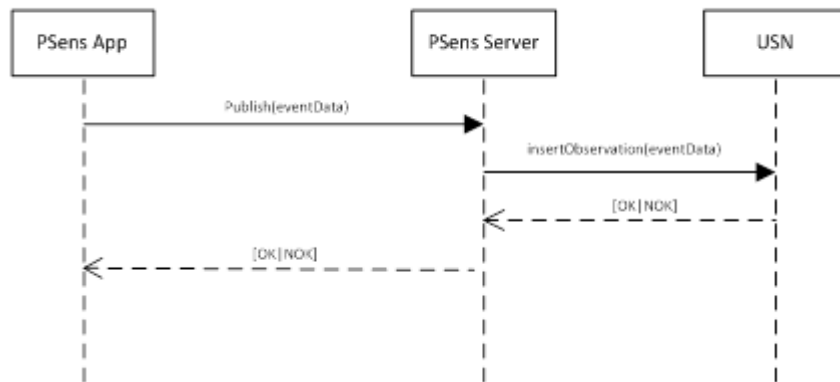


Figure 28: Sequence Diagram: Event Publication

4. Notify

- a. After receiving an event the USN notifies the Pace of the City Server.
- b. The Pace of the City Server notifies the users who are subscribed to that type of event sending the event data.
- c. The UAS is also subscribed to publication of events of this type so it gets notified by the USN; the UAS mediator launches the campaign of messages on UAS, in order to send messages (via e-mail and/or SMS) to all users subscribed on that Topic.
- d. The UAS sends the event information to the subscribed users via phone sms or e-mail.

The following sequence diagram shows the interaction between the components:

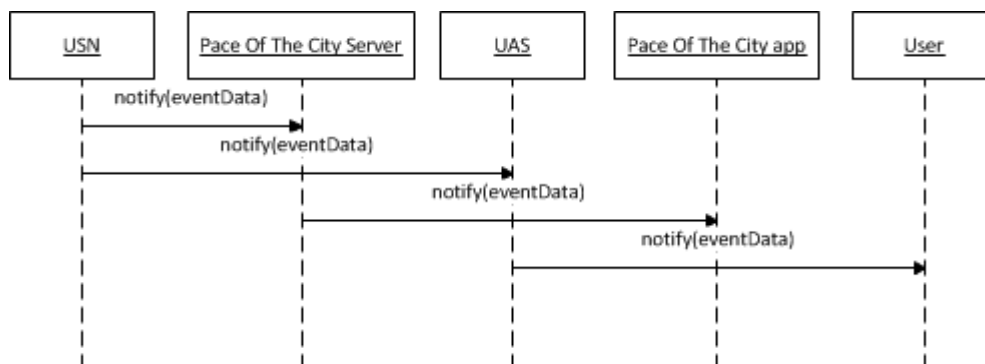


Figure 29: Sequence Diagram: Notification

5. Subscribe

- a. Users can subscribe to new types of events by using the PSens app. The subscription information is sent to the Pace of the City Server.

The following sequence diagram shows the interaction between the components when the user subscribes to a new type of event:

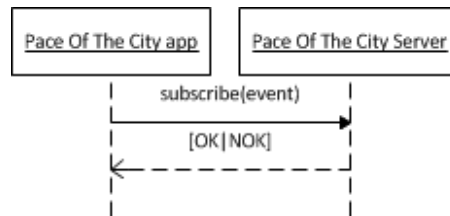


Figure 30: Sequence Diagram: Subscribe

In order to receive the notifications of the Topics they are interested in, the end users (e.g. citizens, authorities) who do not have a smartphone have to subscribe on UAS, creating their profile with their info (phone numbers, e-mail addresses, subscribed topics, etc.). The following diagram shows the user subscribing to UAS:

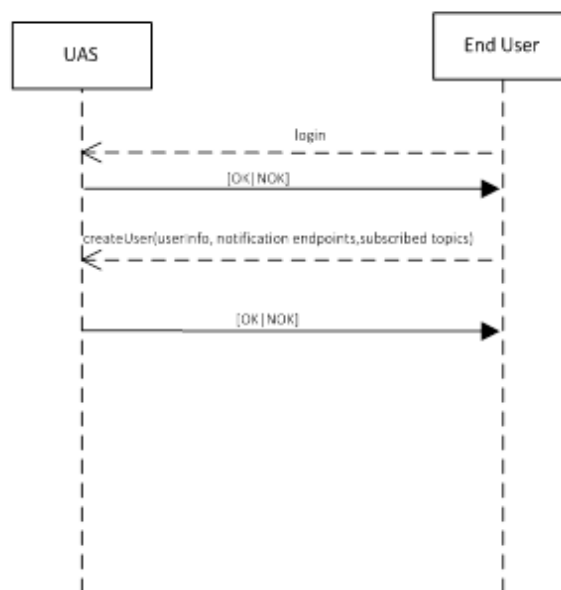


Figure 31: Sequence Diagram: User Subscription on UAS

6. Users Unsubscribing from Events

- a. Users can unsubscribe to events from the PSens app. The unsubscribe message is sent to the Pace of the City Server.

The following sequence diagram shows the interaction between the components:

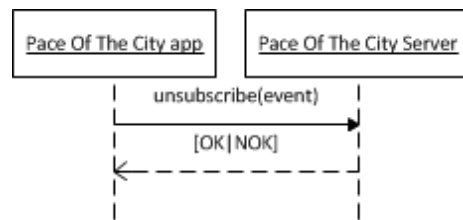


Figure 32: Sequence Diagram: Unsubscribe

The end users without smart phone, who do not want to receive notifications any more, have to unsubscribe on UAS, specifying the topics they are not interested any more. The following diagram shows the component interactions required to revoke subscription at the UAS:

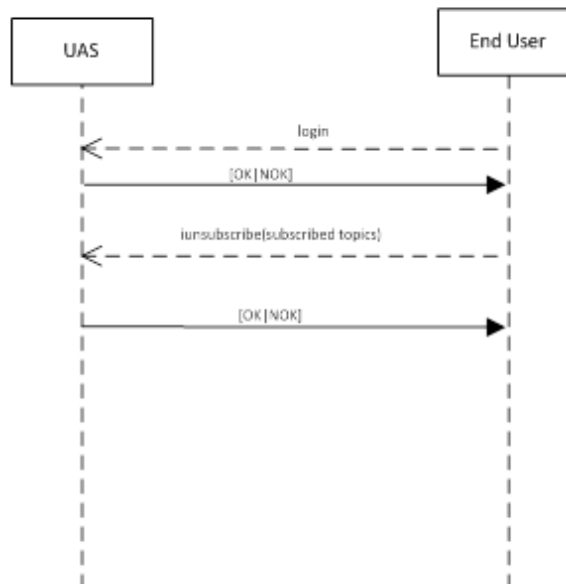


Figure 33 Sequence Diagram: End User revokes the subscription

7. Delete Device

PUBLIC, SMARTSANTANDER PROJECT

- a. From the PSens app the device can be deleted via an unregisterDevice request that is sent to the PSens server.
- b. The PSens server sends a unregister device message with the device id to the Event Broker.
- c. The Event Broker calls the unregister method in the resource directory.
- d. The Event Broker sends a unregister message to the USN.

The following sequence diagram shows us the interaction between the components:

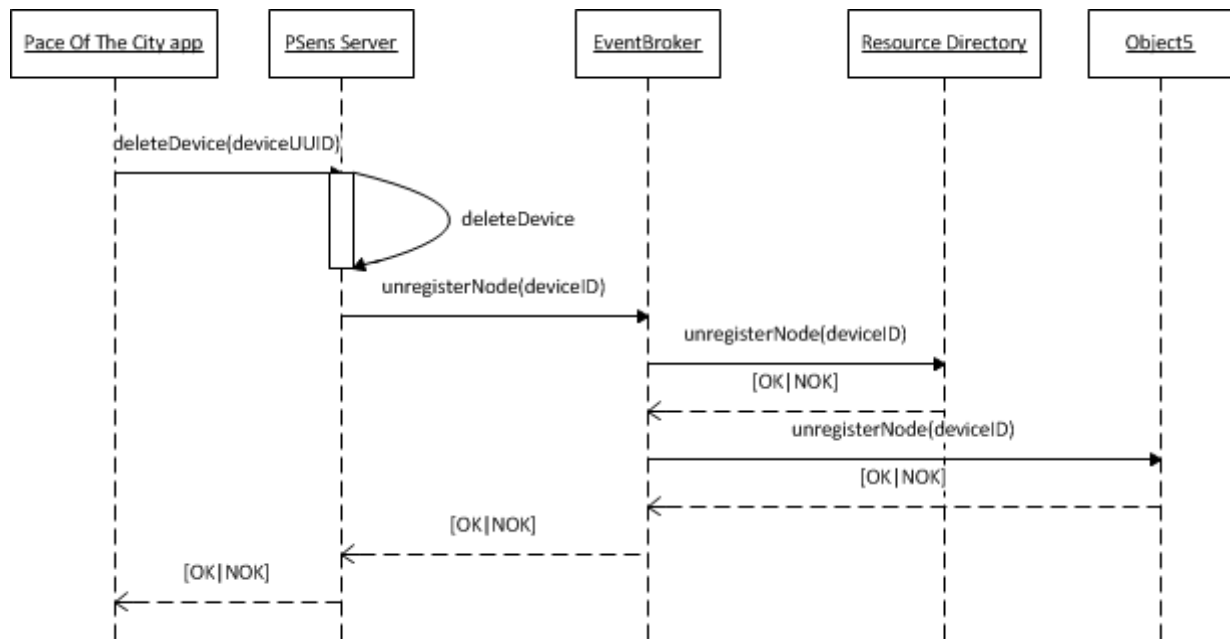


Figure 34: Sequence Diagram: Delete Device

4.3.2. Applications Description

For this use case we have developed two mobile applications; one for the iOS platform and another for the Android platform. The two applications are similar in terms of functionalities and usability.

4.3.2.1.1. Pace of The City iOS and Android mobile Application

For the Pace of The City scenario, we have developed two applications with similar functionalities for Android and iOS platforms. The application provides the following functionalities to the users:

- Visualisation of Historical information in a map or in a list (physical sensing information);
- Subscribe/Unsubscribe to specific types of events occurring in the city;
- Getting Notified of the occurrence of an event of a type the users are subscribed to;
- Search for Events filtering by date, type or location;
- Publish Events;

PUBLIC, SMARTSANTANDER PROJECT

When the application starts the user can visualise in a map the current valid “Pace Of The City” events and its location. The following figure shows this view:



Figure 35: Pace of the City app: Main view

1. In order to publish a new “Pace Of The City” event the user can click on the button on the bottom left corner of the screen “Add Event”. A new view will be loaded where the user can add the details of the new event to be published. The view is as follows:



Figure 36: Pace of the City app: Add Event view

- For searching and visualising specific events the user can click on the “Events” button of the main view of the application. A popup will appear asking the user if he/wants to visualise the events in a list or in a map. If the user clicks on the map option the following view will appear:



Figure 37: Pace of the City application: Events map view

In the Events map view, the user has a button on the top right corner named “Filter”, by clicking on this button several filter possibilities can be tapped by the user; by date, by type, by location.



Figure 38: Pace of the City app: Events filtering view

3. From the main view the user can also visualise his/her measurements historical; for that he/she can click on the “Measurements” button; again a popup will appear asking if the visualisation will be on a map or on a list. On the top right corner a “Filter” button will appear, this will allow the user to filter by specifying the period that the measurements occurred.
4. In order to visualise the events published by the user, he/she can click on the “My events” on the main view of the application. Again the option of visualising the data in a map or in a list will appear.
5. To visualise the “Alerts” the user can click on the “Alerts” button on the bottom right corner of the main view. The alerts can also be visualised in a map or a list. On the right corner of the alerts view a “Subscribe” button will appear. By clicking on this button the user can subscribe/unsubscribe to topics of a specific title. The view looks like the following figure:

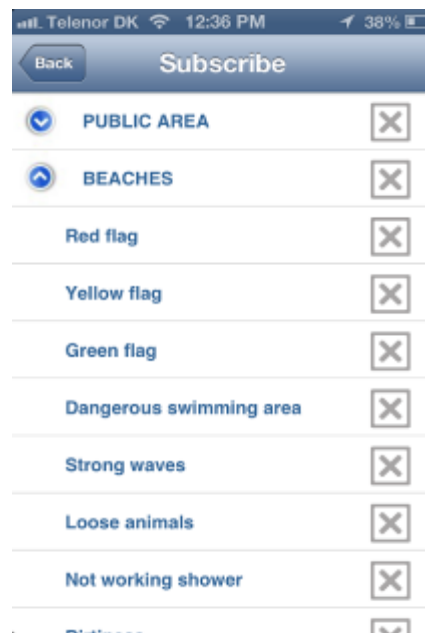


Figure 39: Pace of the City app: Subscriptions view

The user can click on the fold/unfold button on the left side, this will allow to see the titles of the specific topic. For subscribing the user can click on the right side X and the subscription will be made. This means that events occurring in that specific topic will be received by the application as alerts.

PUBLIC, SMARTSANTANDER PROJECT

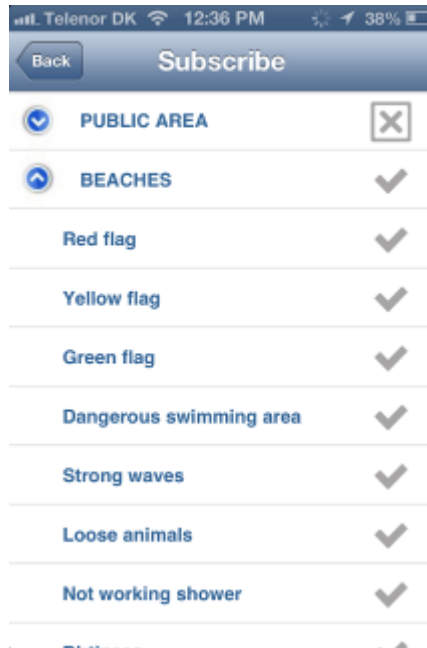


Figure 40: Pace of the City app: Subscription view subscribe

4.3.2.2. Pace of the city: Integration with Incidences management Service in the Santander City Council

In order to tackle internally the incidences sent by the users through the Participatory Sensing scenario (for instance hole in the street events), the Santander City Council implemented two Web services developed using the Microsoft .NET 4.0 platform which integrates the Participatory Sensing Scenario with the pre-existing system in the municipality to manage incidences (INCISYS). These web services collect the information from SmartSantander Participatory Sensing Scenario, then format, adapt and resend that data to the different departments responsible for resolving the incidence provided by the final user.

For this development it was honoured all the standards proposed by the W3C to ensure compatibility with any other platform interacting with the Santander Municipality's systems thus ensuring the interoperability with any other system. The standard adopted for this development has been "SOAP WS-I Basic Profile" (<http://www.ws-i.org/Profiles/BasicProfile-1.1.html>).

4.3.3. UNIVERSAL ALERT SYSTEM (UAS) SERVICE

In the Participatory Sensing scenario, the notification services are provided by Universal Alert System (UAS).

In particular the UAS will send information published via PSens App to all subscribed users (e.g. citizens, tourists, authorities).

4.3.3.1. UAS functional diagram

In order to introduce UAS component, the next figure depicts its standard functional diagram, showing the communication media that potentially could be used for alert/message broadcasting.

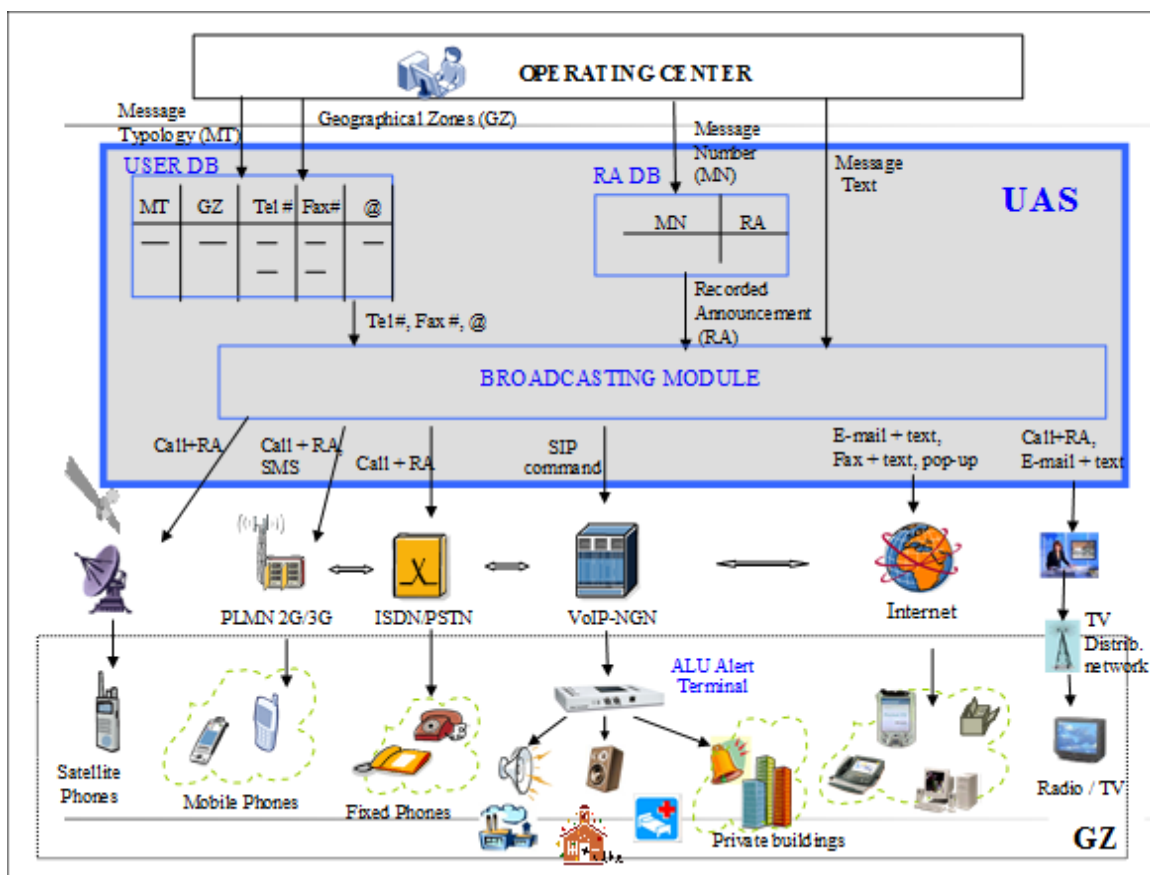


Figure 41: UAS functional diagram

The Operating Center has to provide as input to UAS the Message Typology (MT), the Message Number (MN), the Message Text and the Geographical Zones (GZ) to be addressed by the message.

PUBLIC, SMARTSANTANDER PROJECT

In a nutshell, the UAS performs the following functions:

- gets all the information associated to the MT and GZ's contained in the User DB ;
- provides the Recorded Announcement (RA) corresponding to the MN and/or with the Message Text;
- transfers the above mentioned information, the RA and the Message Text, to its Broadcasting Module, that disseminates the message to the communication media, in the forms allowed by the specific media.

In SmartSantander project, the inputs are not provided by an operating center but they are the notifications sent by USN.

So in order to interface UAS with USN, a specific component, named "UAS Mediator", has been developed.

4.3.3.2. UAS mediator

The UAS Mediator is composed of a JAR package. It is a JAVA application that has:

1. a software component to be deployed by Tomcat Axis on an application server, in order to expose the service to be invoked from USN
2. some classes to be invoked by shell scripts in order to invoke USN web services for subscription of UAS interface

UAS Mediator software is composed by the following main classes:

1. `com.alu.mediator.main.MediatorMainClass`: this class has a main function, this means the class can be invoked by shell command.

The responsibilities of the class are the following:

- a. to call UAS in order to receive the list of events by logging on UAS using a configurable account
- b. for each event to set parameters to correctly call USN web service
- c. to call USN web service subscription
- d. to set and get properties from a properties file

The class can be called using 2 other different ways, in order to unsubscribe services previously subscribed and to disconnect all the services.

The main function tries to subscribe to USN with UAS related events by default; but if the main class is called using unsubscribe or disconnect parameters, the other method is invoked in order to unsubscribe to USN and / or disconnect previously subscribed services.

**PUBLIC, SMARTSANTANDER PROJECT**

2. `om.telefonica.www.wsdl.unica.soap.m2m.notification.v1.services.M2MNotificationServiceSkeleton`: this class exposes the web service to be called by USN in order to notify the event (corresponding to the “topic” of Participatory Sensing service) that has been subscribed by the main class. This class is executed under tomcat/axis environment, then this can be executed on an application server. Main responsibilities of this class are the following:
 - to expose the notify service to USN
 - to parse the message and validate the event topic
 - to call UAS in order to set and launch the campaign of messages, in order to notify the subscribed users via e-mail and/or SMS.

4.3.3.3. User Interaction with the Universal Alert System

The end user has to interact with UAS for the creation of his profile and for subscribing to the topics he is interested to.

For these operations UAS provides a specific web interface: the sub-sections herebelow outline how the user can perform the above operations via UAS web interface.

4.3.3.3.1. Access to the system/Login

In order to access to UAS, the user has to perform the following steps.

1. To launch Mozilla Firefox web browser on a PC connected to Internet.
2. Select the URL: `umb-demo.alcatel-lucent.com/umb`
The page in the next figure will be displayed.
3. Select the favourite language clicking on the relevant flag on the top right
4. Enter own Login / UnId (=Universal Identity) and password and click on “Login” button (see next figure).

NOTE: In the first phase of Smart Santander trial, with a few users, the user accounts (Login) are created by Administrator and communicated to the users.



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

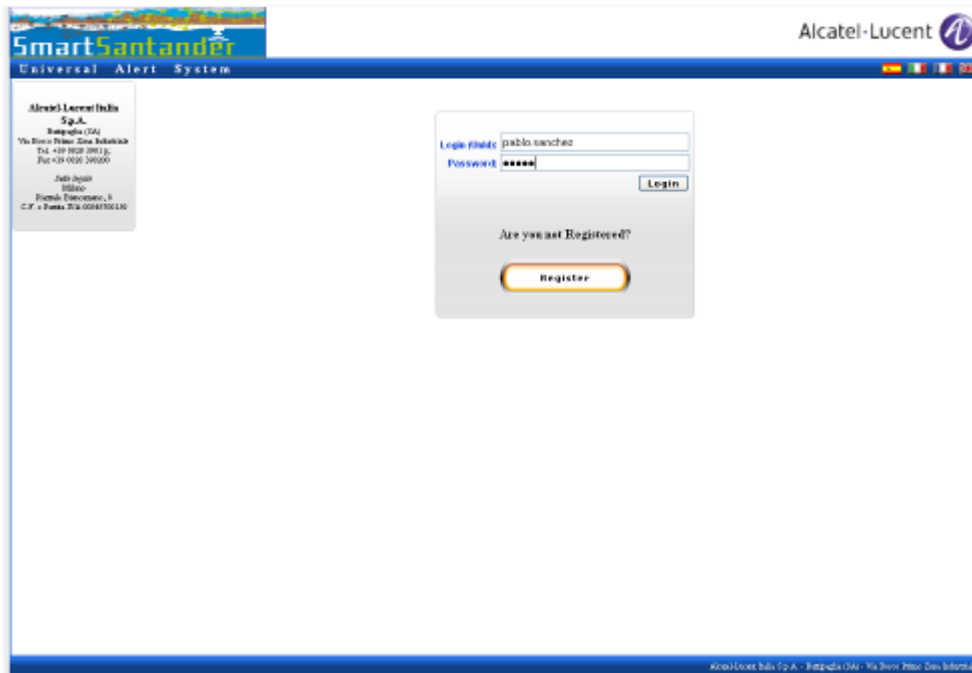


Figure 42 Access to the system

4.3.3.3.2. User menu

After successful login, the user menu is displayed by the system, in order to allow the user to manage his data (see next fig.).



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT



Figure 43 User Menu

The user can select on the left one of the submenu (described in the following paragraphs) in order to insert, display and modify his data.

Only after the insertion of the data requested in each submenu (user details, user references and subscribed events) the user will be able to receive messages.

4.3.3.3.3. User Details

User Details submenu (see next figure) allows the user to insert his personal data, including his PoI (= Point of Interest), i.e. the Geographical Coordinates of the place which the user wants to be informed for (e.g. his home, his office, his children's school), that will be used by the system to discriminate if the user has to be addressed by a geo-targeted message or not.



PUBLIC, SMARTSANTANDER PROJECT

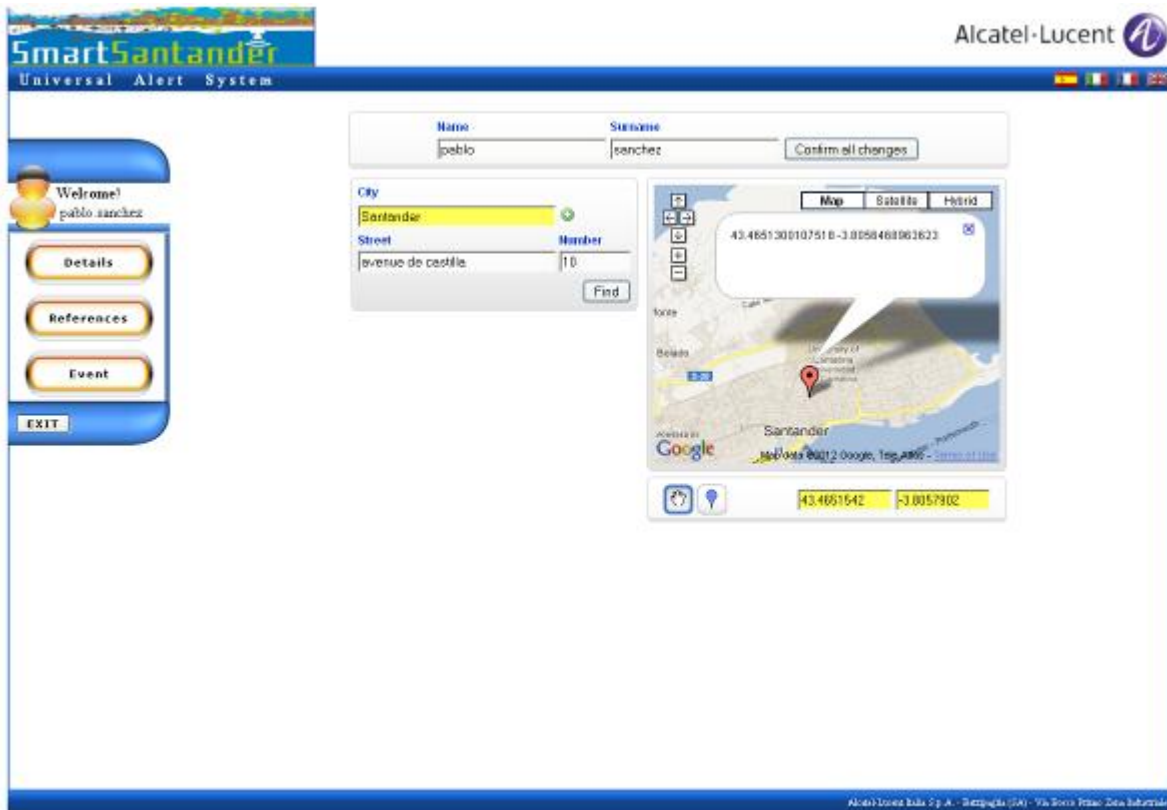


Figure 44 Personal data

The steps that have to be executed are:

- 1) Fill Name and Surname fields.
- 2) Fill City (his PoI)field: the user has to click on “+” button near the “City” field and select the City of the PoI in the menu.

NOTE: in the first phase of Smart Santander trial, only “Santander” can be used in “City” field.
- 3) The user has to enter Street and number of his PoI and click on “Find” button:

On the map on the right, the GIS used by the system (in this example Google map) will display the marker on the position associated by the GIS to the City, Street and number inserted by the user.
- 4) If the marker reflects the exact position of user’s PoI, the user has to click on “Confirm all changes” button

If the marker reflects the exact position of user’s PoI, the user has to click on “Confirm all changes” button.



If the marker does not reflect the exact position, the user has to click on the icon of the marker in the bottom and place the marker (clicking on the map) in the exact position. In order to find the right position, the user can use the features offered by GIS:

- arrows to move the map
- buttons “+” and “-“ to increase or decrease the zoom
- buttons “map”, “satellite”, “hybrid” to change the view

At the end the user has to click on “Confirm all changes” button.

The provided position, expressed in Latitude and Longitude coordinates, will be automatically taken by the system as POI associated to the used UnId.

4.3.3.3.4. References

References submenu (see next figure) allows the user to insert the information about his communication devices (phone numbers, e-mail address) and the relevant priority (the media with highest priority will be the first one used by UAS to contact the end user).

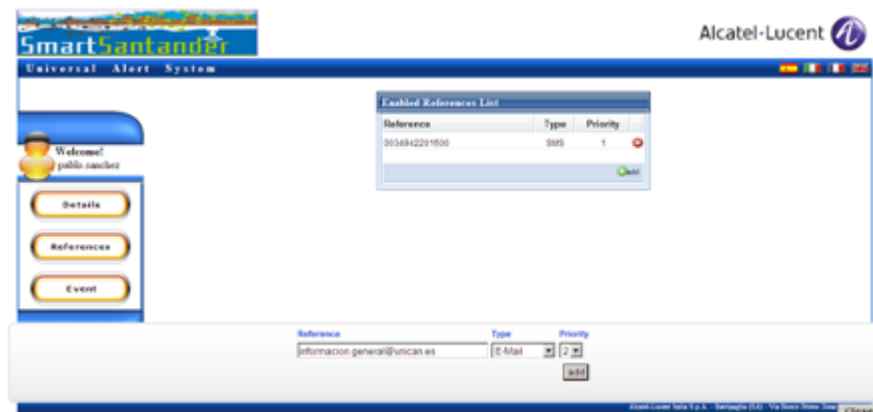


Figure 45 User References

The “Enabled References List” contains all the devices already inserted by the user.

In order to insert/modify the reference of an additional device, the user has to click on “add” button: a window with 3 fields will appear in the bottom.

These fields allow the user to insert for each device 3 data:




PUBLIC, SMARTSANTANDER PROJECT

- Reference (Phone number or e-mail address, depending on Device type)
- Device type (in the first phase: E-mail or SMS)
- Device priority (1= highest)

After the insertion of these data for a device, the user has to click on “add” button of the window in the bottom, in order to save the data: the system will add a row in the “Enabled References List” with the new device.

If the user wants to add more devices, he has to repeat the above actions.

If the user wants to delete a device, he has to click on the red icon  on the right of the device: the system will display the “Enabled References List” without the deleted row.

4.3.3.3.4. Event

Event submenu (see next figure) allows the user to display the list of the events (= “Topics” of Pace of the City service) which the user is registered to, to add other events and to delete a subscribed event.

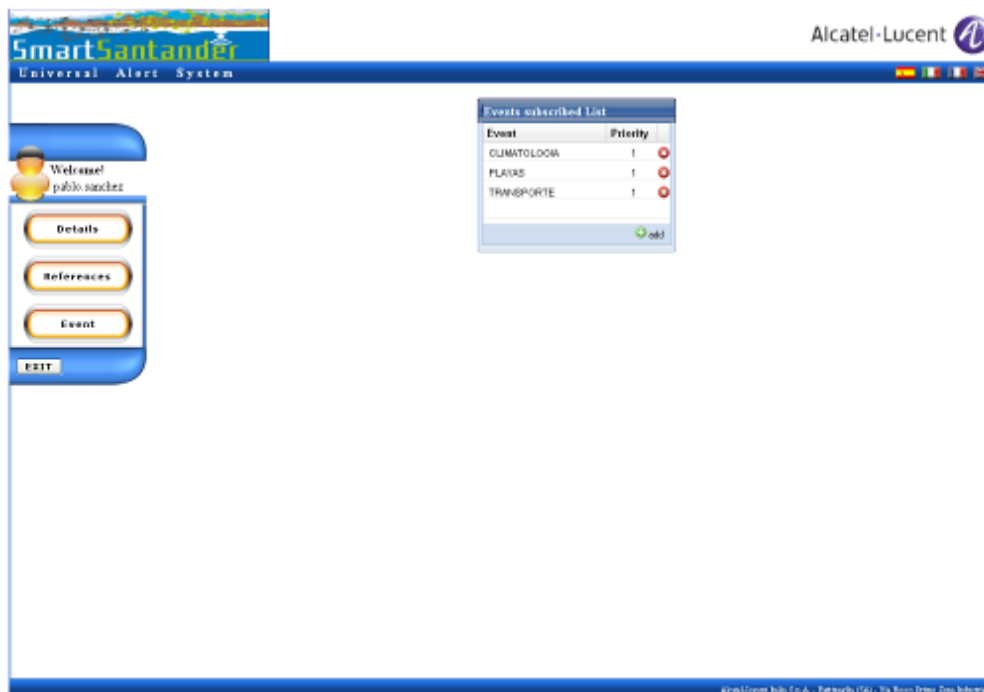


Figure 46 List of subscribed events

The “Event subscribed List” contains all the events already subscribed by the user.



PUBLIC, SMARTSANTANDER PROJECT

In order to subscribe an additional event, the user has to click on “add” button: a window will appear on the right with the list of the events allowed to the user (see next figure).

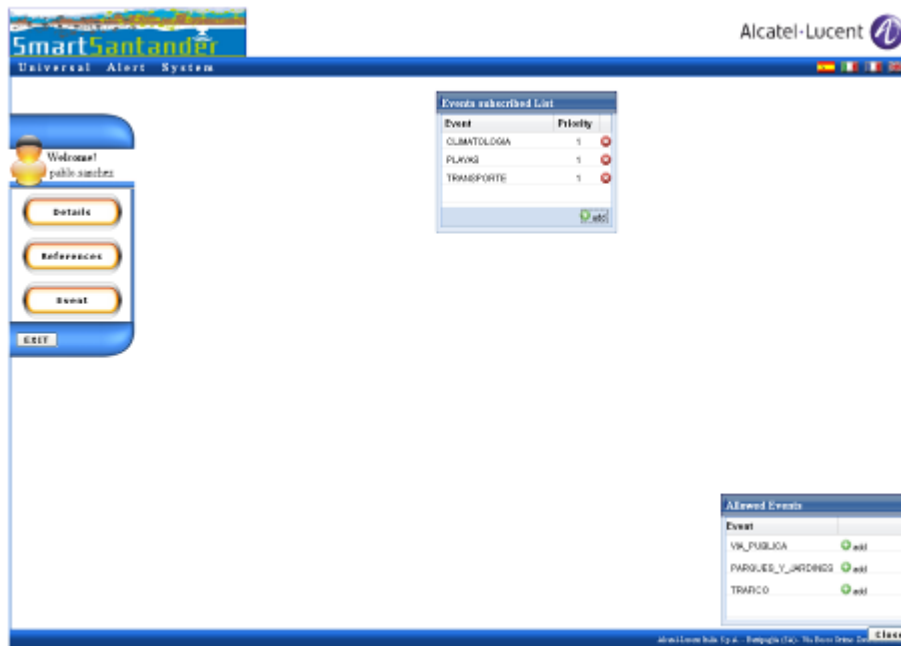


Figure 47 Event subscription

The user has to click in the new window on the “add” button on the right of all the additional events that he wants to subscribe.

If the user wants to see the description of an event, he has to click on the name of the event in the “Allowed events” list. It will be displayed a window with 3 fields: Event name, description and priority.

At the end the user has to click on “close” button: the new items will be displayed at the bottom of Events subscribed list.

The user can also select the days in the week and the timeframe when he wants to receive a message when the subscribed event occurs: the message will be sent to him only if the day and time of the event are included in the ones selected by the users. To use this option, the user has to click on the name of the event in the “Event subscribed list”: it will be displayed a window with the fields to be selected and their present values (see next fig.).



PUBLIC, SMARTSANTANDER PROJECT

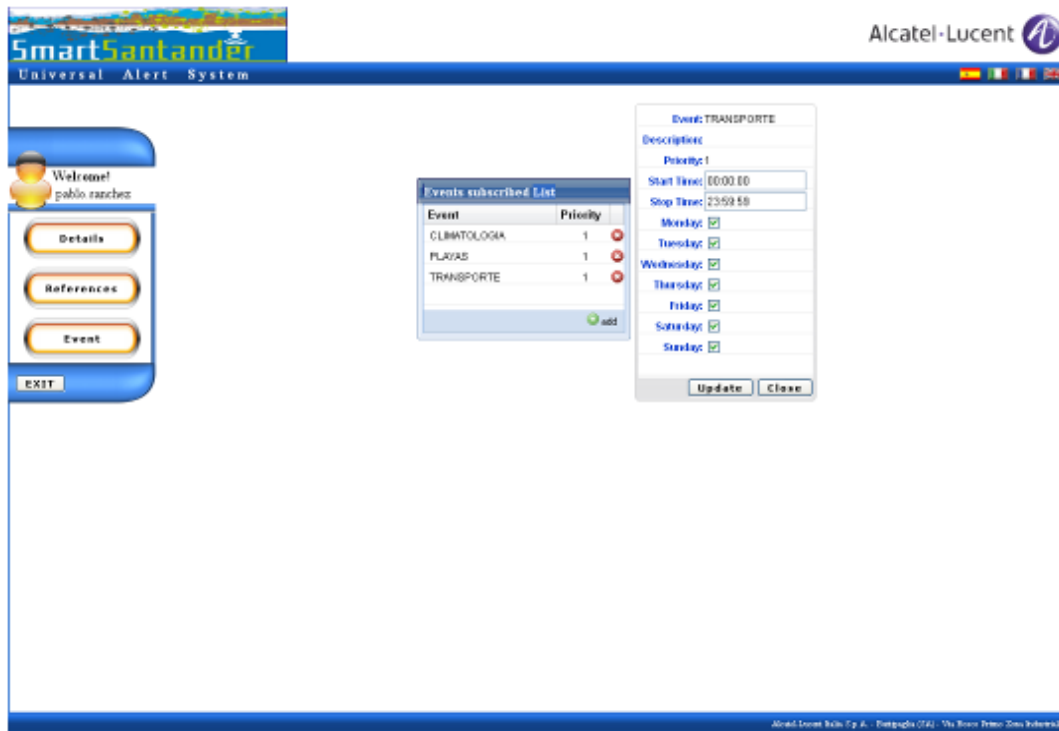


Figure 48 Days of the week and time (default values)

As shown in the previous fig., by default the values are configured in order to send the message in any day of the week and time. If the user wants to limit the timeframe of reception of the messages, he has to deselect the days of the week, define the start and end time of reception and click on “Update” field.

In the example of the next fig. the user wants to receive the messages of the event “Transporte” from 8:00 till 22:30, from Monday till Friday.

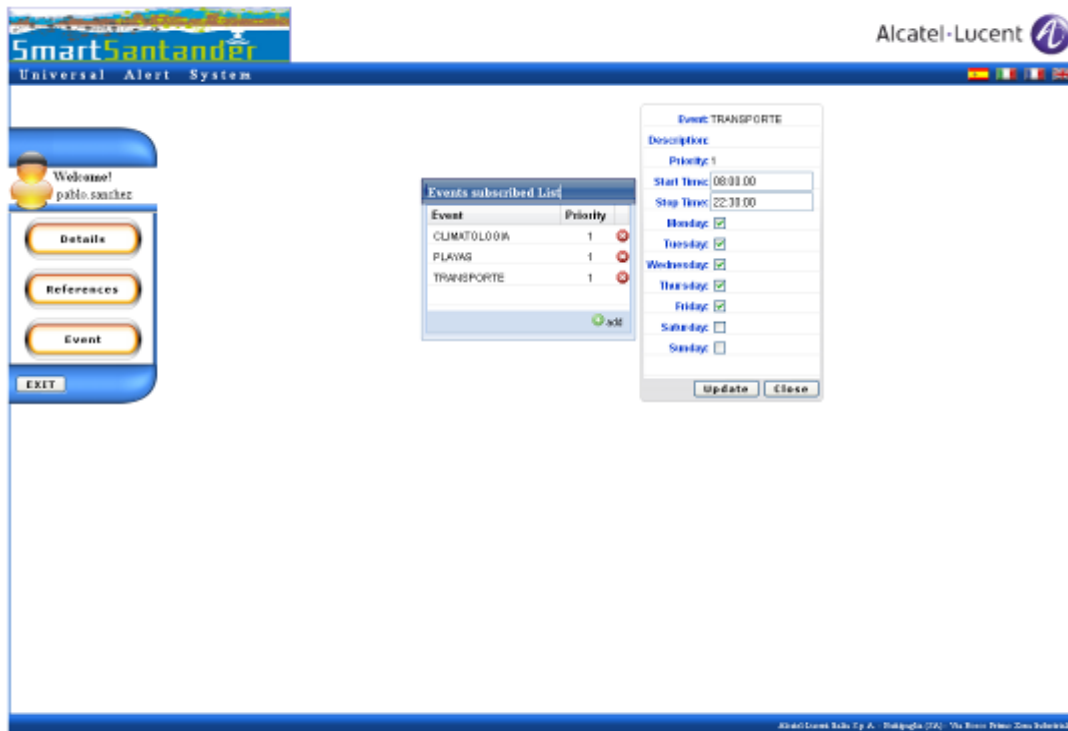



Figure 49 Selection of days of the week and time

The user can change his preferred day/time when he wants, using the same procedure as above.

If the user wants to unsubscribe an event, he has to click on the red icon  on the right of the event: the Events subscribed list will be displayed without the deleted row.

4.3.3.3.5. Exit

When the user has completed his actions, he has to click on "EXIT" button on the left: the session will be closed and the page in Figure 42 Figure 42be displayed again.

PUBLIC, SMARTSANTANDER PROJECT

4.4. AUGMENTED REALITY

In Santander, as in many other cities, there is a glut of information that may be of interest for citizens and tourists but, which may not be readily accessible. The reason for this include the heterogeneous mix of data sources producing data in different formats and the lack of a uniform data access specification/layer that would make data access easier. Information about transport, shopping, leisure activities, cultural agenda and so on is available in many different sites and is unknown to end users. To unify all these data sources and present them in a context-sensitive, location-aware manner to end users, the SmartSantanderRA app was developed.

With an AR application users can define their own preferences (language, touristic places to visit, monuments, etc) and have an interactive context-sensitive experience visiting the city rather than using traditional standalone applications. As an example, the video stream produced by a smartphone camera will be presented to the user of the AR mobile application which augments the live feed of the camera with virtual objects (mainly digital content, video, texts, and photos) of the Point of Interests (POI), based on the current position of the device thereby creating an augmented view of the reality.

For each Point Of Interest, the app provides a description of the place or reference image to be used in the AR view and the type of content to be superimposed. The content itself (3D model, image, videos, audio, etc.) is stored in the AR Server. Moreover, this server is in charge of getting all the real-time data from the Santander city Council legacy System (transport Services, city agenda, etc.).

As an illustration of the type of service the Augmented Reality application supports, the service provides a touristic experience through its “stroll in the city” mode. With the application in this mode, the tourist will receive information on specific monuments in his preferred language as he/she strolls around the city. This, in general, enhances the serendipity effect of the tourist visit.

Furthermore, placing NFC tags on certain shops in the city provides new opportunities for shops to build or strengthen customer relationships. The shops can explore the relationship between physical presence and the web. The users can get specific information about the shop, for instance, opening hours, contact, special offers, accessibility in the shop, etc.

From the City Council perspective, the placement of NFC tags in strategic places in urban facilities will provide location-sensitive information to the citizens. Also the use of location-aware applications using Augmented Reality technology allows the municipality to provide better touristic services to visitors.

The SmartSantander platform offers to the augmented reality scenario the possibility of generating observations with relevant data to generate new services. These observations include:

- Position information: based on smartphone sensors such as GPS, digital compass and accelerometer (location based tracking) or tags reading or with a combination of the two techniques. In this sense, the AR application will support both QR codes as well as NFC tags.
- Users Preferences: the language, the kind of POIs searched the individuals POI visited.
- Devices: device capabilities and OS.

Examples of scenarios that involve the Augmented Reality Service include the following:

**PUBLIC, SMARTSANTANDER PROJECT**

- Carlos is going to the shopping mall with his family in the weekend. He wants to buy a new suit because he will have an important work meeting in two weeks. At the shopping mall Carlos passes by a shop that sells suits. By the entrance there is an NFC tag that Carlos reads using his mobile phone. From reading the NFC tag Carlos is redirected to a web page that displays information about the shop; this information includes opening time, a map of the shop and a list of promotions the shop is currently offering. Carlos can see that there is a 25% discount in suits of the new collection. He decides then to have a look into the shop.
- Antonio, that is at home, is going to visit his grandmother by bus. While he is walking to the bus stop, he checks with SmartSantanderRA app that next bus arrival is in 25 minutes. He wants to use this time so He decided to buy some flowers to his grandmother. He don't know where to buy the flowers... No problem, he launches the app and look for the shops. He finds one close to him and uses the map navigation option to get to the shop. He buys the flowers and goes to the bus stop. His grandmother will be very happy for sure!
- Juan is visiting Santander for the first time. He decides to visit the city where he wants to see some museums. However he is not aware where the museums are located neither what kind of museums the city has. When going in the city Juan sees a building that looks like a cathedral. He takes his phone and uses an augmented reality application, in the image displayed by his camera he can see that he is in front of the "*Catedral de Nuestra Señora de la Asunción de Santander*"; some historical information about the cathedral is also displayed.
- Pablo is a bored citizen at home that is wondering what to do on Saturday, so he decides to connect to the SmartSantanderRA app with his smartphone, and there he discovers that there is a new art gallery on the local art museum. So he decides to visit the new art gallery, together with his wife.
- Maria is a Santander citizen that wants to play a football match with her friends next Saturday. She uses the app to see the weather prediction for this day, she find out that is possible that rains... She needs to book a sports hall. With the app, she visits the sport facilities list, and finds one that meets her expectations. The info shows in the app includes the phone number of the sports hall, so she calls and books it.
- Felipe wants to go shopping to the city center. He doesn't want to spend too much time to park the car. He sees on the real-time traffic webcams that there is a traffic jam in front of the underground parking he plans to go. He finds another underground parking with the help of the app and he arrives in five minutes.

The following use case diagram shows the functionalities or services the user can interact with:

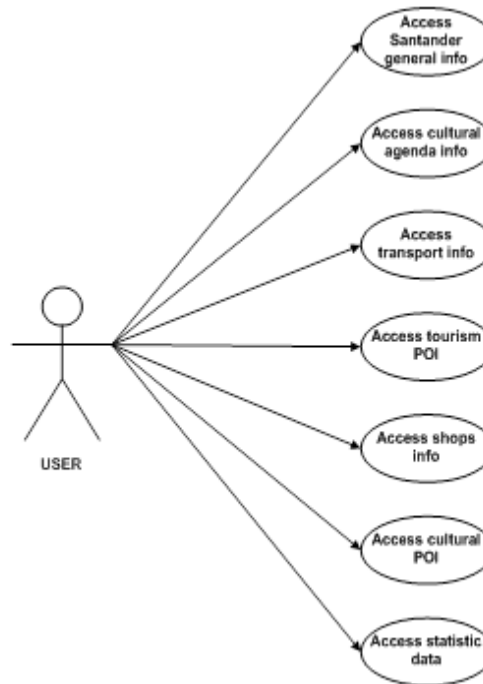


Figure 50: Augmented Reality – Citizen Use Case Diagram

4.4.3. System Overview

To realize this use case, two software components were developed: a mobile application (SmartSantanderRA) for the users and a server component (Augmented Reality Server, AR Server in short) which runs the service business logic and bridges the application with the SmartSantander platform.

The AR Server is responsible for collecting all the information needed from other external organizations servers that collaborate with the city council. For this purpose, it stores a database with all the poi static data. Moreover, this server accesses the legacy system (services previously developed by the Santander City Council) to get the real time data from the public transport services, city agenda, city news, traffic webcams... The communication between the Augmented Reality Server and the Legacy System is based on XML Web Services (using SOAP as transport) and XML for parsing data.

The following diagram shows the composition of components required to implement the AR Service and the interactions between them:

PUBLIC, SMARTSANTANDER PROJECT

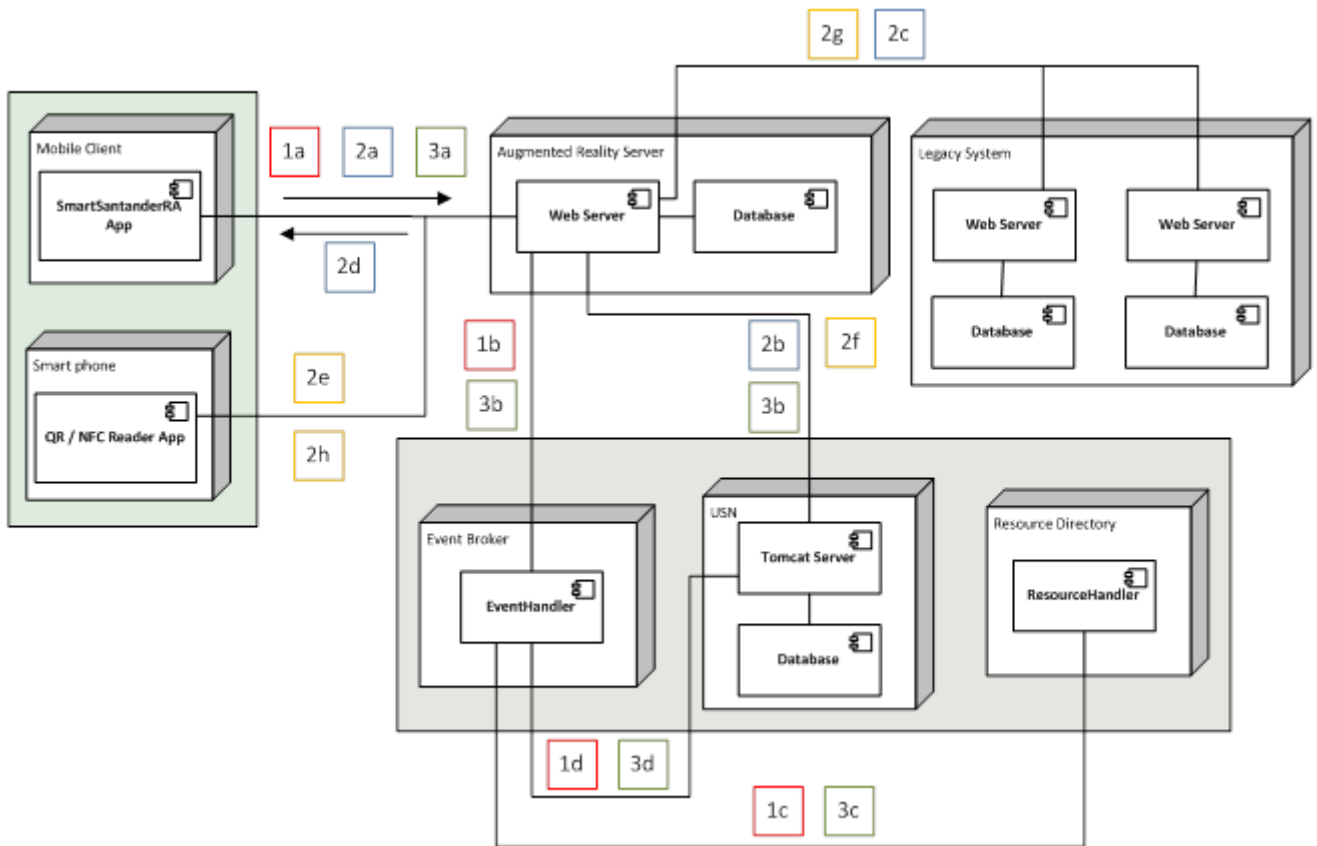


Figure 51: Augmented Reality scenario - Deployment diagram

We hereby describe the functionality required for this scenario and the interactions between the components to realize this functionality.

1. Register Device

- a. Devices get registered in the SmartSantander platform through the Augmented Reality Server. This registration is anonymous. From the mobile client information like phone model and manufacturer is sent to the Augmented Reality Server which stores this information locally and generates a unique identifier for the mobile client. This UUID will be used by the mobile client in all the calls to the Augmented Reality Server. If the registration of the device to the SmartSantander platform was successful then the Augmented Reality Server returns the generated UUID to the Mobile Client.
- b. A new device must be registered in the SmartSantander platform. To this end, the Augmented Reality server sends a registration message containing device information to the Event Broker component. This component is the link between all the components of SmartSantander platform involved in the device registration process.
- c. The Event Broker component sends the registration information to the Resource Directory so the device can be registered in the SmartSantander platform. The Resource Directory returns a device Id.

PUBLIC, SMARTSANTANDER PROJECT

- d. The Event Broker also sends a registration message to the USN containing the device information and the device Id.

The following diagram shows the interaction between the components for a device creation:

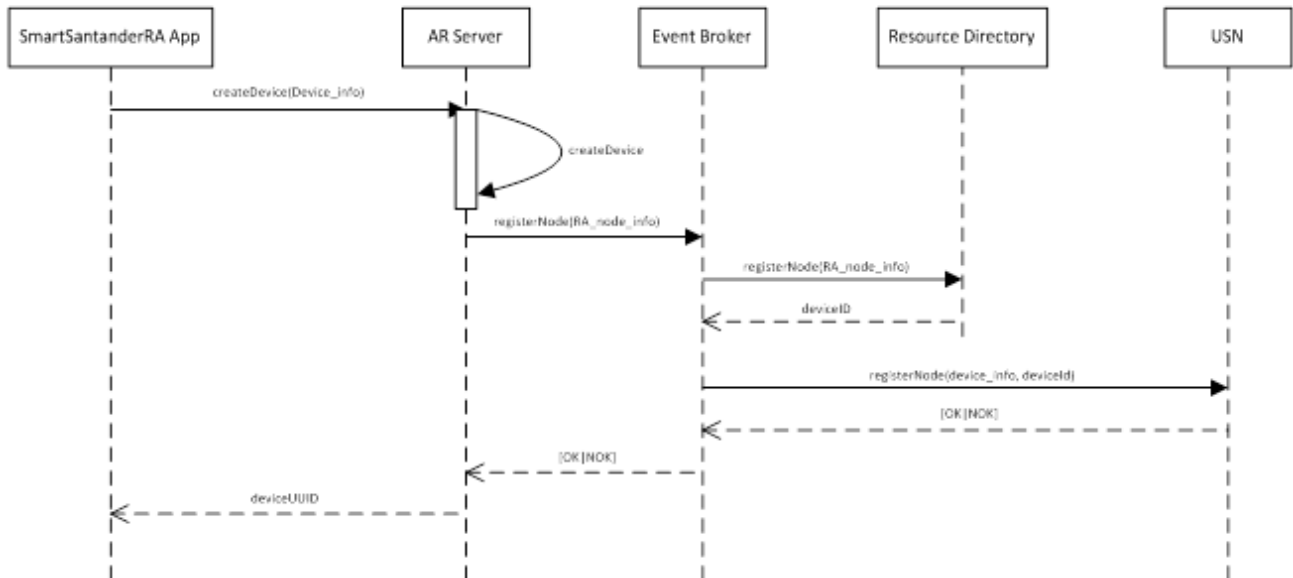


Figure 52: Sequence Diagram: Register Device

2. Data Request and observation sending

- a. Every time the SmartSantanderRa App queries the AR server, it attaches information (about its GPS position, device language, device Operation System, device UUID...). At this point in the AR Server, two new messages are created: one to send the attached info to the USN; and another one that query the System Legacy for real time information if it is necessary or access to de RA database to get the request data. The attached information is forwarded from the Augmented Reality server to the USN. The message sent to the USN is an “Observation Message” containing the measured values.
- b. If it is needed (real – time info like next bus arrivals estimated time). The AR Server forwards the request to the Legacy System. If not, the AR Server collects the info in its own database.
- c. The SmartSantanderRA app gets the data response.

The following sequence diagrams show the interaction between the components involved:

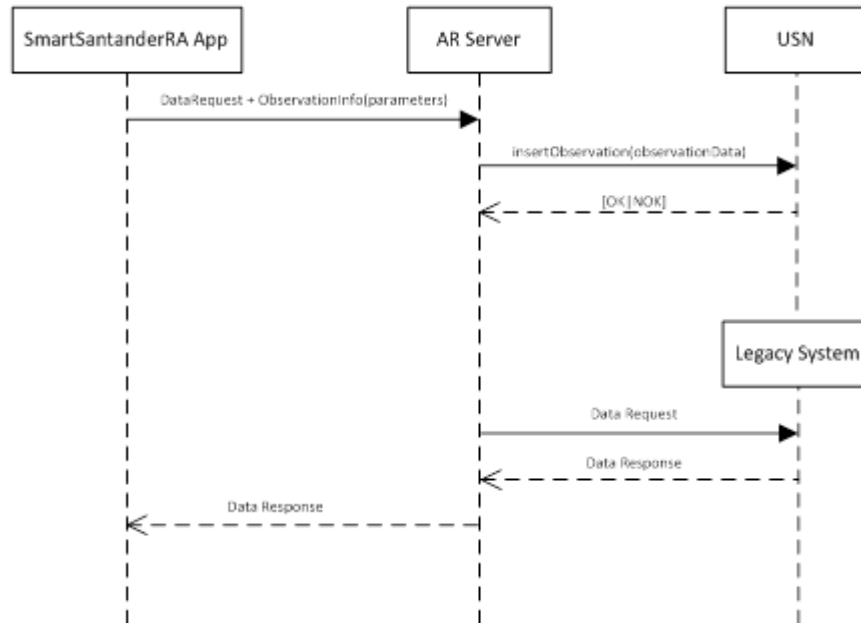


Figure 53: Sequence Diagram: Observation Sensing

The end users can also get the benefits of reading the QR/NFC tags deployed around the city without having the SmartSantanderRA app installed in their smartphones. In this case, each time a QR/NFC reader decoded the tag info it will request the data to the AR Server, but as the device is not registered in the SmartSantander platform all the observations generated will be associated to a general device.

3. Delete Device

- a. From the Augmented Reality app the device can be deleted via an unregisteDevice request that is sent to the AR server.
- b. The Augmented Reality server sends a unregister device message with the device id to the Event Broker.
- c. The Event Broker calls the unregister method in the resource directory.
- d. The Event Broker sends a unregister message to the USN.

The following sequence diagram shows us the interaction between the components:

PUBLIC, SMARTSANTANDER PROJECT

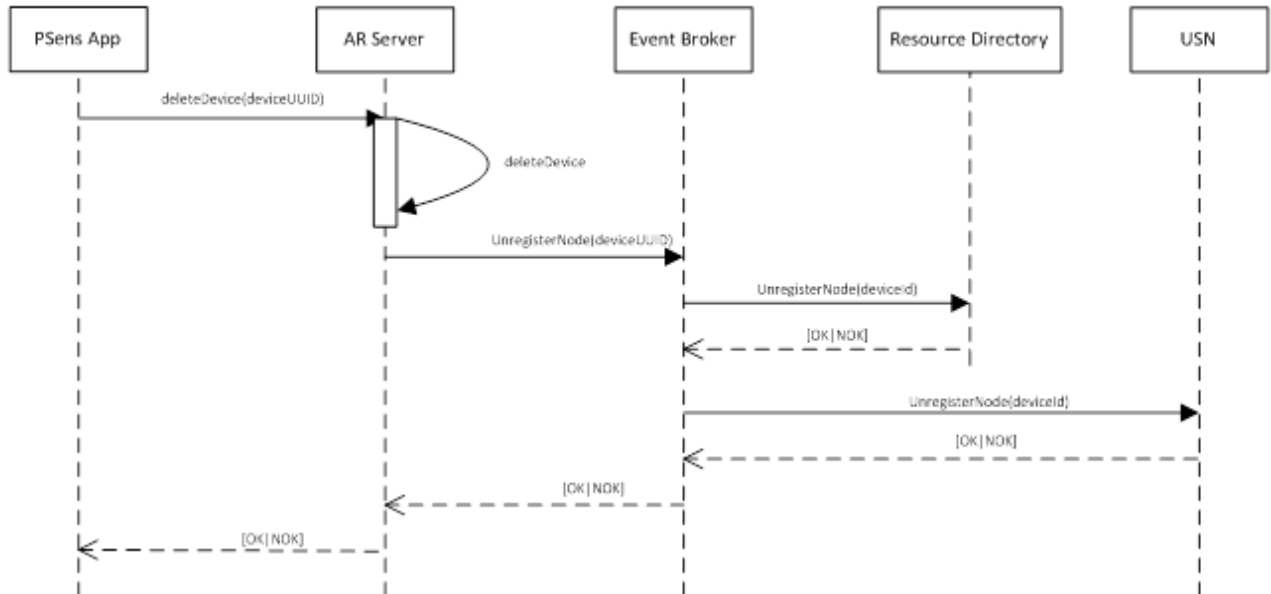


Figure 54: Sequence Diagram: Delete Device

4.3.4. Applications Description

We have developed two mobile applications; one for the IOS platform; for the Android platform. The two applications have similar functionalities and usability, in the following sections screenshots of both applications are shown interchangeably.

4.4.3.3. SmartSantanderRA IOS and Android mobile Application

The Smart SantanderRA application provides information on all areas of interest in the city. The data is structured in two groups:

- Augmented Reality data: The data showed is based and the current position of the device. The app gets the data of a configurable number of POIs closed to the current position.
- City General data: The data showed is independent of the position of the device.

When the app gets started the home view is presented. As the following figure shows the user can interact with 8 buttons:

PUBLIC, SMARTSANTANDER PROJECT



Figure 55: SmartSantanderRA: Home View

1. **About button:** Shows the application version and the partners involved in the development.
2. **Settings button:** Allows the user to configure how the augmented reality views are going to work. The user can define:
 - a. POI
 - i. The max distance of POI search.
 - ii. The max number of POI shown.
 - iii. App Mode. User can select on line mode or off line mode. Using the “off line mode”, makes the app download all POIs data and stored it in the device. It is recommended using this mode only with Wi-Fi connection.
 - b. Data Refresh type
 - i. Manual. The user has to refresh manually the data.
 - ii. Auto. The data get refresh when the user changed its position in the desired amount of meters.
 - c. Position
 - i. Auto. Use the device current position to query for closed POI
 - ii. Fixed. Use a fixed position defined by the longitude and latitude the user introduce.

The following figure show the Settings View interface:



Figure 56: SmartSantanderRA: Settings View (IOS vs Android)

3. **Santander button:** Shows the user relevant info about the city. This info is independent of the device position. Here the user can find info about the city news, beaches info (with real-time webcams), parks and gardens info, weather info (data from AEMET), traffic webcams, tourist offices info, museums and exhibition info, libraries info, number of interest info and sport facilities info. Some examples are:
 - a. The following figure shows how to get the info about a “Primera Playa del sardinero” beach:



Figure 57: SmartSantanderRA: Santander View. Getting Beach Data

PUBLIC, SMARTSANTANDER PROJECT

- b. The following figure shows how to get the info about the traffic webcam in “Cuatro Caminos”.

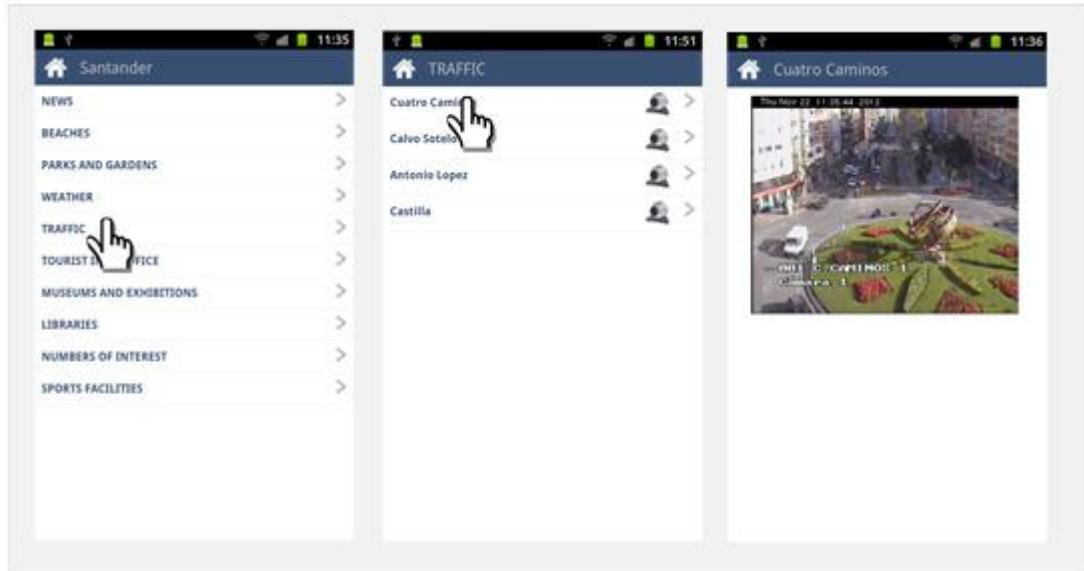


Figure 58: SmartSantanderRA: Santander View. Getting Traffic Data

- c. The following figure shows how to get the phone number of city council department of public works Control (“control de obras”).

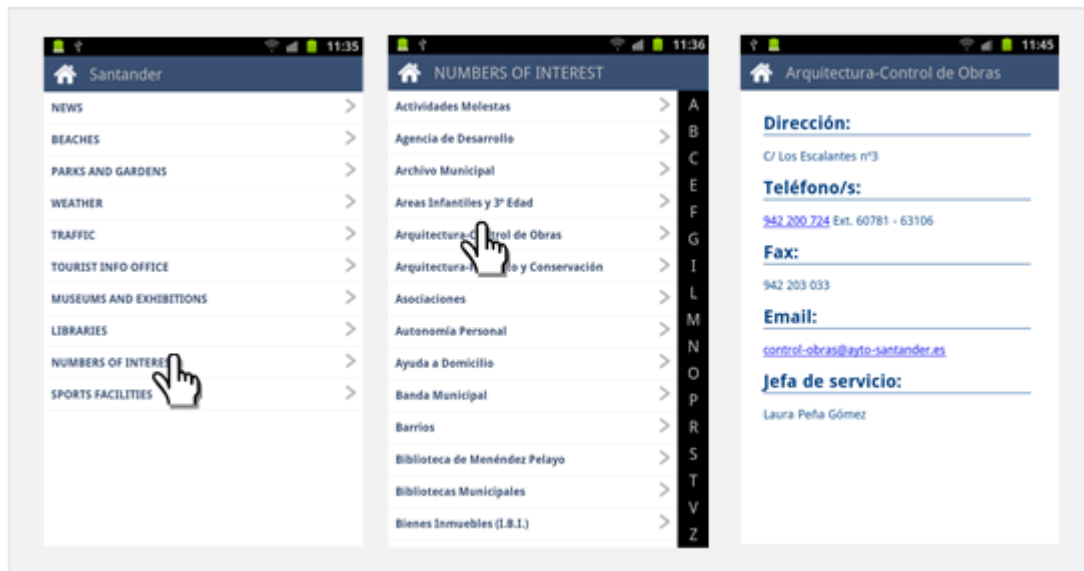


Figure 59: SmartSantanderRA: Santander View. Getting Number of interest Data

- d. The following figure shows how to get the info about the sport hall “Palacio de deportes de Santander”.

PUBLIC, SMARTSANTANDER PROJECT



Figure 60: SmartSantanderRA: Santander View. Getting Sport Facilities Data

4. 5. 6. 7. 8. **AR buttons.** This buttons fire the AR View of the app. When the user clicked one of this button, the app ask the server for the closest “x” (this integer is defined by the variable max number of POIs, described in the settings view), of a defined type (depends on the button the user clicked: tourism, commerce, agenda, transport or culture), POIs to the current position. The smartphone turn on its camera on, and the POIs icon are shown over it. When the user click over an icon a short description is shown and three buttons appears. The following figure illustrated the result:



Figure 61: SmartSantanderRA: AR transport View. Bus stop selected

1. **Info button.** Shows the data associated to the selected POI. The app navigates to the POI details view.

PUBLIC, SMARTSANTANDER PROJECT

2. **Street View button.** Shows the Google Street View centred in the position of the POI.
3. **Navigation button.** Shows how to get to the selected position using the Google Map navigation option.

The following figure shows an example of the information accessible from the AR View by clicking on buttons 1, 2 or 3.



Figure 62: SmartSantanderRA: Extra info accessible from AR view.

The screenshot in the left shows the info of the next bus arrivals in this bus stop (real-time info). The screenshot in the middle shows the Street View centred in the bus stop position. Finally the screenshot in the right show the route from the current position to the bus stop in a map.

4. **Home button.** Returns to home view.
5. **Refresh button.** Refresh the POI list that the view showed.
6. **List button.** Changed to the list view. This view offers a different form of interact with the data. All the POIs are show in a list format. It is available a POI search according to POI names and descriptions. A long click over a row in the list opens a new panel that offers the some options buttons 1, 2, 3 described above. The next screenshots shows the list view appearance:

PUBLIC, SMARTSANTANDER PROJECT



Figure 63: SmartSantanderRA: POI List view.

7. Map View. Changed to the map View. This view shows all the POIs using Google Maps (or the new IOS Map depending on the device). When the user click over an icon, the extra info appears. If the user clicks over the extra information the app will navigate to the POI details view. The following figure show the map view:



Figure 64: SmartSantanderRA: POI Map view.

PUBLIC, SMARTSANTANDER PROJECT

8. **QR Reader button.** Opens the QR reader view. It allows the user to scan QR codes. When a QR code is decoded the app launches the Poi details view. The next screenshot show the QR Reader interface and the result after the QR code is decoded:



Figure 65: SmartSantanderRA: QR Reader view.

9. Settings button already discuss before.

Finally, there is an extra feature in the Android version of the app, the NFC reader. Some Android devices have a built in NCF chip for reading NFC tags. For these devices a new option is shown to allow reading NFC Tags inside the SmartSantanderRA app. This view is accessible by clicking the menu hardware button of the device. When a NFC tag is decoded the app launches the Poi details view. The following figure shows the appearance of the menu and the reading process:



PUBLIC, SMARTSANTANDER PROJECT

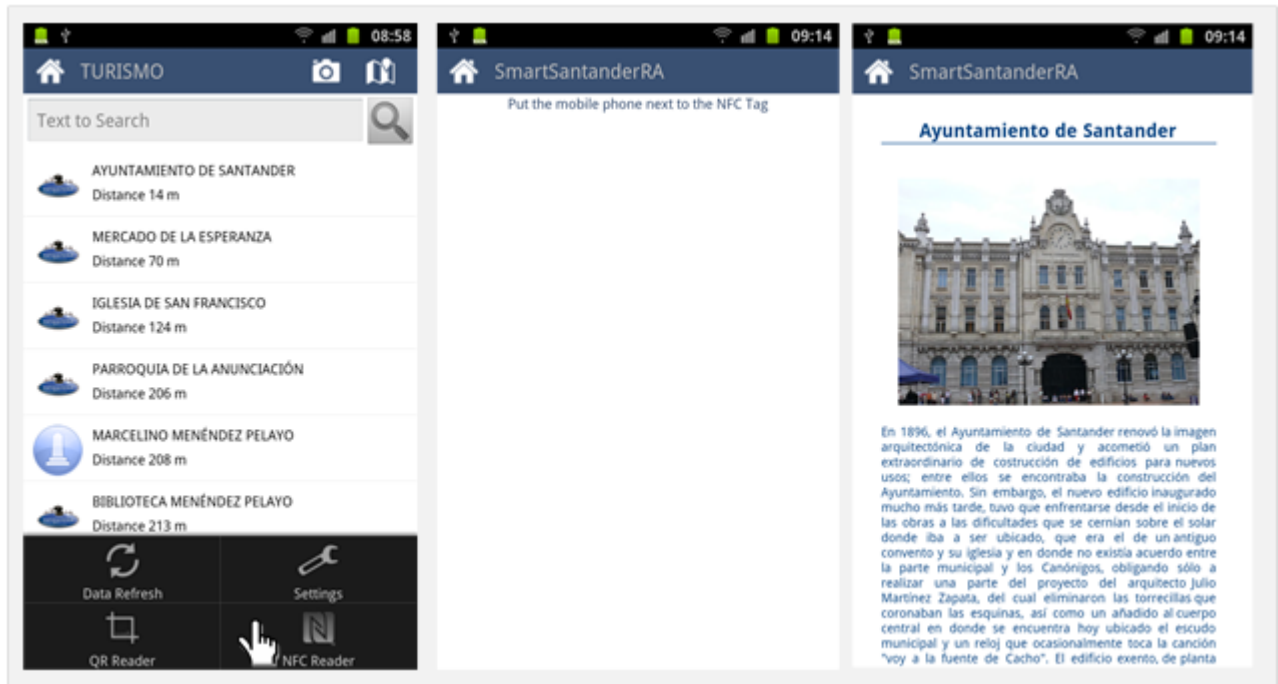


Figure 66: SmartSantanderRA: NFC Reader view.

As the above figure shows, the navigation menu in the Android version app is divided into two menus: one on the top of the view, to navigate through the AR view, list view and map view; another one that appears when the option button (hardware button) of the android device is clicked.

PUBLIC, SMARTSANTANDER PROJECT

4.5. IRRIGATION

The Precision irrigation service provides a means for evaluating plants' requirements in water and supplying them with the right amounts, at the right time. Precision irrigation focuses on individual plants or small areas within a park, rather than taking a 'whole-field' approach. Typical existing systems aim to supply water based on static timetables, without taking into consideration the different needs of the various plants and trees within a specific area inside a park or garden. Thus, a certain degree of flexibility is required in order for such an irrigation system to adapt to specific plant species and their growth while lessening the effort required by the park personnel.

In terms of optimization, the development of such a precision irrigation and park monitoring application also makes it easier to prevent water and labour wastage and lessens the negative impact on the environment. Moreover, the real-time information from the field enables park technicians to adjust irrigation strategies at any time. Instead of taking decisions based on some uncertain average condition, which may not be even close to reality, or having to be physically present on-site constantly, a precision park irrigation approach recognizes differences and automates management actions accordingly.

The current deployment in Santander is situated at two areas, the Las Llamas park and gardens around the Magdalena Palace area. A total number of 48 IoT sensor nodes, covering an area of 55000 m², are deployed at key positions inside these two areas, equipped with special agricultural sensors measuring parameters like:

- air temperature and humidity
- soil temperature and moisture
- atmospheric pressure
- solar radiation
- wind speed/direction
- Rainfall

4.5.1. Objectives of Service

In short, the Precision Irrigation service has the following overall objectives:

- Sense various environmental/agricultural parameters of the parks in Santander.



PUBLIC, SMARTSANTANDER PROJECT

- Report a set of specific parameters to the municipal gardening authorities in order to make decisions and take actions.
- Control the installed irrigation system.
- Alert gardening authorities when detecting irregular conditions.
- Ease the information sharing with any agency or authority involved in the parks and gardens management.
- Automatically apply irrigation (in a distributed manner) to different sub-areas of the park. For each sub-area in the park it has to be possible to define and configure irrigation strategies as a function of soil, types of plants, current and predicted weather conditions, etc.
- Measure/estimate important KPI's like irrigation applied, water absorbed, water waste, and general savings in water.

In terms of user interface (presented in more detail in Section 6), the service will have 2 major software clients, a web application utilised for both administration and monitoring, and a Android smartphone application which is used primarily for viewing measured values and setting/receiving alerts for specific events within the scope of the irrigation scenario.

Examples of use case scenarios for this service include the following:

- Jorge works at the municipal park as a gardener. He wants to cut down water costs and minimize the time he spends maintaining the park irrigation system. He wishes to use both the system and his experience in order to produce a new irrigation schedule and test it. He logs in the web application interface and defines a number of different geographical subareas, which correspond to various plant species. After that, he uses the system for a time period, while data are being generated and stored in the system describing the overall process, He now has specific logs and data to present to the city council regarding the operation of the park.
- Manuel works together with Jorge and checks the irrigation programs to be executed along the weekend. No rain was expected but the situation suddenly changes and rain begins falling on Sunday morning when no one from the gardening department is on duty. While being on travel, Manuel receives an alert from the Android smartphone application, informing him that the pluviometers are registering an important amount of water, so he connects to the commercial irrigation system used in the park and stops all the irrigation programs currently being executed.

PUBLIC, SMARTSANTANDER PROJECT

- Evan is a CS researcher wishing to develop and test machine learning techniques for porting an irrigation protocol, among different sites that share similar environmental parameters and similar vegetation. He also wishes to incorporate online weather forecasts in the irrigation schedule creation and compare it with the ones produced by other systems. He uses the system server to get readings and the web application to have a quick overview of the environmental conditions.
- Alessandra wants to develop serious games: the concept will be to assign to high school students (Biology class) to monitor plants with special mobile apps specific regions in the monitored parks. Using such applications they will record the development of plants: size, colour of leafs, photos, etc. Using the system they already have a detailed view of the kinds of plants in the monitored park areas, without needing onsite inspection. They can also view charts with historic data on irrigation, air and soil parameters and schedule their activities around such data.

The system services discussed in the above examples are summarised in the following use case diagram.

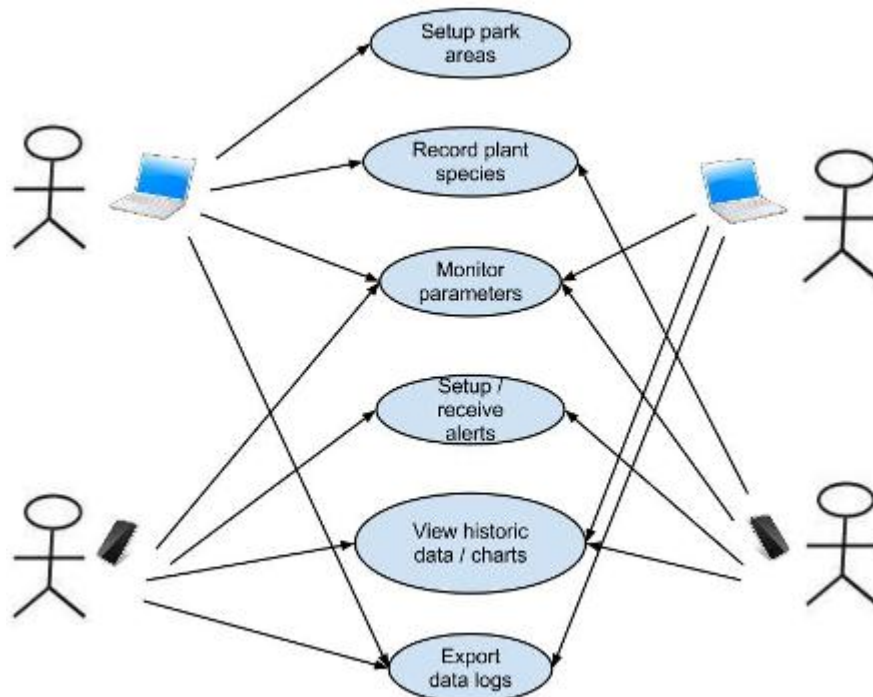


Figure 67 Precision Irrigation Service Use-case Diagram

4.5.2. System Overview

Concerning interfacing with the existing infrastructure we plan to utilize a web service for to broker between data providers, data consumers and application developers. We have used a web service for data sharing via

PUBLIC, SMARTSANTANDER PROJECT

the Web where end-user applications can provide and consume data via REST and WebSockets technologies. End-user applications interested in retrieving data from the web service can obtain them in various formats such as tab delimited text, HTML, JSON, RDF, CSV, WiseML.

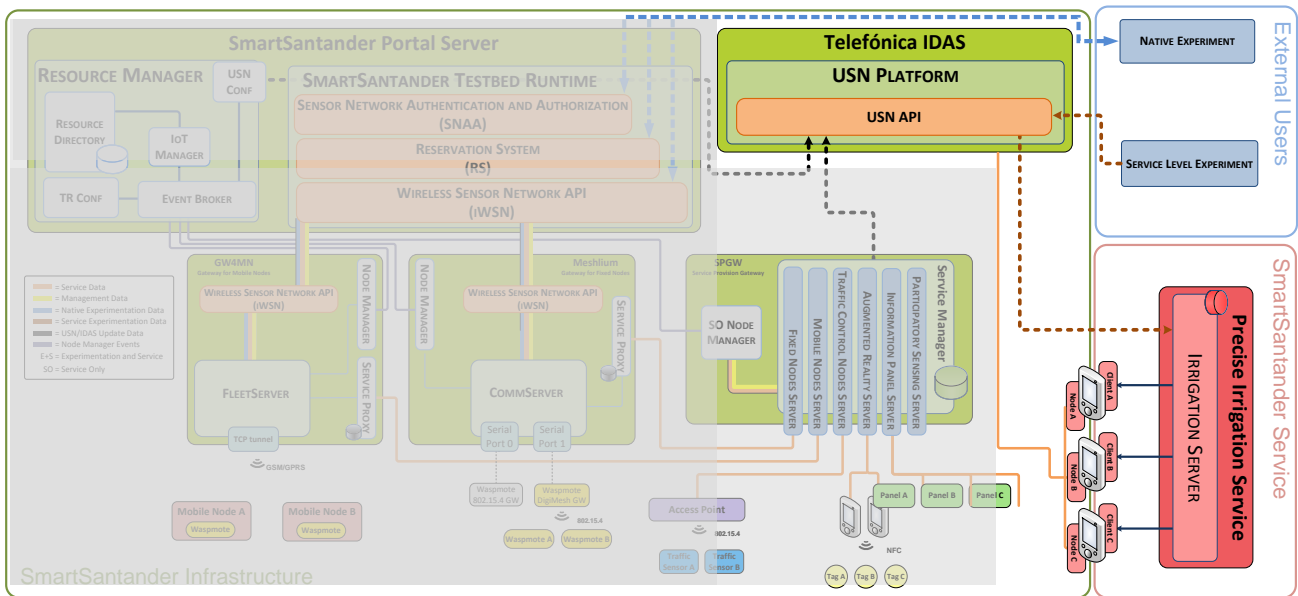


Figure 68: Irrigation architecture

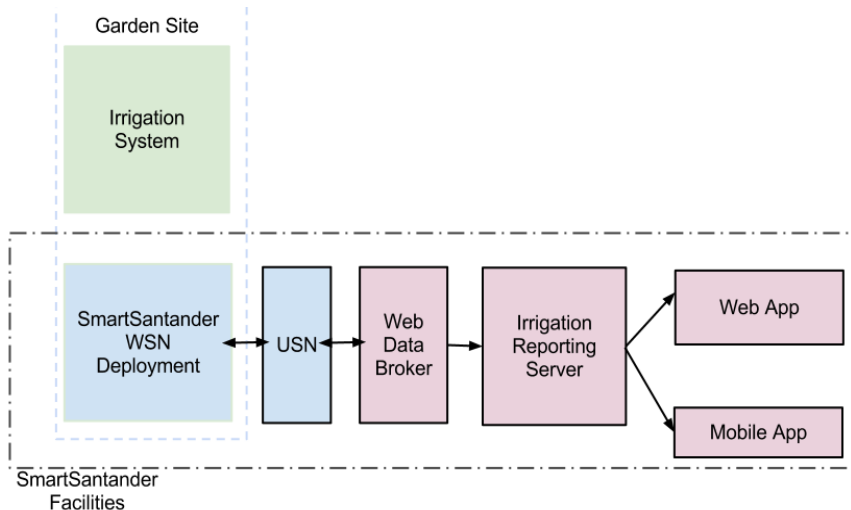


Figure 69 Software Components

The overall architecture of the service is depicted in the previous figure. Irrigation service is interfaced to the main facility through USN platform in order: 1) to register the irrigation IoT devices with the Irrigation Server,



PUBLIC, SMARTSANTANDER PROJECT

and 2) to stream data from the registered irrigation IoT devices to the Irrigation Server. In addition, the Irrigation Server aggregates the stream data in order to produce the desired reports. The Irrigation Server consists of the Web data broker, the Irrigation reporting server and the web app and the mobile app following an MVC architectural approach for increased scalability and performance.

The system provides two ways for user interaction, a web application mainly for monitoring and administration and an Android smartphone application. Although their functionality overlaps to some extent, they are meant to be used by different user groups within the context of various use-cases (see also examples of scenarios in Section 4.5).

4.5.3. Monitoring and Administration Web Application

Suppose that the person responsible for the park areas wishes to utilise the system for the irrigation of the whole park or parts of it. After logging into the system, she has to setup the respective areas of the park and assign plant species to each one of them. The web application is organised according to the following logic:

- A set of parameters referring to the plant species and areas have to be inserted into the system, such as the reference evapotranspiration, the plant species factor and various microclimate factors.
- Such parameters serve to produce indicative irrigation schedules for subareas defined in the system.
- Input from sensors and actual irrigation applied helps to calculate overall water needs for the park, i.e., by combining projections with actual data in order to compensate for environmental or mechanical factors.

An example of setting up such parameters is featured in the following figure. The user inputs reference evapotranspiration values for each month of the year, defines plant species factors and microclimate factors that can be assigned to each one of the areas of the park (also defined by the user). These three parameters are used to provide an estimation of landscape irrigation needs. By defining this set of parameters and combining them with input from the actual environment (environmental sensors) and irrigation infrastructure (manual or automatically), the system can estimate if the currently applied irrigation scheme is sufficient or not, or exceeds the needs of the plants (e.g., due to rain or other environmental parameters).

PUBLIC, SMARTSANTANDER PROJECT

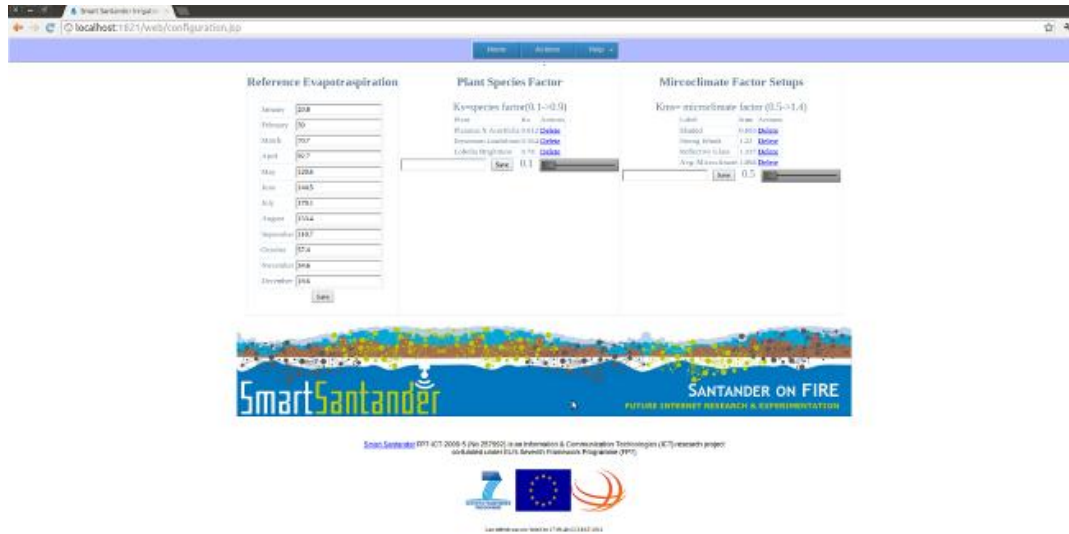


Figure 70: Configuration Page of the Irrigation Service

Geographical subareas can be defined easily through the web application interface. The system creates an overlay map depicting the SmartSantander infrastructure and all of the available sensors. On the sides of this overlay map, there a number of control options available. E.g., the user can select to display specific types of sensors only, to help deciding how to divide the available monitored areas into smaller ones, along with plant species criteria. He can also determine certain criteria, such as plant density or irrigation efficiency.

By clicking to specific points serially in the map, the user draws a polygon sequentially, which:

- Defines a specific subarea of the park.
- Automatically includes all sensors residing inside this area.
- Can be tuned for area-specific parameters.

All such polygon areas defined are assigned a name and description to help in easy identification later on. Administrators can add, modify or delete such areas at any point through the web application interface.

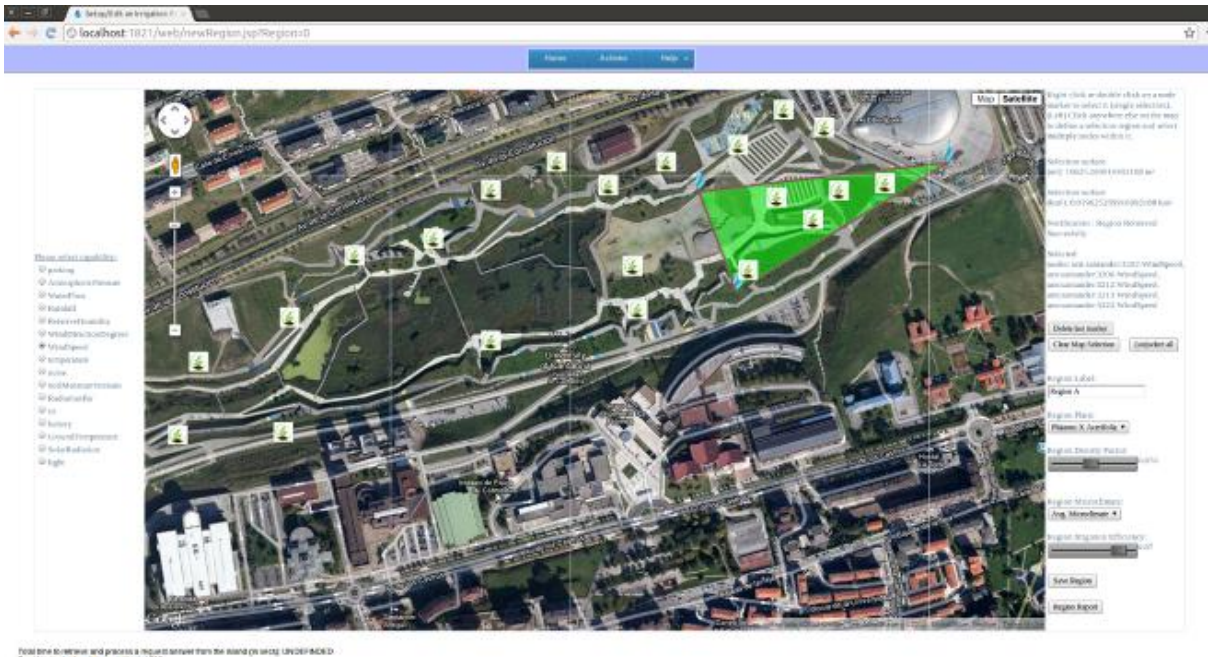


Figure 71: Region Management of the Irrigation Service

After setting up the necessary parameters for the system, users can monitor the gathered data for a specific region over time, in a single web page. E.g., monitor relative humidity, air temperature, ground temperature and soil moisture, along with suggested irrigation plan and actual irrigation applied, over a period of 2 weeks, or whatever time period is available through the stored system data. The system can dynamically produce graphs over a user-defined time period for all the parameters measured, and also provide a visualisation of the irrigation scheme applied and compare it with the one actually required. In order to provide such functionality the irrigation system has to provide automatically such data, or the administrator has to input it manually. Moreover, since all irrigation systems have a specific efficiency factor, i.e., a percentage of the water used does not reach its destination, these factors can also be fed to the system. This way, over a period of time the application can produce a more realistic overall view of the irrigation process.

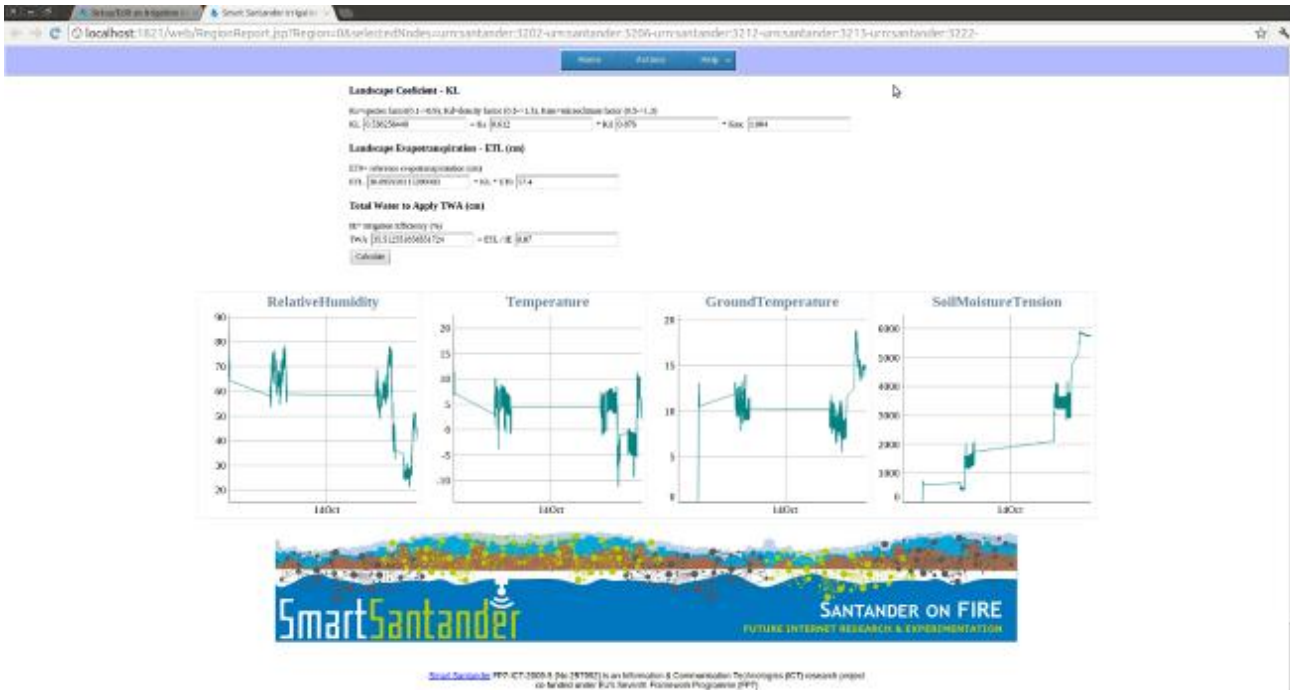


Figure 72: Region Report

4.5.4. Smartphone Application

A smartphone application, developed for the Android platform, complements the main web application providing easy access to the measured parameters inside the park areas, e.g., soil temperature or air humidity. Thereby, the gardening authorities and people involved in parks and gardens management can take accurate decisions based on almost real time information as the measured values are updated periodically. The system can post notifications regarding whether values from a specific sensor are outside a predefined and configurable range.

Once the user starts the application, icons representing the different types of IoT nodes deployed are shown. Just clicking any of them, a detail window shows the last information available from all the sensors. The application menu offers the possibility to change the view from one park to the other, access the configuration screen (notifications can be enabled/disabled and ranges for parameters can be set) and view the node list for each of the parks.



PUBLIC, SMARTSANTANDER PROJECT

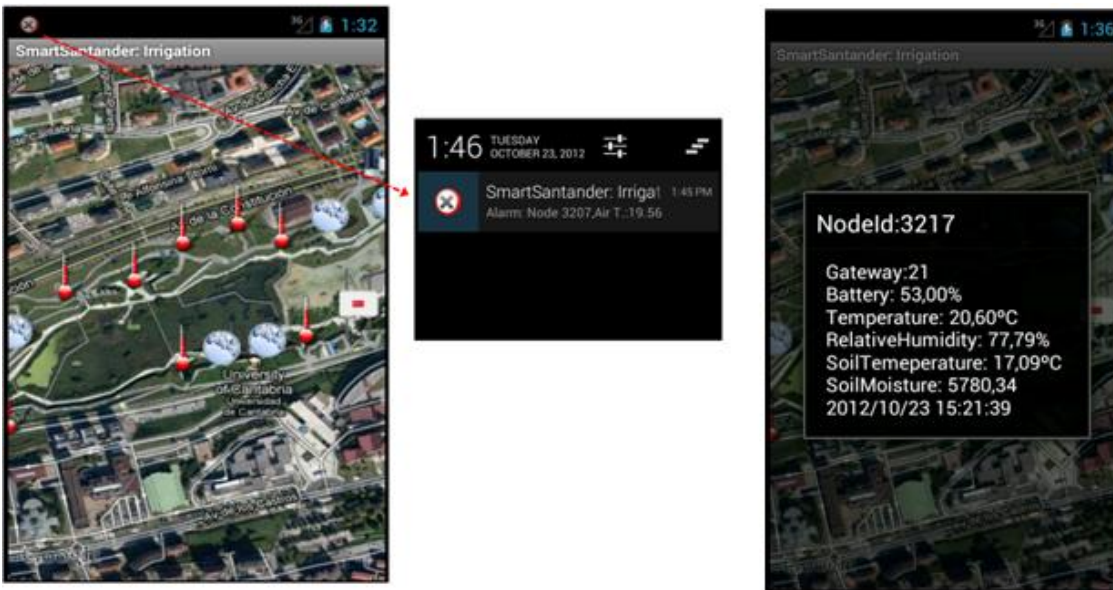


Figure 73 Irrigation Mobile App – Node Monitoring

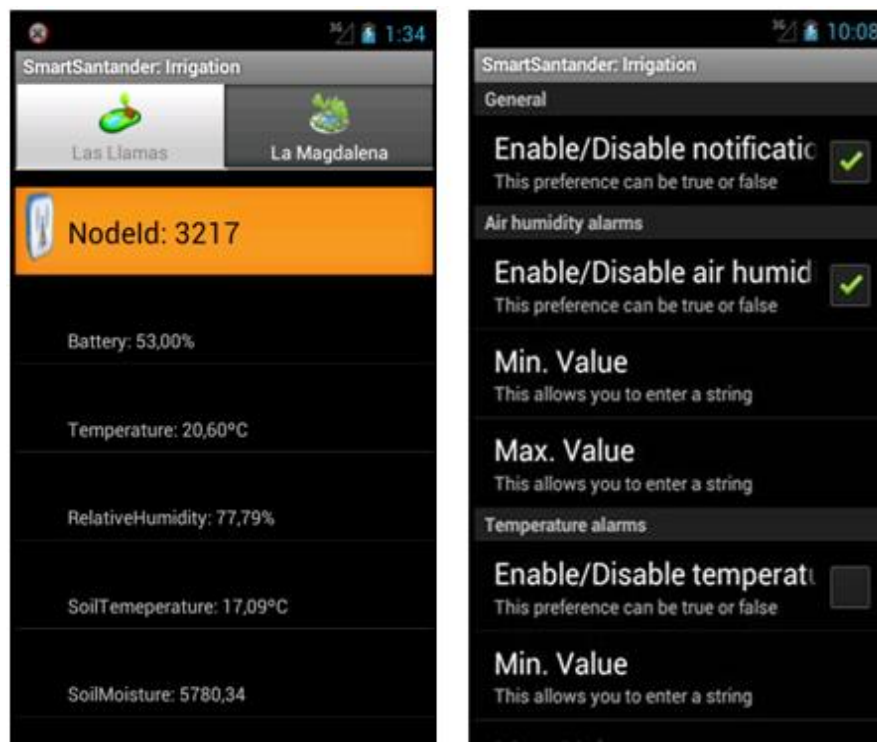


Figure 74: Irrigation Mobile Application – Node Report and Setup

**PUBLIC, SMARTSANTANDER PROJECT****4.6. PUBLIC TRANSPORTATION**

The system utilizes public transport vehicles (buses, trams etc.) and other public service vehicles (police cars, taxis etc.) to carry various IoT devices across a city. The main functions of the deployed system are the following:

- Continuous monitoring of several environmental parameters across the region covered by the transport and public service vehicles;
- Continuous tracking of public transport vehicles to enable full public transport fleet management as well as provision of services to the citizens (e.g. bus arrival time to a bus stop);
- Provision of other derived services such as traffic lights management based on the monitoring of the time taking for each vehicle to cover certain distances.

The devices continuously observe environment parameters, location of the buses as well as the current speed of the vehicles. All measurements are transferred to a central database. End users can query the system using a web or mobile phone application to get real time environment measurements as well as locations of the public transportation vehicles.

Examples:

- Milena lives in Belgrade and she commutes to work every day using public transport. She can travel to work using two different busses, one taking shorter and one taking much longer route. If she takes the bus going shorter route, she can save over 30 minutes of her time. However, this bus goes less frequently and the timetable is not reliable due to the traffic road works currently in progress. At the arrival at the bus stop, she sends SMS containing the bus stop number she is at as well as the bus numbers she could take to work. The system returns the estimated arrival times of the busses to the bus stop and she chooses to wait for the one that would take her quicker as it arrives only two minutes later than the bus travelling the longer route.
- John goes to his office by bus planned at 8.30 AM and comes back home with the bus planned at 5.30 PM, but often the bus arrives with some delay due to traffic. He needs 5 minutes to reach the bus stop both from house and from office and doesn't want to stay too much at bus stop to suffer hot weather in the summer and cold and rain in the winter. So he subscribes to the notification service asking to know the position of his bus at 8.20 AM and at 5.20 PM in the weekdays. In this way he receives a call

PUBLIC, SMARTSANTANDER PROJECT

on his fixed phone at home and a call on his fixed phone at office at the specified time, so he can leave in order to be just in time at bus stop.

- City of Pancevo council is interested in continuous monitoring of air pollution parameters in the city and for this purpose they can use the mobile phone or desktop application to connect to the system. However, they would like to be notified by SMS if one of the parameters goes out of normal range at any given time. They subscribe to the service specifying the upper and lower thresholds for each of the parameters and they receive SMS as soon as the notification conditions are met at any measurement point (i.e. bus). The message contains the exact location and time where the notification was triggered and the parameter that has exceeded the limit.

Urban traffic management and control in Santander is a public company responsible for the management of traffic signals in a city. They are interested in any potential traffic problems that might occur due to increased volume of traffic at any location in the city so the appropriate adjustments to the traffic lights can be made in order to clear the congestion area allowing faster filtering through the cross roads. They subscribe to the service specifying the notification conditions for each of the cross-roads location with the maximum time it takes the vehicle to cover the distance starting before the cross-roads end finishing just after the cross-roads. In this way, they receive notifications (SMS and email) as soon as the time necessary to cover certain distance around particular cross-roads is exceeded. In this case, the data is analyzed in more detail and if necessary, the traffic lights duration is modified so that more efficient filtering is achieved in a particular direction on the cross-roads.

- Citizens/experimenter perspective
 - Visualize in a map the latest environmental measures of a sensor from their mobile phones.

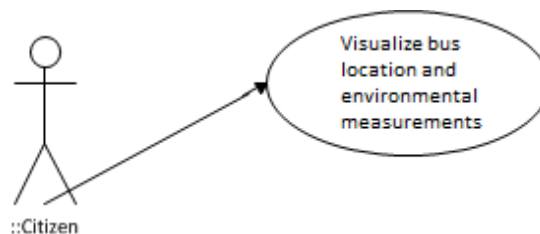


Figure 75: Public Transportation – Use case diagram

4.6.1. System Overview

In this section are described the main components in the Public Transportation Scenario, as well as their interaction. EcoBus is a commercial system in Pancevo, thus restarting/reprogramming is not allowed, thus,

PUBLIC, SMARTSANTANDER PROJECT

EcoBus platform performs data forwarding to the SmartSantander system using the SmartSantander’s components like NodeManger and USN. EcoBus-SmartSantander Integration is performed on the level of registration and data forwarding. Registration is performed using appropriate messages of the NodeManager component. NodeManager component performs appropriate registration in the SmartSantander’s RD and USN. EcoBus sensor devices are restarted in the SmartSantander like service nodes. EcoBus observations are forwarded to the USN. Figure 76. Displays the main components involved in the EcoBus-SmartSantander integration.

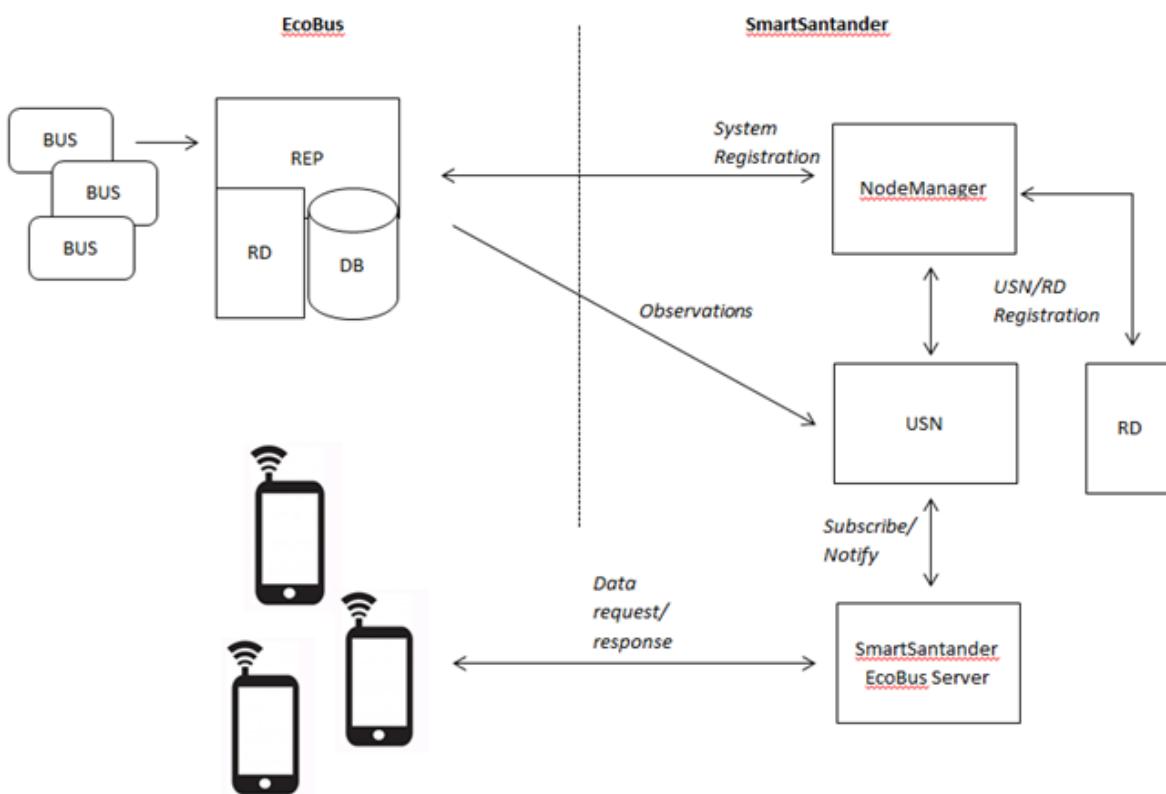


Figure 76: Components in the Public Transportation Scenario

EcoBus system performs service nodes registration in the SmartSantander using NodeManager component and data forwarding to the USN. SmartSantander EcoBus Server performs subscribing to the USN and receives Notifications on the every new EcoBus observation. Communication sequence is presented on the Figure 77.

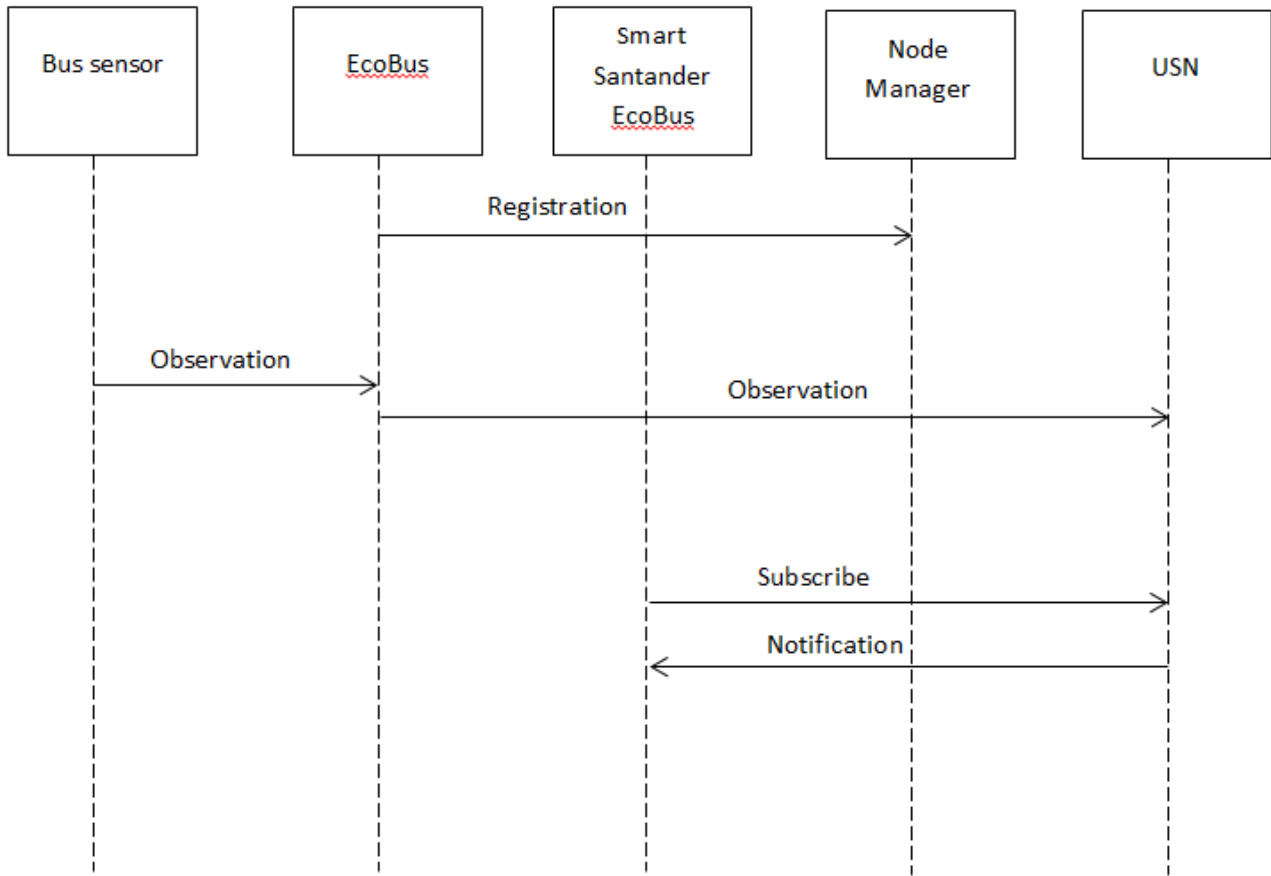


Figure 77. EcoBus-SmartSantander integration sequence

Current implementation performs forwarding only for observations with environmental measurements.

4.6.2. Public Transportation application

For the public transportation scenario we have developed an Android application. This application displays current bus location as well as collected environmental measurements. Data are collected using SmartSantander EcoBus Server by the USN subscribing mechanism. SmartSantander EcoBus Server provides simple XML interface to the end user client application. Android application performs periodical check for the new observations and performs data refreshing in the GUI on the every change in data. Additionally functionalities of the Android applications are: The best fit zoom and Keep screen on. This features helps to the user to easier monitor bus location changes. Application screenshots are shown on the Figure 78.



PUBLIC, SMARTSANTANDER PROJECT

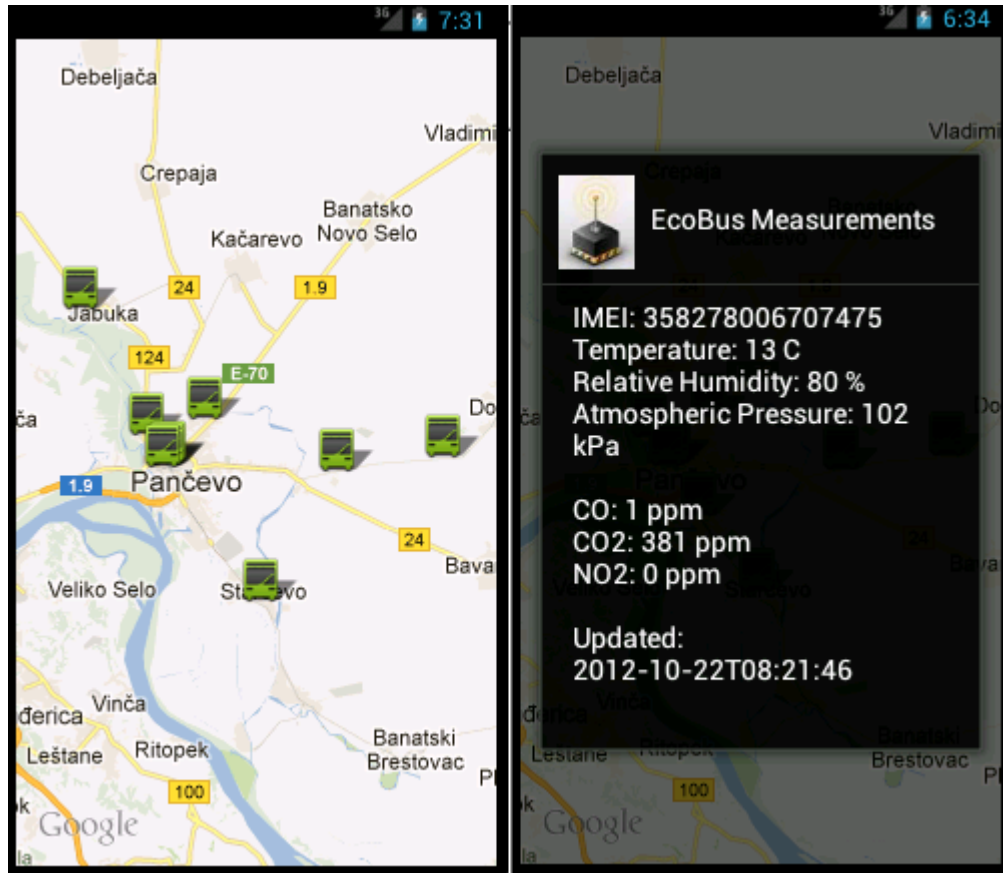


Figure 78. Public transportation monitoring application: Real-time bus location and environmental measurements

PUBLIC, SMARTSANTANDER PROJECT

4.7. SMART METERING

The overall goal of this scenario and its related use cases is to explore the use of Internet of Things (IoT) -based solutions to manage user energy demand in an office environment. This scenario builds upon an instance of the SmartSantander testbed, which is currently deployed throughout the office space of the Centre for Communication System Research (CCSR) at the University of Surrey, Guildford and consists of 200 SmartMeters and 100 reprogrammable gateways. On top of this testbed instance, a system has been built to collect energy consumption information of all devices associated with a particular work desk and relate this information with a particular user. Apart from collecting metrics for the energy consumption of devices, the system is expected to also recognize the corresponding consumption context (such as employee presence at his work desk, meeting rooms occupancy, etc etc). Based on the collected information the following use cases has been developed:

- Bob is a researcher of CCSR. CCSR disposed a prize for the employee that is more energy efficient at his work place. This means that no energy should be wasted when an employee is not present at his work desk. This can be achieved for instance if Bob takes care of turning off his devices before leaving the office at the end of the day or if he uses sleep modes for them when not present for more than half an hour. In order to check his progress towards the prize Bob can register to a specific website for daily reports about its energy efficiency (energy consumed when present with respect to energy consumed when not present) to check the report on his mobile phone/laptop at the end of the day.
- Bob is the same employee willing to win the CCSR prizes for the most energy efficient employee at his workplace. Checking his daily feeds he realizes that he's not behaving very well with this respect, mainly because he forget often to turn off his LCD screen and desktop PC when he leaves the office. In order to be helped by the system toward this goal, he can subscribe for alert/reminder notification on his mobile phone when inefficiency at his desk is detected. If energy inefficiency is detected, presumably because Bob is in a meeting in a different room or out for lunch or he just left the work place, while his appliances are not in a sleep state, a notification indicating this misbehaviour is sent to his mobile phone together with the possibility to remotely turning off its appliances and verify their corresponding energy consumption.

PUBLIC, SMARTSANTANDER PROJECT

Beyond the scope of this specific use case, different stakeholders/users as well as different service providers can be identified and different information flows can be derived depending on their different perspectives. Below, the specific way the system interacts with the users depending on their perspective, is presented.

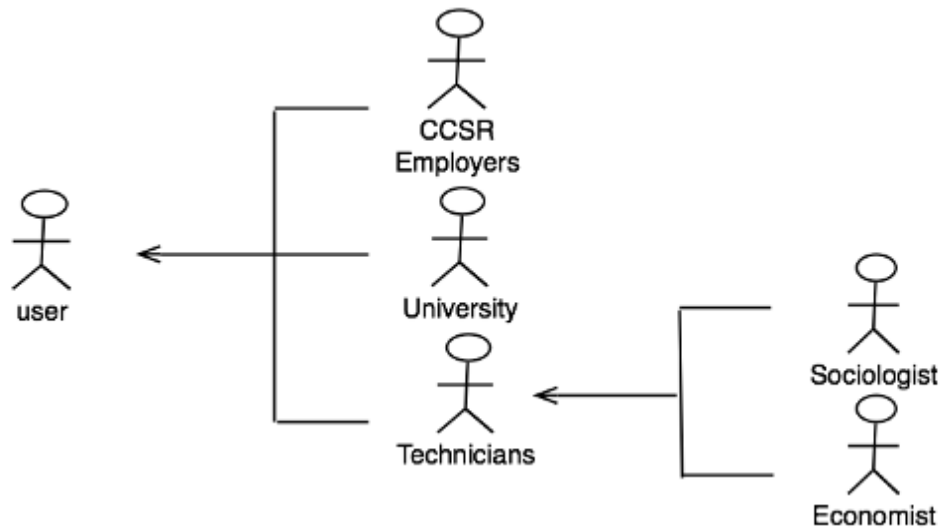


Figure 79: SmartMetering – Different involved actors

Provided services/Users perspective

- User perspective (CCSR employers)
 - The user can know and visualize measurements about the energy consumption produced by its desk along with context information regarding its presence at its working desk. It can also receive automated feedback in case of misbehaving is detected to which it can respond with actuation requests.

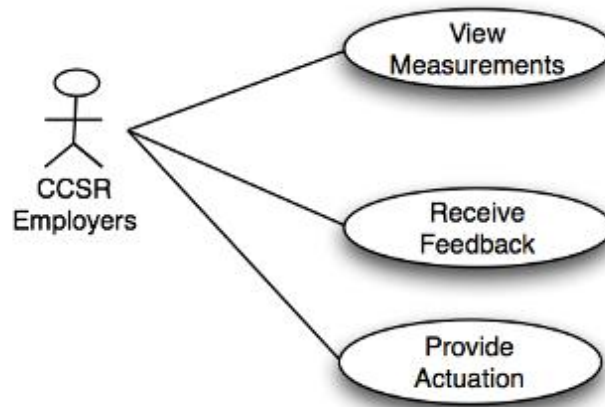


Figure 80: SmartMetering – Use Case Diagram user perspective

- Authorities perspective (University)
 - From the university point of view, the provider controls and monitors all source of wastefulness, being able to react in a quick way to these circumstances.

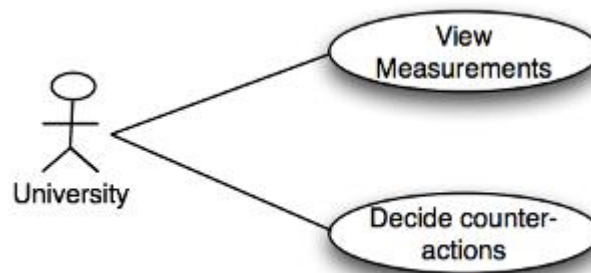


Figure 81: SmartMetering – Use Case Diagram authorities perspective

- Technical user perspective (economist, sociologist)
 - Sociologist can detect social behaviour that lead to energy wastage. Based on that they can propose some persuading strategies in order to correct these misbehaviours.
 - Economist can estimate the amount of money saved with the application of the proposed strategies with respect to when these strategies are not in place.

PUBLIC, SMARTSANTANDER PROJECT

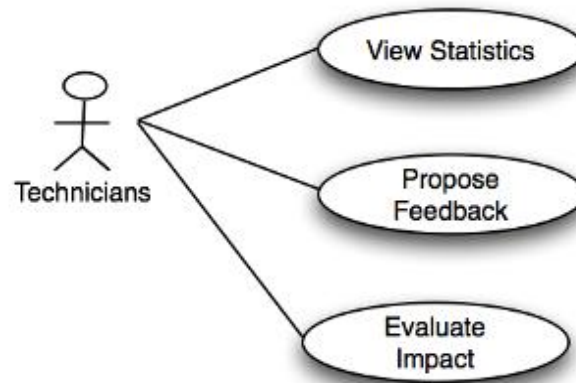


Figure 82: SmartMetering – Use Case Diagram technical user perspective

Services provisioning/Service provider perspective

In order to realize the envisioned system and related services, the following requirements need to be fulfilled from a service provider perspective:

- In order to achieve the proposed results, each desk and common areas at the CCSR building should be monitored by a smart sensor node, able to collect information about the energy consumed by all the appliance connected to it, along with information useful for understanding the contextual environment, such as presence (PIR), noise level (microphone), light, temperature etc;
- Moreover the smart sensor node should be also able to provide some actuation mechanism, such as the possibility of remote switching appliances;
- Furthermore provision for various user interaction channels has to be made, which will exploit the availability of user terminals, such as work PC or mobile phone of the employee;
- By maintaining a certain degree of privacy and anonymity, a link between a given desk and its assigned CCSR employer should be provided.

With respect to the specific user perspective considered in this specific scenario and represented by the CCSR employee, there is the need to address the following non-functional requirements:

- There needs to be an initial willingness from the user for behavioural change towards an energy conscious way. Otherwise he may mistrust the system and the user interventions may not be perceived as motivational;

PUBLIC, SMARTSANTANDER PROJECT

- A user may still have the suspicion that information captured could be exploited outside the experiment for learning about his behavioural patterns at the work place and using this information in some way to assess his performance. Such abuse has to be avoided;
- Therefore each user should be always maintained in the loop during the data collection phase. For this reason the possibility for him to decide when to share his data should be allowed.

4.7.1. SYSTEM BEHAVIOUR

For this scenario, different software components have been developed in order to generate, collect and store the information about the energy consumption/presence at work desk/office for each involved user from on side. From the other side these components have been be paired with additional ones able to extract knowledge from this collected information (i.e., occurrence of situation of energy inefficiency), visualize it and expose it to the user through adequate feedback mechanisms and visualization tools, such as Web-based GUI dashboard and notification systems for delivering information to the user owned mobile phone.

The envisioned components required to realise this service are illustrated in **Error! Reference source not found..**

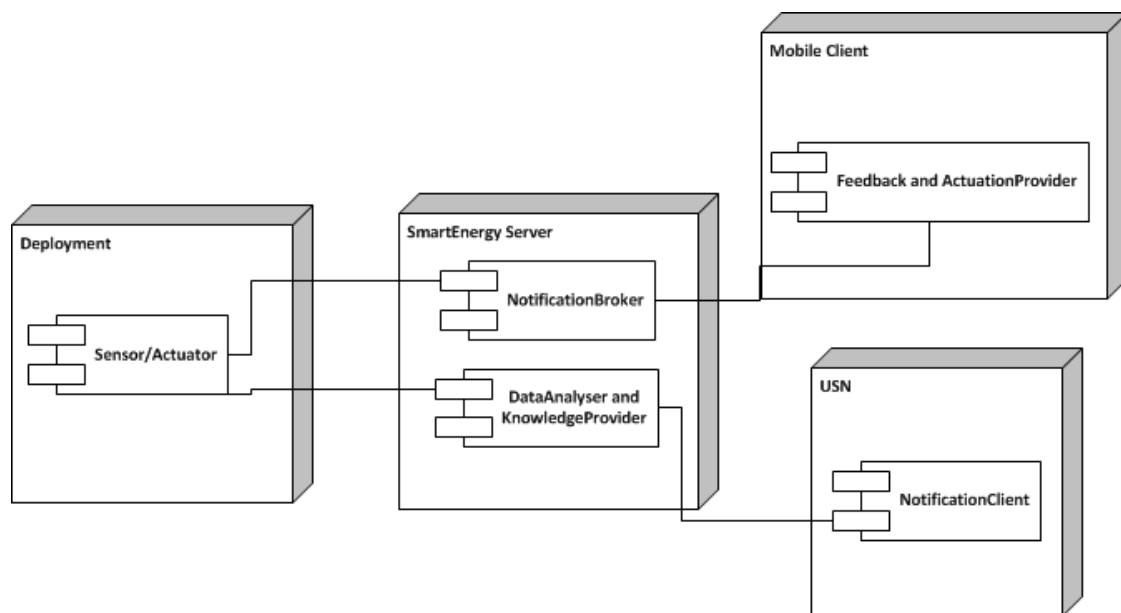


Figure 83 Architecture components

The role of each specific component is as follows:

- SmartSantander deployment at CCSR: it refers to the set of tools developed at both node side for generating the user specific information and at server and infrastructure side for collecting this

PUBLIC, SMARTSANTANDER PROJECT

information and interacts with specific nodes using the Testbed Runtime SmartSantander infrastructure;

- Information storage: it refers to the module providing integration and able to redirect user specific generated information for being stored into the USN platform provided by the SmartSantander architecture and in a way subject to user approval and decision (this in order to fulfil the non-functional requirement of user in the loop in the data sharing process);
- SmartEnergy Server:
 - Data Analyzer and Knowledge Provider: it refers to the set of tools developed for real-time analysis of the generated data and identification of situation of energy inefficiency and for periodic generation of aggregated reports and their more detailed breakdown about the specific user energy wastage. Within this category falls also web-tools (based on Google Web Toolkit and MySQL) developed for allowing the user to register and unregister from the service and for detailed inspection of his related data/reports;
 - Notification broker: it refers to the infrastructure developed for sending notifications to the user owned mobile phones based on the observed situation inefficiency detected by the Data Analyzer and Knowledge provider and for collecting feedback and actuation action from the user (i.e., remote switching of appliances) and report them back to the system.
 - Feedback/Actuation provider (named SmartEnergy app): it refers to the mobile application implemented on the user owned mobile phone and able to provide the user with feedback about its energy consumption and inefficiency, generate remote actuation and control the information flow and the share of testbed generated data related to him. This tools consist of Android based application and back end server both relaying on lightweight MQTT pub/sub technology.

Following the main functionalities provided by the implemented services and the interaction with the architecture components are reported.

Error! Reference source not found. shows the interaction among the *Feedback/Actuation Provider* running on the user mobile phone, the *Notification Broker* and the *Information Storage (USN)* in order to register to the service, subscribe for a given feedback and the creation of the corresponding topic in the *Notification Broker* and for allowing data sharing with the *Information Storage (USN)*. In the specific case the considered topic is the identification of energy inefficiency at the user desk (i.e., energy is consumed when the user is not present).

PUBLIC, SMARTSANTANDER PROJECT

The following steps are performed:

- User subscribe through the *Feedback Provider* to the *Notification Broker* for topic about its energy inefficiency;
- The *Notification Broker* sends confirmation about the required registration;
- At the same time, the *Notification Broker* registers the user related SmartMeter and its corresponding testbed gateway to the *USN*;
- The *USN* confirms the node registration and it is now ready to accept and store user related data from the corresponding SmartMeter sent to it by the *Data Analyzer*.

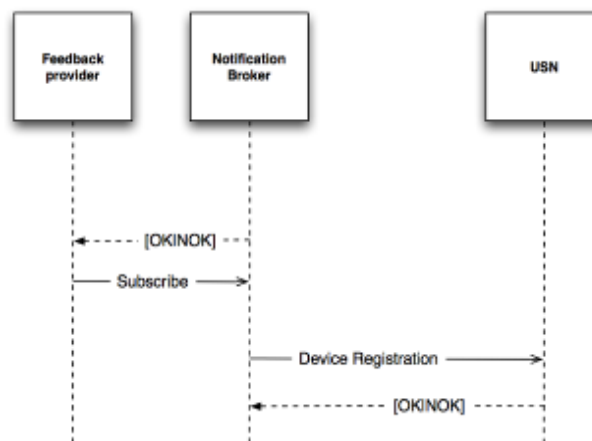


Figure 84 Sequence Diagram: Subscription

Figure 85 shows a very simple interaction between the system and the *Feedback/Actuation Provider* running on user mobile phone in order to deliver to the user the required notification about its energy consumption.

The following steps are performed:

- The *Data Analyzer* receives data from the testbed and extract user related information such as its related devices energy consumption and identify situation of energy inefficiency;
- The *Notification Broker* receives data and in case of corresponding user subscription it deliver notification to the *Feedback provider* module running on the user mobile phone;
- The *Feedback Provider* receives notification and provides feedback to the user (such as notification about its energy inefficiency or user related devices energy consumption) and confirms the reception of the notification.

PUBLIC, SMARTSANTANDER PROJECT

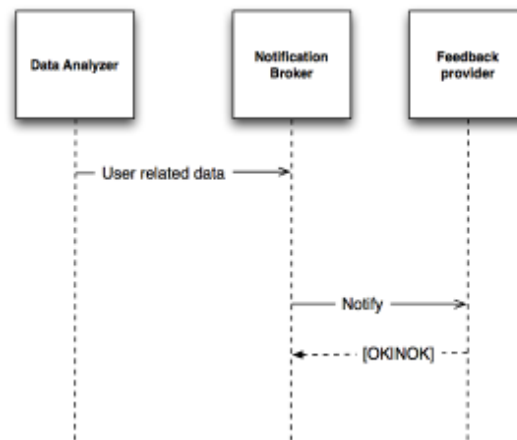


Figure 85: Sequence Diagram: Notification

Finally, **Error! Reference source not found.** hows the interaction between each *SmartMeter* composing the *SmartSantander testbed* at CCSR for reporting data to the *Data Analyzer and Knowledge Provider*, generating notification and storing the observed data in the USN in the form of *Observation&Measurement* entry. The content of the information reported by each *SmartMeter (observationData)* is the following:

- timestamp
- SmartMeterLocation
- PIR
- noiseLevel
- temperature
- light
- vibration
- powerConsumption
- other appliance energy profiling information

The following steps are performed:

- The SmartSantander deployment at CCSR collects information from the user associated SmartMeter;
- The corresponding *userData* are delivered to the *Data Analyzer*;
- The *Data Analyzer* translates the received information into SensorML format and in case of the user and its associated SmartMeter have been register to share data, it provides this information to the *USN* through the *insertObservation* procedure;
- The *USN* receive the user related data and store it, confirming the reception to the *Data Analyzer*.

PUBLIC, SMARTSANTANDER PROJECT

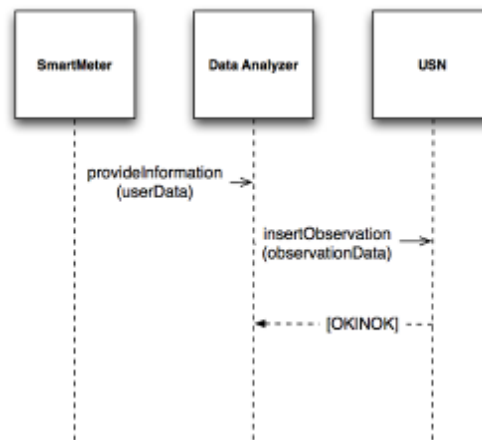


Figure 86 Sequence Diagram: Sharing and Storage

Because the SmartMeters report raw data that do not follow a SensorML model, an adaptation layer is provided by the Data Analyzer in order to create a SensorML representation of each received data before storing them into the *USN* subsystem. The relevant information provided to the *USN* is the following:

- UserIdentifier
- SmartMeterLocation
- observationPeriod
- presence
- energyConsumption

From this data the Data Analyzer extracts also the required knowledge needed for triggering the notification system.

4.7.2. SmartMetering applications

The following section provides a tutorial on how the user interacts with the SmartMetering system in place at the SmartSantander deployment at CCSR.

The following initial steps and action are performed.

- First of all the user is required to register to the system through a web interface;
- After the registration process is completed the user receive a unique key used by the system for relate the specific user to its associated *SmartMeter* and the corresponding generated data. Together with the key a unique web link is also provided to the user;
- The user can utilize the unique key for configuring a gadget running on its laptop or for accessing its own web dashboard, showing the breakdown of its personal energy consumption at its work desk.



Figure 87 MyEcoFootprint Gadget and Dashboard

Error! Reference source not found. shows in clockwise order the gadget running on the user personal laptop providing indication on how good its is behaving with respect to achieving its target to be energy efficient, where:

- Green means good progress toward the target;
- Amber means average progress toward the target;
- Red means very bad progress toward the target.

The remain figures shows similar information through a web based GUI and also additional information about the breakdown of user energy consumption towards its target, and with respect to other people willing to achieve the same target. All the information is provided using different aggregation metrics, such as by hour, by day and by week.

Other than accessing its energy related data, the user can also subscribe for a mobile service through the SmartEnergy app in order to get notification on its mobile about its energy consumption, its energy inefficiency and eventually remotely turning off its related devices in order to improve its energy efficiency, by reducing the energy wasted when not present at its work desk. The following steps are required:

PUBLIC, SMARTSANTANDER PROJECT

- Through the *Feedback/Actuation Provider* the feedback can register to the *Notification Broker* for Energy inefficiency notification and user device real-time energy consumption notification using its own secret key. This can be done using the “Settings menu” of the application (**Figure 88-a**);
- The user starts then the application using the “Start” button in the main application window (**Figure 88-b**);
- The *Notification Broker* then communicates to the *Data Analyzer* to extract information about the specific user energy inefficiency and to forward to it and to *USN* its related device energy consumption information;
- In case the user leaves its desk for a given amount of time while its devices are still draining power, the system detects energy inefficiency situations and it triggers a notification to the user mobile phone (**Figure 88-c**).
- In order to improve its energy inefficiency, the user can then decide if it wants to remotely turn off its appliances;

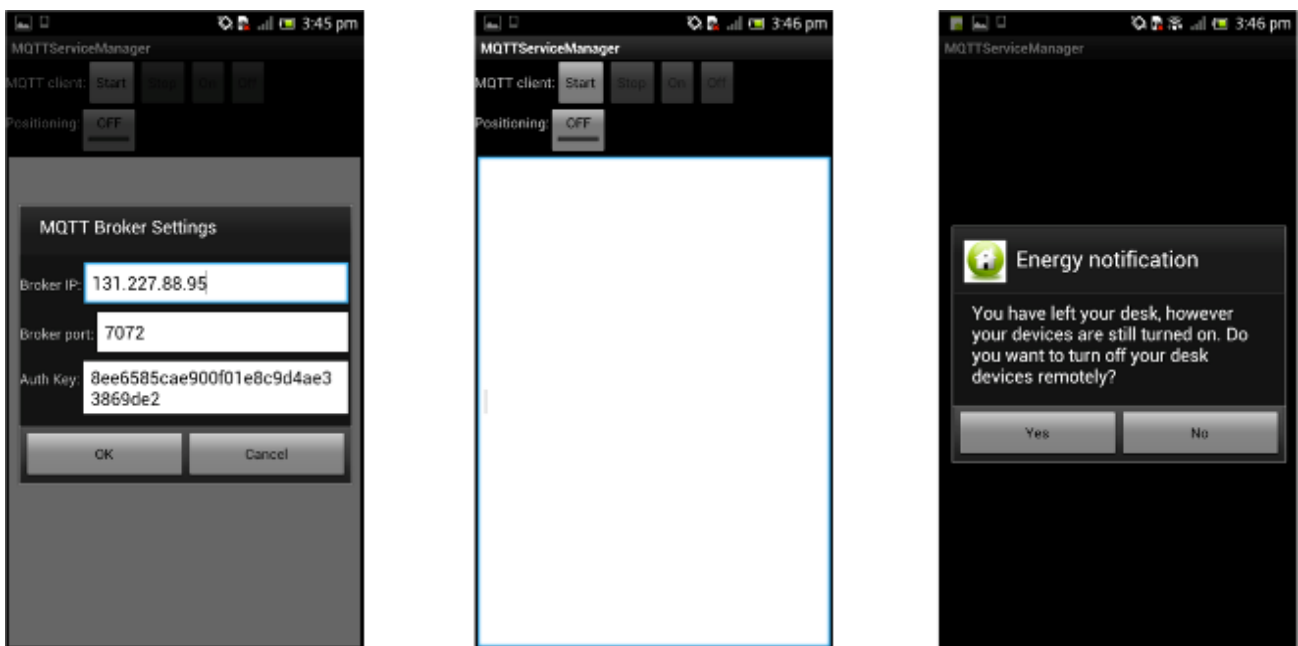


Figure 88 SmartEnergy App: Registration and inefficiency notification

- Depending on its choice the user will receive different real-time information about its device energy consumption. Such information can look in two different ways:

PUBLIC, SMARTSANTANDER PROJECT

- If not actuation is required the consumption will remain at high level and other energy inefficiency notification will be received for a fixed amount of time (**Error! Reference source not found.-a**);
- If actuation is required the consumption will drop and no further energy inefficiency notification will be received if the condition are not met again (**Error! Reference source not found.-b**);
- On the other side, in case the user wants to proactively turning on its device when it is supposed to come back at his work desk (such as its LCD screen), because this won't affect its energy efficiency but it will instead improve its productivity (because it does not have to wait for some set up time), it can do using the *Feedback/Actuation Provider Control* feature:
 - This will allow the user to come back to the main application window and select the "On" button;
 - An actuation request will be then sent to the *Notification Broker* and from there a turning on command will be sent to the corresponding SmartMeter by the SmartSantander deployment tools that have previously subscribed for such kind of notifications;
 - The user will start to observe that energy consumption of its related devices will start to increase again (**Error! Reference source not found.-c**).

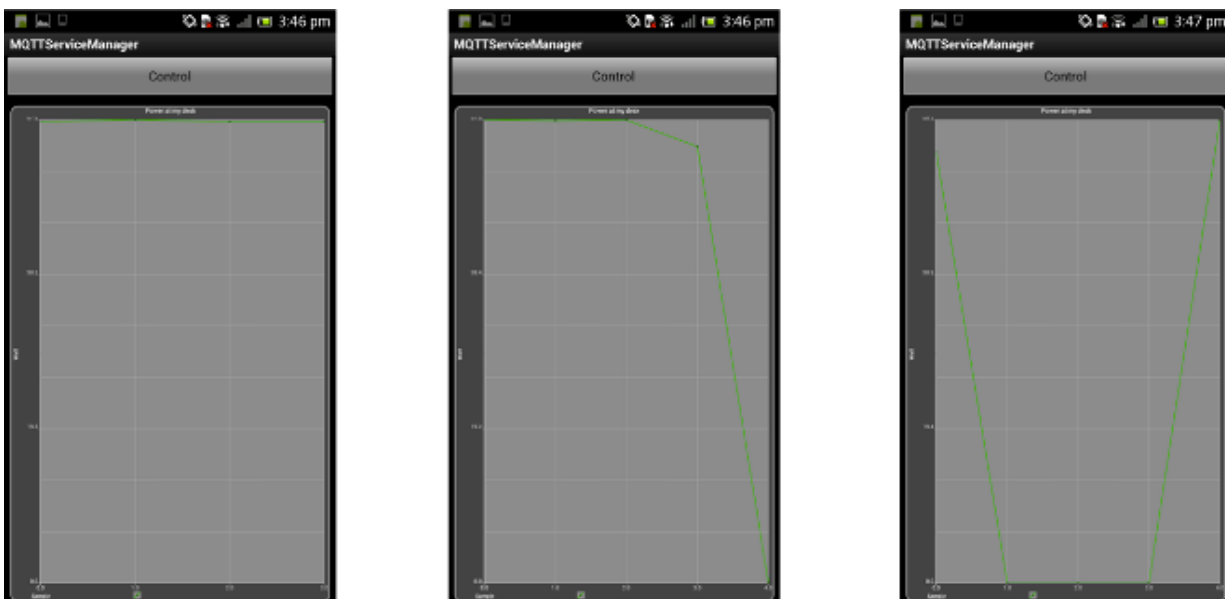


Figure 89 SmartEnergy App: Real-time energy consumption notification

5. REALISATION OF SERVICES ON TOP OF SMARTSANTANDER PLATFORM

In this section we describe in detail the tools and software components available by the SmartSantander platform that the service developer can use in order to develop his service on top of it. Namely the USN component to which services can connect to via web service interfaces, in order to retrieve sensory information of the SmartSantander test beds and two repositories: one for participatory sensing applications and another for augmented reality. Both repositories provide important functionalities for these services and applications that connect them to the SmartSantander platform.

5.1. USN: WEB SERVICE INTERFACES

For all the use cases described in previous paragraphs, USN node provides the storage of the information sent by the devices and it notifies this information to the related subscribed applications.

Once devices have been registered and they are sending information to USN platform (and it is stored in its database), interested applications can access these data using two different approaches:

- Make use of the Publish-Subscribe-Notify paradigm
- Two different APIS (M2M REST API and M2M SOAP API) are exposed by USN platform using two HTTP protocols: HTTP/SOAP and HTTP/REST.

To interchange information with the rest of participant nodes, USN offers several interfaces:

- An interface to receive the information sent by the devices. This interface is based on HTTP protocol, using SensorAPI protocol to describe the structure of the device (register message) or the generated measures (O&M message). This protocol follows the OGC standard. An optimized version of SensorAPI (LightSensorAPI) is also available to send the same information in a more compress way.
- A web service interface to interact with the service applications. This interface includes the next operations:
 - a) **Subscription.** The service applications use this operation to report USN that they are interested in receiving information from some devices and the URL where they want to be notified when this information were received by USN.

**PUBLIC, SMARTSANTANDER PROJECT**

- b) **Notification.** This operation is used by USN to send the information generated by any device to a service application that has been subscribed using the previous operation.
- c) **Queries.** With this operation, USN offers the capability to the service applications to gather information from USN data base in any moment without using the subscribe-notify mechanism.

5.1.1. Protocols used by USN Gateway**5.1.1.1. SensorML & O&M (Observations and Measurements)**

SensorML provides a common model for the description of devices and systems (resources), easily adaptable to any implementation needs. Additionally, it allows describing characteristics and capacities of the resources, parameters, location information, etc.

It is a standard defined by the Open Geospatial Consortium (OGC).

SensorML is a description model to specify the features and capabilities of the means by which sensor systems or processes can make themselves known and discoverable. SensorML provides a rich collection of metadata that can be mined and used for discovery of sensor systems and observation processes. This metadata includes identifiers, classifiers, constraints (time, legal, and security), capabilities, characteristics, contacts, and references, in addition to inputs, outputs, parameters, and system location.

Observation and Measurements (O&M) was defined also by OGC. O&M is a model to represent measurements coming from devices. It covers simple and complex models to describe these observations.

5.1.1.2. Light Sensor Protocol

Keeping This protocol preserves all the syntactic description capabilities of both SensorML and O&M, this but it reduces protocol decreases the size of the messages transmitted between USN and devices using an algorithm to codify SensorML and O&M tags to obtain a shorter one..

5.1.1.3. Ultralight Sensor Protocol

This protocol (An even lighter version of the Light Sensor protocol) is associated to with constrained environments (for instance, a sensor with a very low bandwidth to send the measures) where the total information transmitted from the devices to USN must be kept to the minimum.

5.1.1.4. Registration request

It is used to describe resources and register them on the platform. Registration can be done either with SensorAPI or with the Light Sensor protocol.

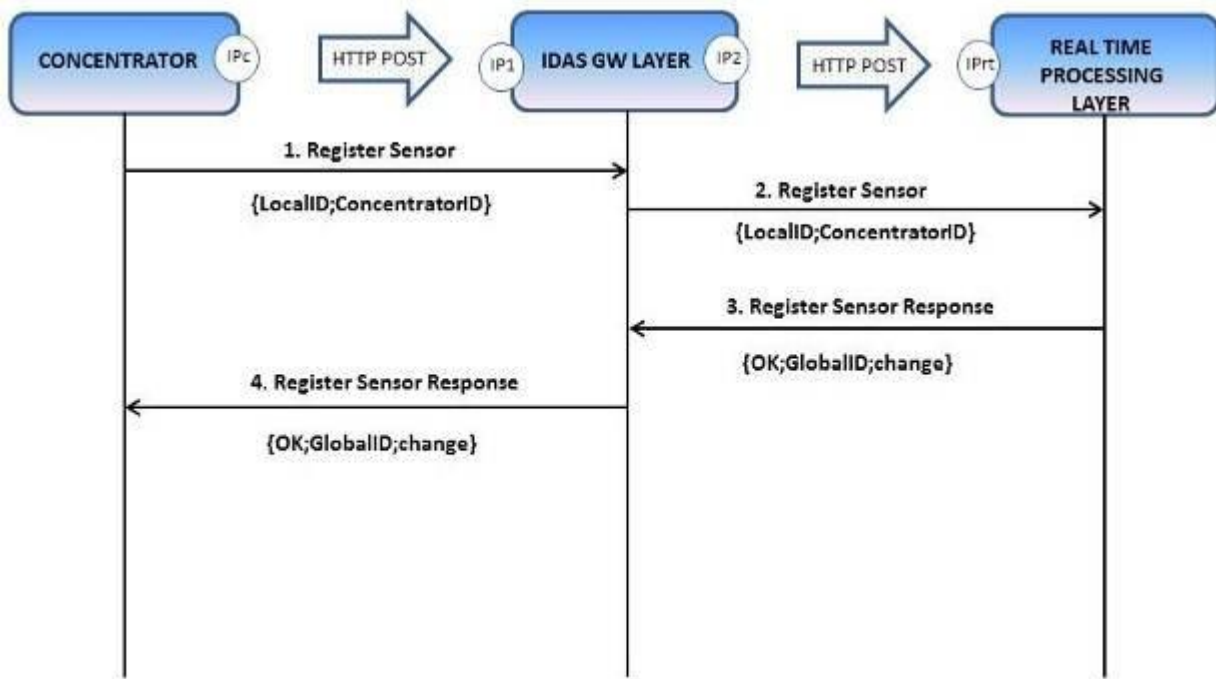


Figure 90: Sensor rRegistration flow

1. Through the HTTP POST, hub the IoT gateway sends an XML RegisterSensor document following the instructions in this document. It checks the request and the resource description included in the XML RegisterSensor document.
2. It responds to the hub with an XML-based RegisterSensorResponse or ExceptionReport document, depending on whether the request was accepted or not.
3. If the request is accepted by the gateway it starts the procedure of sending the XML RegisterSensor XML document adapted to the model defined in the platform.
4. In the response answer of the Sensor Enabler, a new name for the sensor can be sent to the IoT gateway in order to be requested the forwarding of the request to update the named resource identifier that has to be sent in the measure messages.. The gateway will perform the change of the name assigned to resources on the platform and try to log again in the Sensor Enabler.

5.1.1.5. Measure notification request

This request is used to inform the platform about the collection of an observation or measurement. You can perform use both SensorAPI, Light Sensor or Ultralight as lightweight protocol to perform this task..

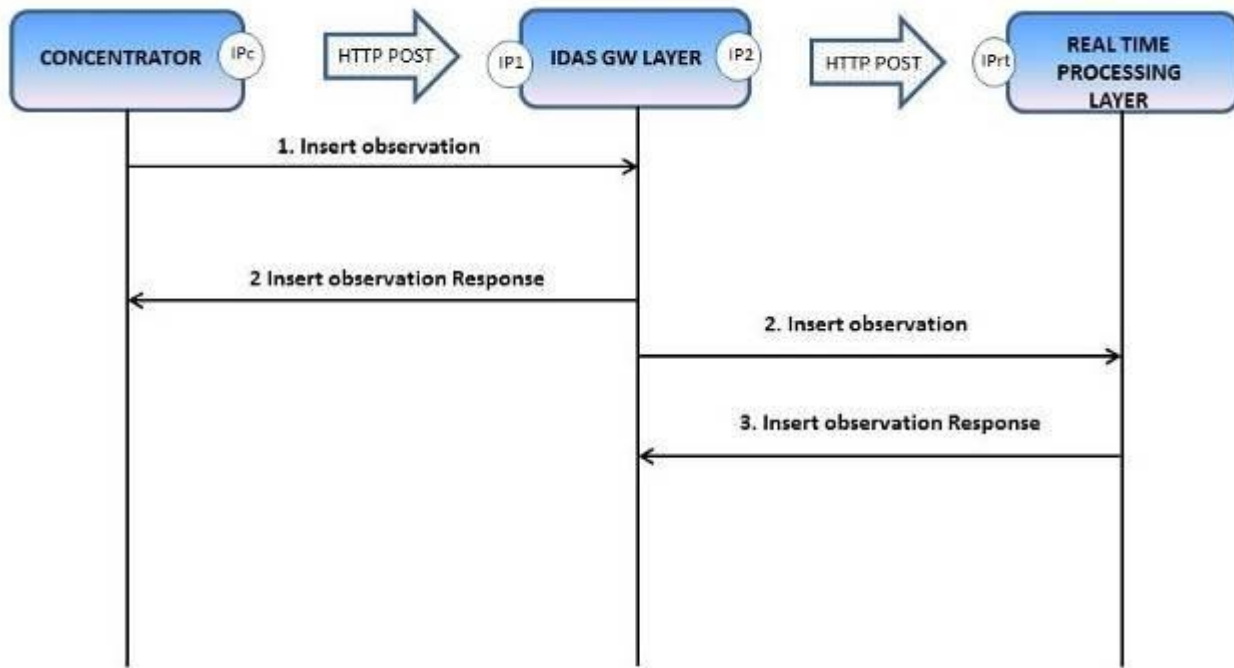


Figure 91: Measure flow

1. The hub IoT Gateway sends via an HTTP POST request an XML document that follows InsertObservation the instructions in this document.
2. If the resource identified in this document as the source of the measure hasn't been previously registered as a resource, the request with a XML ExceptionReport document will be rejected. If there is a request that does not pass parsing/semantic it will also be rejected. If accepted, it responds with an InsertObservationResponse document.
3. The InsertObservation document adapted to the needs of the sensor platform is sent to Enabler only if the analysis phase is successful.

5.1.1.6. Examples using SensorAPI

In the following examples, the *curl* command has been used It will be used *curl* command to simulate register and sending of measurements.

The following steps have to me be made:

1. Previous data Provision configuration before doing the tests elements
 - Provision a service (Defining it type: Immediate delivery, retention till aggregated data or retention till activation)

PUBLIC, SMARTSANTANDER PROJECT

- Provision client's application.
 - Provision a concentrator (In "Concent. Univ." section).
2. Register the simulated sensor into USN.
3. Fill "register_test.req.xml" template described in the next page with the following information:

- Device/sensor "serialNumber" to be registered.

```
<sml:Term definition="urn:x-ogc:def:identifier:IDAS:1.0:serialNumber">  
<sml:value>00:21:9B:51:8F:BA</sml:value>
```

- Provided concentrator "UniversalIdentifierOfLogicalHub".

```
<sml:Term definition="urn:x-ogc:  
def:identifier:IDAS:1.0:UniversalIdentifierOfLogicalHub">  
<sml:value>parkSmartS0000001</sml:value>
```

- Send the "xml" file to the gateway using the following command:

```
curl --dump-header - --output - --data @register_Test.req.xml  
http://IPNAT:8002/idas/sml
```

IPNAT is the IP address where at which USN's GW is listening.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <sos:RegisterSensor service="SOS" version="1.0.0"  
xsi:schemaLocation="http://www.opengis.net/sos/1.0 sosRegisterSensor.xsd"  
xmlns:swe="http://www.opengis.net/swe/1.0.1" xmlns:gml="http://www.opengis.net/gml"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink"  
xmlns:om="http://www.opengis.net/om/1.0" xmlns:sml="http://www.opengis.net/sensorML/1.0.1"  
xmlns:sos="http://www.opengis.net/sos/1.0" >  
<sos:SensorDescription>  
<sml:System>  
<sml:identification>  
<sml:IdentifierList>  
<sml:identifier>  
<sml:Term definition="urn:x-ogc:def:identifier:IDAS:1.0:serialNumber">  
<sml:value>00:21:9B:51:8F:BA</sml:value>  
</sml:Term>  
</sml:identifier> <sml:identifier>  
<sml:Term definition="urn:x-ogc:def:identifier:IDAS:1.0:UniversalIdentifierOfLogicalHub">  
<sml:value>parkSmartS0000001</sml:value>  
</sml:Term>  
</sml:identifier>  
</sml:IdentifierList>  
</sml:identification> <sml:classification>  
<sml:ClassifierList>  
<sml:classifier>  
<sml:Term definition="urn:x-ogc:def:classifier:IDAS:1.0:system">  
<sml:value>system00001</sml:value>  
</sml:Term>  
</sml:classifier>  
</sml:ClassifierList>  
</sml:classification>
```

PUBLIC, SMARTSANTANDER PROJECT

```

<sml:inputs>
<sml:InputList>
<sml:input name="Potencia Instantanea">
<swe:ObservableProperty definition="urn:x-ogc:def:phenomenon:IDAS:1.0:power"/>
</sml:input>
</sml:InputList>
</sml:inputs>
<sml:outputs>
<sml:OutputList>
<sml:output name="Potencia Instantanea">
<swe:Quantity definition="urn:x-ogc:def:phenomenon:IDAS:1.0:power">
<swe:uom xlink:href="urn:x-ogc:def:uom:IDAS:1.0:kilowatt"/>
</swe:Quantity>
</sml:output>
</sml:OutputList>
</sml:outputs> <sml:parameters>
<sml:ParameterList>
<sml:parameter name="Command URL" xlink:arcrole="urn:x-ogc:def:property:IDAS:1.0:read" xlink:href="urn:x-ogc:def:property:IDAS:1.0:commandURL" xlink:role="urn:x-ogc:def:property:IDAS:1.0:operationProperty">
<swe:Text>
<swe:value>http://127.0.0.1:8080/CommandParser</swe:value>
</swe:Text>
</sml:parameter>
</sml:ParameterList>
</sml:parameters>
<!--<sml:components/--> </sml:System>
</sos:SensorDescription> <sos:ObservationTemplate>
<om:Observation >
<om:samplingTime> </om:samplingTime> <om:resultTime> </om:resultTime> <om:procedure /> <om:observedProperty /> <om:featureOfInterest /> <om:result></om:result>
</om:Observation>
</sos:ObservationTemplate>
</sos:RegisterSensor>

```

Figure 92: register_test.req.xml templateRegister example using SensorML

4. Simulate sending of measurements from a device.

5. Fill "observation_test.req.xml" template with the following information:

- Device/sensor "serialNumber" simulating the sending of information.

```
<sos:AssignedSensorId>00:21:9B:51:8F:BA</sos:AssignedSensorId> y
```

```
<om:procedure xlink:href="00:21:9B:51:8F:BA" />
```

- Provided former provided concentrator

```
<om:parameter xlink:href="urn:x-
```

```
ogc:def:identifier:IDAS:1.0:UniversalIdentifierOfLogicalHub">
```

PUBLIC, SMARTSANTANDER PROJECT

```
<swe:Text>
<swe:value>parkSmartS0000001</swe:value>
```

- Send the “xml” file to the gateway using the following command:

```
curl --dump-header - --output - --data @observation_Test.req.xml
http://IPNAT:8002/idas/sml
```

IPNAT is the IP address where at which USN’s GW is listening

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<sos:InsertObservation service="SOS" version="1.0.0" xsi:schemaLocation="http://www.opengis.net/sos/1.0
sosInsert.xsd" xmlns:swe="http://www.opengis.net/swe/1.0.1" xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:om="http://www.opengis.net/om/1.0" xmlns:sos="http://www.opengis.net/sos/1.0" >
<sos:AssignedSensorId>00:21:9B:51:8F:BA</sos:AssignedSensorId>
<om:Observation >
  <om:samplingTime>
    <gml:TimeInstant>
      <gml:timePosition frame="urn:x-ogc:def:trs:IDAS:1.0:ISO8601" >2011-04-
25T11:07:02Z</gml:timePosition>
    </gml:TimeInstant>
  </om:samplingTime> <om:procedure xlink:href="00:21:9B:51:8F:BA" /> <om:observedProperty xlink:href="urn:x-
ogc:def:phenomenon:IDAS:1.0:power" /> <om:featureOfInterest />
  <om:parameter xlink:href="urn:x-ogc:def:identifier:IDAS:1.0:UniversalIdentifierOfLogicalHub" >
    <swe:Text>
      <swe:value>parkSmartS0000001</swe:value>
    </swe:Text>
  </om:parameter>
  <om:result>
    <swe:Quantity definition="urn:x-ogc:def:phenomenon:IDAS:1.0:power" >
      <swe:uom xlink:href="urn:x-ogc:def:uom:IDAS:1.0:kilowatt" /> <swe:value>10.525</swe:value>
    </swe:Quantity>
  </om:result>
</om:Observation>
</sos:InsertObservation>
```

Figure 93: observation_test.req.xml” templateObservation example using SensorML

5.1.1.7. Using LightSensorAPI& UltraLightSensorAPI

1. Provision Previous data configuration before doing the tests elements
 - Provision a service (Either “Instant measures delivery service”, “not deliver till business date not defined” or “not deliver till device activation”).
 - Provision client’s application.
 - Provision a concentrator (In "Concent. Univ." section).
2. Register the simulated sensor into USN.
3. Fill “register_test.req.xml” template with the following information:

PUBLIC, SMARTSANTANDER PROJECT

- Device/sensor "serialNumber" to be registered.

```
<id href="1:2">12345678901234</id>
```

- Provided concentrator "UniversalIdentifierOfLogicalHub".

```
<id href="1:7">" UniversalIdentifierOfLogicalHub "</id>
```

- Send the "xml" file to the gateway using the following command:

```
curl --dump-header - --output - --data @register_Test.req.xml  
http://IPNAT:8002/idas/2.0
```

IPNAT is the IP address where at which USN's GW is listening

```
<rs>  
  <id href="1:1">MyTestService</id>  
  <id href="1:2">12345678901234</id>  
  <id href="1:5">MANUFACTURER</id>  
  <id href="1:7"> UniversalIdentifierOfLogicalHub </id>  
  <what href="8:19"/>  
  <param name="GPS" id="gps" role="3:13" arcr="3:10" href="8:27">  
    <gps lat="0.0" lon="0.0"/></param>  
  <data name="acceleration" id="acceleration"><quan uom="9:28">0</quan> </data>  
</rs>
```

Figure 94: register_test.req.xml" template Register example using LightSensorAPI

4. Simulate sending of measurements from a device.
5. Fill "observation_test.req.xml" template with the following information:

- Device/sensor "serialNumber" to be registered.

```
<id href="1:2">12345678901234</id>
```

- Provided concentrator "UniversalIdentifierOfLogicalHub".

```
<id href="1:7"> UniversalIdentifierOfLogicalHub "</id>
```

- Send the "xml" file to the gateway using the following command:

```
curl --dump-header - --output - --data @obserTest.xml  
http://IPNAT:8002/idas/2.0
```

IPNAT is the IP address where at which USN's GW is listening

PUBLIC, SMARTSANTANDER PROJECT

```
<io>
  <obs from=" MyTestService ">
    <stm>2011-10-18T12:08:09Z</stm>
    <what href="8:19"/>
    <param href="1:7"><text> UniversalIdentifierOfLogicalHub </text></param>
    <param><gps lat="43.0" lon="2.21"/></param>
    <data name="acceleration" id="acceleration"><quan uom="9:28">2</quan></data>
  </obs>
</io>
```

Figure 95: observation_test.req.xml" templateObservation example u using LightSensorAPI

M2M SOAP API

Communications to M2M SOAP API is made using HTTP protocol and exchanging SOAP messages.

SOAP (Wikipedia op. cit.), originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) for transport..

It exposes the following M2M service capabilities:

1. The capability for an application to subscribe to M2M network/service events. Two types of subscriptions are considered in terms of the nature of the event:
 - a. subscription to the registration events (new M2M user devices register on the M2M network)
 - b. subscription to observation events (measurements) of M2M user devices (registered M2M user devices publish observations)
2. The capability for an application to be notified of events from the M2M network/service,
3. The capability for the application to get data of the underlying M2M network/service (get concentrators data, devices data, stored measurements data, etc.).

5.1.1.8. Detailed functional description

Subscription (subscribe, unsubscribe, disconnect)

This functional block consists of the following operations:

- **Subscribe:** The client application subscribes to events. Two kinds of events are possible: Register events and Observation (ie. measurements) events. On success it receives (asynchronously) a subscription response with an assigned connection ID.
- **Unsubscribe:** The client application unsubscribes from a given subscription.

PUBLIC, SMARTSANTANDER PROJECT

- **Disconnect:** The client application signals that no further notifications should be received for a given connection ID.

The core of the subscription primitive is implemented by the Xpath element. In that element, Xpath expressions must be included. XPath is a language to perform some operation on an XML message and it is used in the Subscribe primitive to define the trigger condition for the subscription. Those trigger conditions can be very simple or more complexes, but they can be summarized in the following options:

- Event that triggers the notifications (Register/Observation)
- Specific element (concentrator/device) from which the event must come (the target resource of the subscription)
- Additional conditions/thresholds over the data the specific resource.

Some examples that could be used as a trigger condition are the following XPath expressions:

- ***//sos:RegisterSensor***

With this XPath, the application will be notified every time that the sensor networks send a register SensorML message.

- ***//sos:RegisterSensor/sos:SensorDescription/sml:System/sml:identification/sml:IdentifierList/sml:identifier/sml:Term[@definition="urn:x-ogc:def:identifier:IDAS:1.0:UniversalIdentifierOfLogicalHub"]/sml:value="parkSmartS0000001"***

With this XPath, the application will be notified every time that the sensor networks send a register SensorML message from a sensor which has a specific UniversalIdentifierOfLogicalHub ("parkSmartS0000001").

- ***//sos:InsertObservation***

With this XPath, the application will be notified every time that the sensor networks send a measure O&M message.

- ***//sos:InsertObservation/om:Observation/om:parameter[@xlink:href="urn:x-ogc:def:identifier:IDAS:1.0:UniversalIdentifierOfLogicalHub"]/swe:Text/swe:value="parkSmartS0000001"***

It is important to highlight and understand at least which should be the identity that must be used for precisely stating the subscription targeted resource. The answer is quite simple in any case: it must be a



PUBLIC, SMARTSANTANDER PROJECT

Logical Name previously registered in the USN database under as a identifier related to which actual data from a sensor which is going to send measures to USNa physical source is received and stored by the platform. . That is, only when the subscription uses the same name as the one given for USN to refer to a source of data, it will be possible for the platform to correlate both legs: the subscription and the targeted data.

A possible java code snippet is shown here:

- **Subscription Request. USN API primitive: subscribe**

```
private int solicitaSuscripcionObservacionEsperaRespuesta(String idasID,  
    long correlator, ObservationDescription desc)  
    throws ResultException, Exception {  
    SubscriptionType ref = new SubscriptionType();  
    ref.setServiceLogicalName(fldServiceLogicalName);  
    ref.setClientAppLogicalName("MyEnergy");  
    ref.setEventKind(EventKindType.OBSERVATION);  
    ref.setNotifyURI(fldServidorNotificacion);  
    ref.setOriginal(true);  
    ref.setPriority(1);  
    ref.setXpath(generaXpathObservacionPorFenomeno(desc.observationID));  
    ref.setSubscriptionLogicalName(String.valueOf(correlator));  
    ref.setTimed(false);  
    ref.setSeconds(0);
```



PUBLIC, SMARTSANTANDER PROJECT

```
try {
    int result = stSubscriptionPort.subscribe(fldSubscriptionResponse, ref);
    if (result == 0) {
    } else {
        throw new ResultException(200,
            "USN couldn't process the subscription");
    }
} catch (ClientException e) {
    throw new ResultException(300, "ClientException connecting to USN",
        e);
} catch (ServerException e) {
    throw new ResultException(300,
        "ServerException return from USN", e);
}
}
```

Notification (notification of events)

This functional block consists of the following operations:

- **Notification of Events:** The client application receives events. Possible event types are register and observation events

SensorData

Includes the following operations:

- **getDevicesByLogicalGroup**
- **getDataByService**
- **getDeviceData**
- **getConcentratorData**
- **addDataByService**
- **updateDataByService**
- **deleteDataByService**

The most important operation is `getDataByService` that is used by applications to query USN storage system.



PUBLIC, SMARTSANTANDER PROJECT

| Parameter | Mandatory/Optional | Description |
|--------------------------------|--------------------|---|
| serviceId | Mandatory | Service ID |
| choice of | Mandatory | |
| devices | Optional | Used to obtain devices list |
| concentrators | Optional | Used to obtain concentrators list |
| logicalGroups | Optional | Used to obtain logical Groups list |
| businessDevices | Optional | Used to obtain business devices list |
| serviceAttributes | Optional | Used to obtain service attributes list |
| accumulatedMeasurements | Optional | Used to obtain accumulated measurements by filter (See below) |

| Parameter | Mandatory/Optional | Description |
|--------------------------------|--------------------|---|
| AccumulatedMeasurements | | |
| businessDeviceAttribute | Mandatory | Business device attribute |
| name | Mandatory | businessDeviceAttribute name |
| value | Mandatory | businessDeviceAttribute value |
| filterDate | Mandatory | Date filter |
| from | Mandatory | From date |
| To | Mandatory | To date |
| dateAttributeName | Mandatory | Name of the date attribute to which this filter applies |
| interval | Mandatory | "hour" or "day" |

Figure 96: Parameters of the getDataByService operation

Through this API, the application could retrieve:

- The attributes related to a specific USN customer (i.e. Santander)
- Business information related to a specific USN customer. The business information is a set of values that the customer can associate to a specific sensor, for example, the owner of the sensor, the deployment address, etc.
- The list of sensor network concentrators related to a specific USN customer
- The description data of a specific sensor network concentrator
- The set of devices related to a specific USN customer
- The list of Logical Groups related to a specific USN customer. A logical group is an aggregation of sensor with some common features.
- The set of devices related to a specific Logical Group
- The value of the last measure sent by a specific sensor
- The set of measures sent by a specific sensor in a range of time
- The set of values of a specific phenomenon measured by the sensor in a range of time
- The last register SensorML message sent by a specific sensor



PUBLIC, SMARTSANTANDER PROJECT

Additionally, the application could retrieve accumulative values over a particular storage attribute that have been defined previously during the provision process of the USN customer.

The accumulative operations that can be configured are:

- COUNT. To obtain the number of times that the attribute has been received.
- MAX. To obtain the maximum number of the attribute.
- MIN. To obtain the minimum number of the attribute.
- SUM. To obtain the addition of the received values of the attribute.
- AVERAGE. To obtain the averaged value of the received values of the attribute.

And the accumulative queries offered by the API are:

- List of accumulated values related to a specific sensor in a particular day
- List of accumulated values related to a specific sensor per hour
- List of accumulated values related to a particular attribute of the Business information in a particular day
- List of accumulated values related to a particular attribute of the Business information per hour



PUBLIC, SMARTSANTANDER PROJECT

- **Query from getting stored data. USN API primitive: getDeviceData**

```
// In these fields the answer will be received
ResultException excp = null;
for (int i = 0; i < 2; i++) {
    Holder<ServiceIdType> tipo = new Holder<ServiceIdType>(serviceType);
    Holder<DeviceIdType> device = new Holder<DeviceIdType>(deviceType);
    Holder<String> sensorML0 = new Holder<String>();
        Holder<AccumulatedMeasurementsType>          acumulatedML0          =          new
Holder<AccumulatedMeasurementsType>();
    Holder<List<MeasureType>> measure = new Holder<List<MeasureType>>();
    try {
        stSensorDataPort.getDeviceData(tipo, device, sensorML,
            lastMeasure, measures, null, sensorML0, measure,
            acumulatedML0);
        List<MeasureType> medidas = measure.value;
        String medidaSensorml = sensorML0.value;
        MECObservationInfo[] res = null;
        if (medidas != null) {
            res = tratamedidasLista(plugID, medidas, dato, deviceId);
        } else if (medidaSensorml != null) {
            res = tratamedidasSensorML(medidaSensorml, dato);
        } else {
            stLogger.warn(
                plugID,
                "recuperDatos: No values for the device "
                    + deviceId
                    + " in the time range selected for "
                    + dato);
        }
    }
}
```

Example with a SOAP Body message using by a real application that use USN:



PUBLIC, SMARTSANTANDER PROJECT

```

</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <m2msdatas1t:getDataByService>
    <m2msdatas1t:serviceID>
      <m2msdatas1t:logicalName>MyEnergy</m2msdatas1t:logicalName>
    </m2msdatas1t:serviceID>
    <m2msdatas1t:accumulatedMeasurements>
      <m2msdatas1t:businessDeviceAttribute>
        <m2msdatas1t:name aggregator="">Perfil</m2msdatas1t:name>
        <m2msdatas1t:value>PARAM_PERFIL_VALUE</m2msdatas1t:value>
      </m2msdatas1t:businessDeviceAttribute>
      <m2msdatas1t:filterDate>
        <m2msdatas1t:from>PARAM_DATE_FROM</m2msdatas1t:from>
        <m2msdatas1t:to>PARAM_DATE_NOW</m2msdatas1t:to>
        <m2msdatas1t:dateAttributeName>EndEventTime</m2msdatas1t:dateAttributeName>
      </m2msdatas1t:filterDate>
      <m2msdatas1t:interval>PARAM_INTERVAL</m2msdatas1t:interval>
      <m2msdatas1t:attributes>
        <m2msdatas1t:name aggregator="PARAM_AGGREGATOR">PARAM_ATTRIBUTE_NAME</m2msdatas1t:name>
      </m2msdatas1t:attributes>
    </m2msdatas1t:accumulatedMeasurements>
  </m2msdatas1t:getDataByService>
</SOAP-ENV:Body>

```

When the parameters could have different values like:

- Hourly measures received between two hours query (getDeviceData):

PARAM_ATTRIBUTE_NAME=energyMeterHour, energyPlugHour

PARAM_ATTRIBUTE_NAME=enCostMeterHour, enCostPlugHour

Or

- maximumPower

PARAM_SCOPE: DAILY (7 days or 30 days between two dates maximum)

PARAM_SCOPE_ACCUMULATED: YEARLU (the maximum from the beginning of the year=

PARAM_PROFILE: null

PARAM_SCOPE_PROFILE: null

SOAP CLIENT SKELETON USING AXIS2 & ECLIPSE

This paragraph aims to help developers in to rapidly prepare a skeleton of an application able to perform subscriptions to IDAS and to receive notifications from the platform.

The user only has to add the GUI on top of the skeleton to start interacting with IDAS.

5.1.1.9. Implementation

A tutorial for service implementation using Eclipse IDE can be found in appendix 6.

5.2. REPOSITORY FOR PARTICIPATORY SENSING

In this section we explain in detail the functionalities provided by the SmartSantander platform that assist the developer to build service(s) and/or application(s) that follow a participatory sensing approach. Moreover we describe the functionalities provided by the PSens server and how it can be used by other applications and services.

5.2.1. PSens Server

The PSens Server is a server component that sits on the Service Manager Layer of the SmartSantander platform. This component supports the registration/deregistration of participatory sensing devices into the SmartSantander platform; submission of observations into the SmartSantander platform; and retrieval of historical sensed data. The following figure shows the architecture of the Pace Of The City service; the service that makes use of the participatory sensing approach.

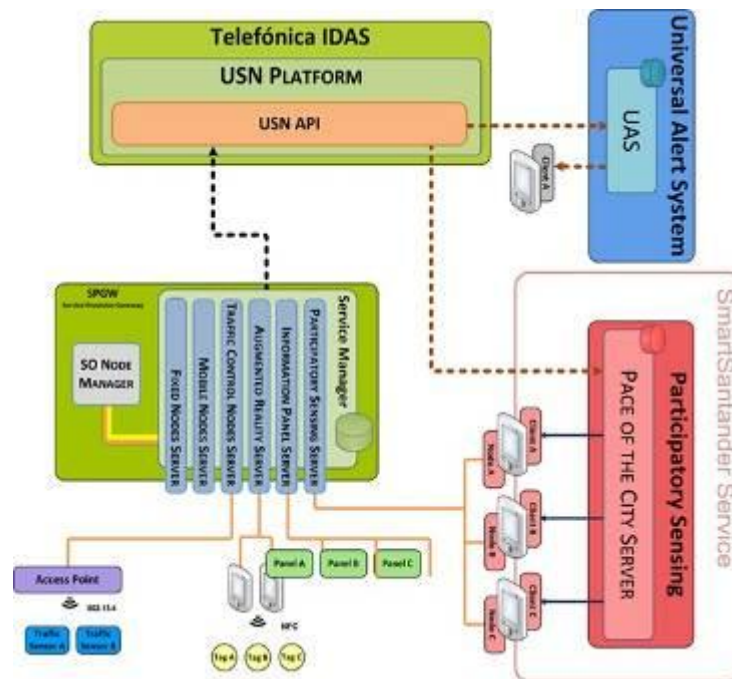


Figure 97: Pace of The City Architecture

The mobile nodes on the right connect to the PSens Server in order to register themselves; to send observations; and to retrieve historical data. Each of these functionalities is available via a RESTfull interface provided by the PSens Server. In the following sections we will explain in detail each of the functionalities.

5.2.2. PSens Server interface

The PSens Server component provides the following operations:

- `RegisterDeviceResponseDTO registerDevice(DeviceDTO device);`

PUBLIC, SMARTSANTANDER PROJECT

- `UnregisterDeviceResponseDTO unregisterDevice(String deviceUUID);`
- `ObservationPublicationResponseDTO submitObservations(Collection<Observation> observations, String deviceUUID);`
- `Collection<Observation> getHistoricalData(HistoricalPeriod period);`

5.2.3. Device Registration

The registration message call contains the device information. The device information is represented by the DeviceDTO class. This object contains the following structure:

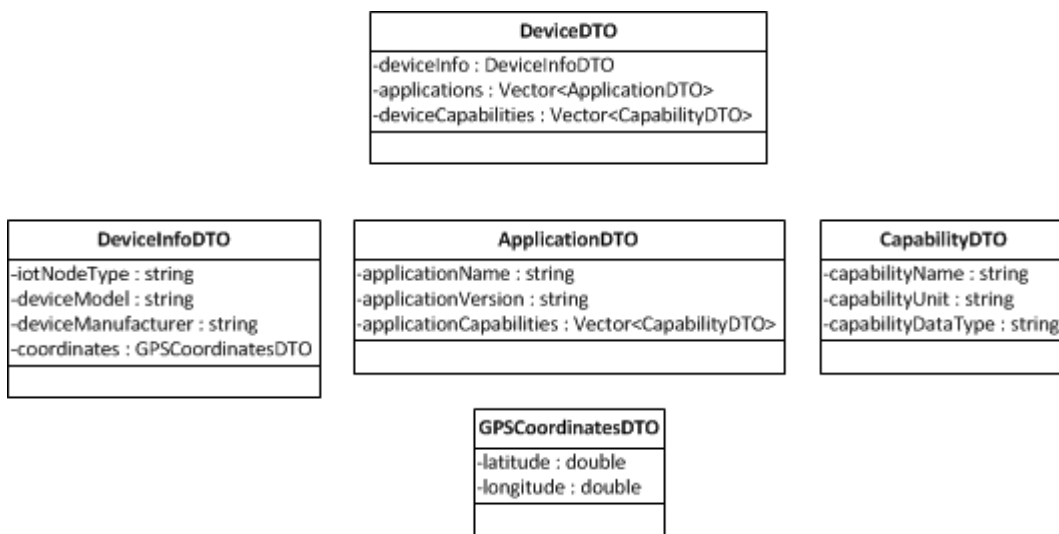


Figure 98: Class diagram - DeviceDTO

The DeviceDTO object contains three elements: the deviceInfo of type DeviceInfoDTO, a collection of device capabilities, and a collection of applications. The device capabilities are the sensor capabilities that the device has; and the applications are the applications that will make use of the participatory sensing; each application has a collection of capabilities that are the application related capabilities that will be used by the application to submit observations to the SmartSantander platform. As result to the device registration call the application or service gets a unique identifier deviceUUID generated by the PSens Server when registering the device into the SmartSantander platform. This deviceUUID needs to be stored in the phone since you will need it to interact with the PSens Server in further calls.

5.2.4. Submit Observations

To submit observations the application or service needs to call the submitObservation method; an observation contains a date; the gps coordinates where it took place; a deviceUUID and a set of measurements; a measurement contains a type, a raw value and units. The following class diagram shows the classes involved:

PUBLIC, SMARTSANTANDER PROJECT

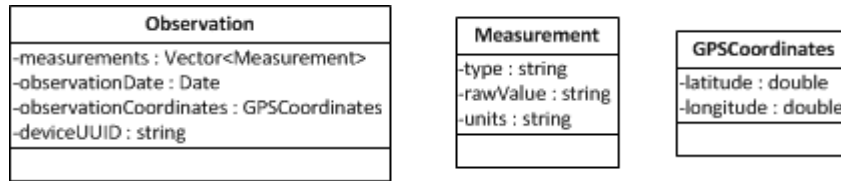


Figure 99: Class diagram - Observation

The method `submitObservations()` takes a set of observations and the deviceUUID of your device (the one returned from PSens when calling the register method).

- `ObservationPublicationResponseDTO submitObservations(Collection<Observation> observations, String deviceUUID);`

As result of the submit observations call you will receive a boolean; if true then the submission of observations to the SmartSantander was successful; false if an error occurred.

5.2.5. **Get Historical Data**

To retrieve historical observation data from your device you need to call the `getHistoricalData` of the participatory sensing service interface; this method takes one parameter, the period of `HistoricalPeriod` type; this type contains a date from and to where the user specifies the period of time from which you want to retrieve the historical data.

- `Collection<Observation> getHistoricalData(HistoricalPeriod period);`

As result you will receive the set of observations of your device that occurred in the time frame you specified as parameter.

5.2.6. **Unregister Device**

In order to deregister your device in the SmartSantander platform you will need to call the `unregisterDevice` method; this method takes the deviceUUID of your device as parameter.

- `UnregisterDeviceResponseDTO unregisterDevice(String deviceUUID);`

As result you will receive a boolean; if true the device was successfully deregistered from the SmartSantander platform; false otherwise.

PUBLIC, SMARTSANTANDER PROJECT

5.3. REPOSITORY FOR AUGMENTED REALITY

In this section we explain in detail the tools provided by the SmartSantander platform that assist the developer to build service(s) and/or application(s) that follow an augmented reality approach. Furthermore we describe the functionalities provided by the augmented reality server.

5.3.1. Augmented Reality Server

To support all these desired functionalities, the SmartSantander platform provides the Augmented reality Server that supports the registration/deregistration of augmented reality devices into the SmartSantander platform; submission of observations into the SmartSantander platform; and retrieval of data. The following figure shows the interaction between the SmartSantander platform and the Augmented Reality Service. Moreover, the platform offer over 2500 NFC/QR tags placed all over the city (shops, bus stops, tourist point of interest ...) that all accessible to the end users.

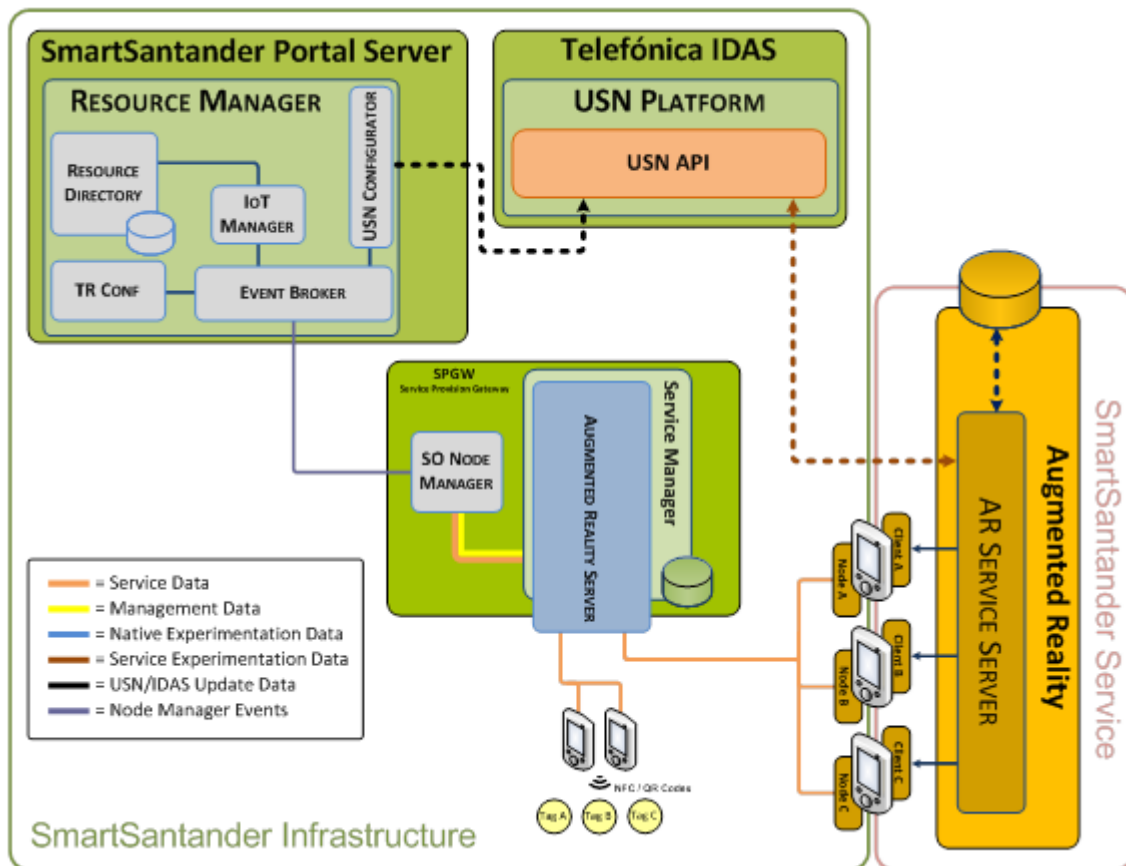


Figure 100: Augmented Reality Architecture

PUBLIC, SMARTSANTANDER PROJECT

The mobile nodes on the right connect to the AR Server in order to register themselves and to send observations. Each of these functionalities is available via a RESTfull interface provided by the Augmented Reality Server. In the following sections we will explain in detail each of the functionalities.

5.3.2. Augmented Reality Server interface

The Augmented Reality Server component provides the following operations:

- RegisterDeviceResponseDTO registerDevice(DeviceDTO device);
- UnregisterDeviceResponseDTO unregisterDevice(String deviceUUID);
- To get data and submit observation is need to make a HTTP POST request to the Server with a valid url.

5.3.3. Device Registration

The registration message call contains the device information. The device information is represented by the DeviceDTO class. This object contains the following structure:

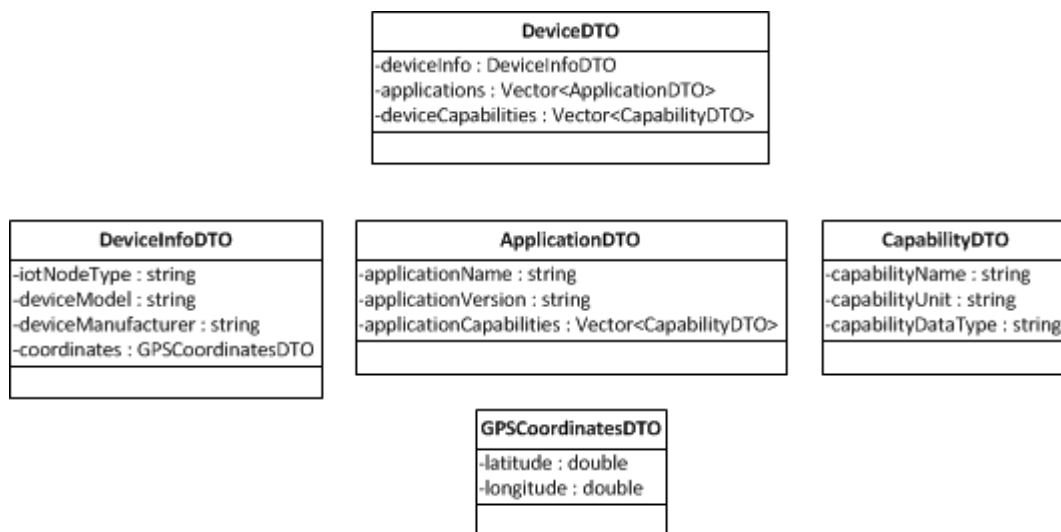


Figure 101: Class diagram - DeviceDTO

The DeviceDTO object contains three elements: the deviceInfo of type DeviceInfoDTO, a collection of device capabilities, and a collection of applications. The device capabilities are the physical capabilities that the device has (NFC chip, camera); and the applications are the applications that will make use of the augmented reality; each application has a collection of capabilities that are the application related capabilities that will be used by the application to submit observations to the SmartSantander platform. As result to the device registration call the application or service gets a unique identifier deviceUUID generated by the AR Server when registering the device into the SmartSantander platform. This deviceUUID needs to be stored in the phone since you will need it to interact with the AR Server in further calls.

5.3.4. Get Data and submit Observations

**PUBLIC, SMARTSANTANDER PROJECT**

When the application or service ask for POI data needs to send an HTTP Post request to the AR server; the URL used in the request will have and array of parameters that the AR server will use to create observations. The possible parameters includes in the AR server are:

- Max number of POIs (“maxPoi”): The maximum number of POI that the app showed in the AR views.
- Max Search Distance (“dist”):
- InfoType(“type”): The type of info request. The possible values are:
 - TUS: Transport
 - CUL: Culture
 - COM: Commerce
 - TUR: Tourism
 - AGENDA: Agenda
- Language(“lang”).
- deviceOS. This parameter is included in the http header of the request.
- POI id (“id”).
- Tag Type (“tagType”): NFC or QR
- Tag id (“tagId”).
- URL Latitude (“lat”).
- Longitude (“lon”).
- deviceUUID (“uuid”).

An example of an URL extract from the SmartSantanderRA:

<http://fake.server.url/server.php?lat=-3.40005&lon=42.543563&maxPoi=30&lang=EN&type=CUL&dist=10&uuid=bf5eb4cd-97eb-498b-af09-dca3810b68ca>

This URL request the server the closest 30 culture POIs to a position (42.543563, -3.40005).

As result a HTTP Response with the available data is returned.

5.3.5. Unregister Device

In order to deregister your device in the SmartSantander platform you will need to call the unregisterDevice method; this method takes the deviceUUID of your device as parameter.

- UnregisterDeviceResponseDTO unregisterDevice(String deviceUUID);

As result you will receive a Boolean; if true the device was successfully deregistered from the SmartSantander platform; false otherwise.

PUBLIC, SMARTSANTANDER PROJECT

6. INNOVATION INCUBATOR

Besides the services above explained and already operative, we believe that Smart Santander projects goes beyond the traditional smart city concept. This is motivated by several reasons. Firstly, besides supporting the traditional vertical services we are creating a framework which companies, research centers and individuals can access to experiment on top of the platform. This means that the project provides a unique platform which enables to design and develop services very close to present and future society needs. Indeed, the fact that a significant number of services are already supported implies that data associated with these services are generated. These data become a key enabler for conceiving new services and applications. Furthermore, the appropriate combination of such data allows reaching an optimal working point for the mix of urban services, all this in real time. Indeed, going beyond the data the project has demonstrated the important role that the urban platform will play in the future of the smart city concept. Not just in terms of monitoring and controlling the city but also integrating the citizen in the scenario. In this sense, the figure bellow [1] shows an approach for such urban platform which accommodates service management and citizen participation.

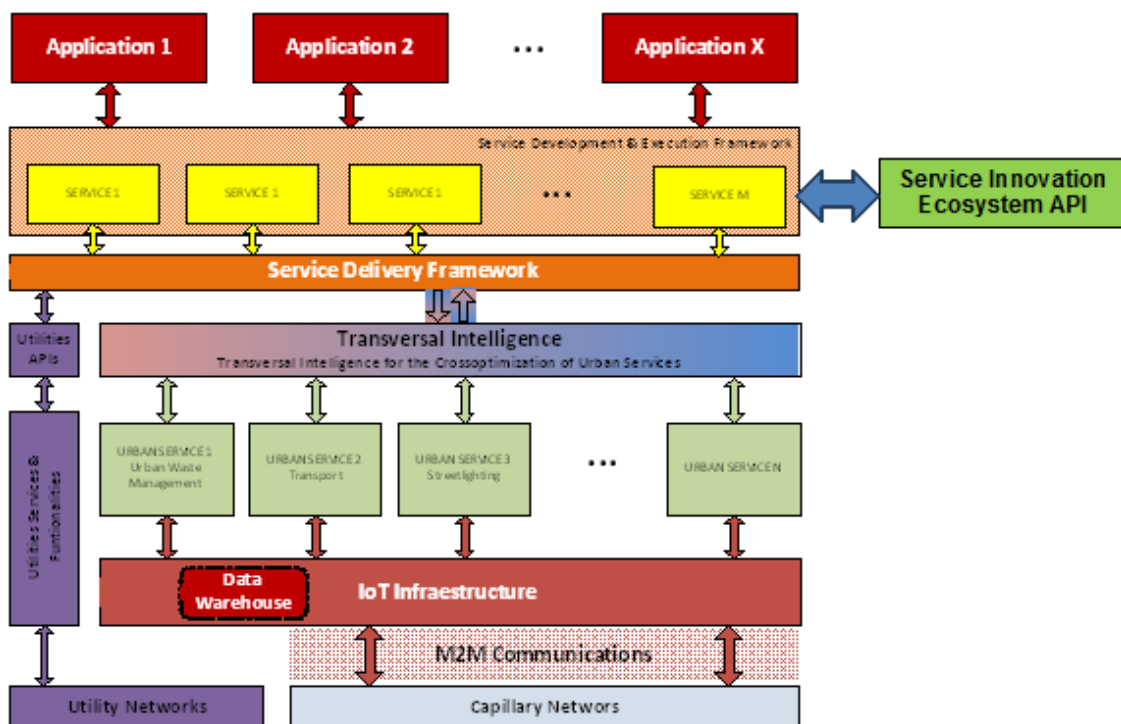


Figure 102: Urban Platform

Besides the vertical service silos and transversal intelligence the figure clearly highlights the API associated to the innovation ecosystem which should stimulate the development of new services and its seamless integration in the smart city landscape.

PUBLIC, SMARTSANTANDER PROJECT

7. CONCLUSIONS

In WP4 we have been looking into the service umbrella of the smart city paradigm. Due to its heterogeneity the services comprise different application domains that make the service and application realisation a very challenging and interesting task. As has been shown in this deliverable, the task 4.2 has carried out an extensive work on developing services that have served to demonstrate and validate several points:

- It has been achieved the integration of pre-existing systems and services to architecture and facilities created and deployed in SmartSantander.
- We have been able to validate and demonstrate the duality of service and experimentation provision by the SmartSantander platform.
- Through the different developed scenarios has been shown that SmartSantander is not only a platform designed to be operated in a single location or city such as Santander, the realization of scenarios in other cities like Belgrade or Guildford has highlighted the exportability of its model and demonstrated the simplicity of integrating such test-bed facilities into a common framework.
- It has been used many different technological approaches for developing the different services and applications, using different programming technologies (mobile, web, desktop, etc) which also have demonstrated the interoperability of the SmartSantander facilities.
- In addition to the above, it has been provided to the citizens several useful and innovative services, being this one of the main objectives of the SmartSantander project. Nevertheless in Task 4.3 it will be assessed precisely all these services with reference to the indicators that were described in D4.1 as result of T4.1.
- In this task we have also developed and integrated new software components in the SmartSantander platform. These components include the Participatory Sensing Server and the Augmented Reality Server and allow easier integration of new and innovative services by providing interfaces that allow registration/deregistration of devices and submission of observations into the SmartSantander platform.
- In some use cases the developed smart phone applications are available to the users in the respective markets for download; meaning that its state is not as prototype but as a product available to the citizens.

With all the above mentioned we can conclude that through the task 4.2 of WP4 we have been able to verify the correct operation of the deployed facilities in SmartSantander. We have been able to develop valuable services that in some cases are already available to the citizens and therefore have direct impact on them.

In the scope of service development we are also providing tools/software components that can serve as basis for improvements of the existing use case scenarios covered in this deliverable and also for future developments of new and innovative services of diverse nature. This approach, outlined in Section 6 brings the possibility of generating an innovation ecosystem that makes more feasible the concept of a Smart City. It allows a simple integration of applications and services into a common platform and reduces the amount of work to be done by a service and/or application developer in order to integrate such developments with this common framework.



SMARTSANTANDER

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES



PUBLIC, SMARTSANTANDER PROJECT

REFERENCES

- [1] L. Muñoz et al., "Boosting Smart Cities through the Synergies with the Utilities", submitted to the IEEE Communications Magazine special issue on Smart Cities, January 2013
- [2] http://www.cragssystems.co.uk/why_use_uml.htm



APPENDIX

1. Parking Service WSDL specification

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions          targetNamespace="http://smartsantander.eu/parkingservice/"
xmlns:parking_types="http://smartsantander.eu/parkingservice/types"
xmlns:parking="http://smartsantander.eu/parkingservice/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  <wsdl:types>
    <xsd:schema              targetNamespace="http://smartsantander.eu/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      <xsd:import          namespace="http://smartsantander.eu/parkingservice/types"
schemaLocation="SmartSantanderParkingServiceTypes.xsd"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="getParkingLotsRequest">
  </wsdl:message>
  <wsdl:message name="getParkingLotsResponse">
    <wsdl:part name="parkingLots" element="parking_types:GetParkingLotsResponse">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="ParkingServicePort">
    <wsdl:operation name="getParkingLots">
      <wsdl:input message="parking:getParkingLotsRequest">
      </wsdl:input>
      <wsdl:output message="parking:getParkingLotsResponse">
      </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ParkingServiceSOAPBinding" type="parking:ParkingServicePort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getParkingLots">
      <soap:operation soapAction="urn:getParkingLots"/>
      <wsdl:input>

      </wsdl:input>
      <wsdl:output>

      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ParkingService">
    <wsdl:port name="ParkingService" binding="parking:ParkingServiceSOAPBinding">
      <soap:address location="http://localhost/services/parkingService"/>
    </wsdl:port>
```



PUBLIC, SMARTSANTANDER PROJECT

```
</wsdl:service>
</wsdl:definitions>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:parking_types="http://smartsantander.eu/parkingservice/types"
attributeFormDefault="unqualified" elementFormDefault="unqualified"
targetNamespace="http://smartsantander.eu/parkingservice/types">
  <xsd:complexType name="GPSCoordinates">
    <xsd:sequence>
      <xsd:element name="latitude" type="xsd:double"/>
      <xsd:element name="longitude" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ParkingSpaceType">
    <xsd:sequence>
      <xsd:element name="parkingSpaceType" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SpaceCurrentStatusType">
    <xsd:sequence>
      <xsd:element name="space" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ParkingSpace">
    <xsd:sequence>
      <xsd:element name="parkingSpaceType" type="parking_types:ParkingSpaceType"/>
      <xsd:element name="currentStatus"
type="parking_types:SpaceCurrentStatusType"/>
      <xsd:element name="parkingSpaceCoordinates"
type="parking_types:GPSCoordinates"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ParkingLot">
    <xsd:sequence>
      <xsd:element name="parkingLotAddress" type="xsd:string"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="parkingSpaces"
type="parking_types:ParkingSpace"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="GetParkingLotsResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="parkingLot"
type="parking_types:ParkingLot"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



PUBLIC, SMARTSANTANDER PROJECT

2. Environmental Monitoring WSDL specification

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://smartsantander.eu/environmentalmonitoringservice/"
xmlns:environmental_types="http://smartsantander.eu/environmentalmonitoringservice/types"
xmlns:environmental="http://smartsantander.eu/environmentalmonitoringservice/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://smartsantander.eu/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
namespace="http://smartsantander.eu/environmentalmonitoringservice/types"
schemaLocation="SmartSantanderEnvironmentalMonitoringServiceTypes.xsd"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="getEnvironmentalDataRequest">
  </wsdl:message>
  <wsdl:message name="getEnvironmentalDataResponse">
    <wsdl:part name="environmentalData"
element="environmental_types:GetEnvironmentalDataResponse">
  </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="EnvironmentalMonitoringServicePort">
    <wsdl:operation name="getEnvironmentalData">
      <wsdl:input message="environmental:getEnvironmentalDataRequest">
  </wsdl:input>
      <wsdl:output message="environmental:getEnvironmentalDataResponse">
  </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="EnvironmentalMonitoringServiceSOAPBinding"
type="environmental:EnvironmentalMonitoringServicePort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getEnvironmentalData">
      <soap:operation soapAction="urn:getEnvironmentalData"/>
      <wsdl:input>
  </wsdl:input>
      <wsdl:output>
  </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="EnvironmentalMonitoringService">
    <wsdl:port name="EnvironmentalMonitoringService"
binding="environmental:EnvironmentalMonitoringServiceSOAPBinding">
      <soap:address
location="http://localhost/services/environmentalmonitoringservice"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:environmental_types="http://smartsantander.eu/environmentalmonitoringservice/types"
```



PUBLIC, SMARTSANTANDER PROJECT

```
attributeFormDefault="unqualified" elementFormDefault="unqualified"
targetNamespace="http://smartsantander.eu/environmentalmonitoringservice/types">
  <xsd:complexType name="GPSCoordinates">
    <xsd:sequence>
      <xsd:element name="latitude" type="xsd:double"/>
      <xsd:element name="longitude" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SensorType">
    <xsd:sequence>
      <xsd:element name="sensorType" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CurrentMeasuredValue">
    <xsd:sequence>
      <xsd:element name="value" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CurrentMeasuredValueUnits">
    <xsd:sequence>
      <xsd:element name="units" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="EnvironmentalSensor">
    <xsd:sequence>
      <xsd:element name="sensorType" type="environmental_types:SensorType"/>
      <xsd:element name="currentMeasuredValue"
type="environmental_types:CurrentMeasuredValue"/>
      <xsd:element name="currentMeasuredValueUnits"
type="environmental_types:CurrentMeasuredValueUnits"/>
      <xsd:element name="sensorCoordinates"
type="environmental_types:GPSCoordinates"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="GetEnvironmentalDataResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="environmentalSensor"
type="environmental_types:EnvironmentalSensor"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

3. USN subscription WSDL specification

UNICA_API_SOAP_m2m_subscription_services_v1_1.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- October, 2010 -->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:m2m_subs_s1="http://www.telefonica.com/wsdl/UNICA/SOAP/m2m/subscription/v1/services"
xmlns:m2m_subs_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscription/v1/types"
xmlns:uch="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v1/security_headers"
xmlns:ucf="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v1/faults"
xmlns:utih="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v2/transaction_info_header"
```

PUBLIC, SMARTSANTANDER PROJECT

```
targetNamespace="http://www.telefonica.com/wsd/UNICA/SOAP/m2m/subscription/v1/services">
  <wsdl:import
    namespace="http://www.telefonica.com/wsd/UNICA/SOAP/common/v1/faults"
    location="UNICA_API_SOAP_common_faults_v1_0.wsdl"/>
  <wsdl:import
    namespace="http://www.telefonica.com/wsd/UNICA/SOAP/common/v1/security_headers"
    location="UNICA_API_SOAP_common_security_headers_v1_0.wsdl"/>
  <wsdl:import
    namespace="http://www.telefonica.com/wsd/UNICA/SOAP/common/v2/transaction_info_header"
    location="UNICA_API_SOAP_common_transaction_info_header_v2_0.wsdl"/>
  <wsdl:types>
    <xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.telefonica.com/wsd/UNICA/SOAP/m2m/v1/" elementFormDefault="qualified">
      <xsd:import namespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscription/v1/types"
        schemaLocation="UNICA_API_SOAP_m2m_subscription_types_v1_1.xsd"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="SubscribeRequest">
    <wsdl:part name="parameters" element="m2m_subs_s1t:subscribe"/>
  </wsdl:message>
  <wsdl:message name="SubscribeResponse">
    <wsdl:part name="parameters" element="m2m_subs_s1t:subscribeResponse"/>
  </wsdl:message>
  <wsdl:message name="UnsubscribeRequest">
    <wsdl:part name="parameters" element="m2m_subs_s1t:unsubscribe"/>
  </wsdl:message>
  <wsdl:message name="UnsubscribeResponse">
    <wsdl:part name="parameters" element="m2m_subs_s1t:unsubscribeResponse"/>
  </wsdl:message>
  <wsdl:message name="DisconnectRequest">
    <wsdl:part name="parameters" element="m2m_subs_s1t:disconnect"/>
  </wsdl:message>
  <wsdl:message name="DisconnectResponse">
    <wsdl:part name="parameters" element="m2m_subs_s1t:disconnectResponse"/>
  </wsdl:message>
  <wsdl:portType name="SubscriptionPort">
    <wsdl:operation name="subscribe">
      <wsdl:input message="m2m_subs_s1:SubscribeRequest"/>
      <wsdl:output message="m2m_subs_s1:SubscribeResponse"/>
      <wsdl:fault name="ClientException" message="ucf:ClientException"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:service>
```

PUBLIC, SMARTSANTANDER PROJECT

```
<wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
<wsdl:operation name="unsubscribe">
  <wsdl:input message="m2m_subs_s1:UnsubscribeRequest"/>
  <wsdl:output message="m2m_subs_s1:UnsubscribeResponse"/>
  <wsdl:fault name="ClientException" message="ucf:ClientException"/>
  <wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
<wsdl:operation name="disconnect">
  <wsdl:input message="m2m_subs_s1:DisconnectRequest"/>
  <wsdl:output message="m2m_subs_s1:DisconnectResponse"/>
  <wsdl:fault name="ClientException" message="ucf:ClientException"/>
  <wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="SubscriptionSOAPBinding" type="m2m_subs_s1:SubscriptionPort">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="subscribe">
    <soap:operation soapAction="urn:subscribe"/>
    <wsdl:input>
      <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
    <soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="unsubscribe">
  <soap:operation soapAction="urn:unsubscribe"/>
```


PUBLIC, SMARTSANTANDER PROJECT

```
<wsdl:input>
    <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
<soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
    <soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
    <soap:body use="literal"/>
</wsdl:output>
<wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="disconnect">
    <soap:operation soapAction="urn:disconnect"/>
    <wsdl:input>
        <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
<soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ClientException">
        <soap:fault name="ClientException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="ServerException">
        <soap:fault name="ServerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="M2MSubscriptionService">
    <wsdl:port name="Subscription" binding="m2m_subs_s1:SubscriptionSOAPBinding">
        <soap:address location="http://localhost/UNICA_SDP/M2M/Subscription"/>
    </wsdl:port>
</wsdl:service>
</wsdl:binding>
</wsdl:service>
```

PUBLIC, SMARTSANTANDER PROJECT

```

        </wsdl:port>

    </wsdl:service>

</wsdl:definitions>

```

UNICA_API_SOAP_m2m_subscription_types_v1_1.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- October, 2010 -->
<xsd:schema                                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:m2m_subs_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscription/v1/types"
xmlns:uct="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
targetNamespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscription/v1/types"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:import                            namespace="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
schemaLocation="UNICA_API_SOAP_common_types_v1_0.xsd"/>
    <!-- Business data -->
    <xsd:simpleType name="EventKindType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Register"/>
            <xsd:enumeration value="Observation"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="SubscriptionType">
        <xsd:sequence>
            <xsd:element name="eventKind" type="m2m_subs_s1t:EventKindType" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="notifyURI" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="serviceLogicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="clientAppLogicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="subscriptionLogicalName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="xpath" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="timed" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="seconds" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="original" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="priority" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>

```

PUBLIC, SMARTSANTANDER PROJECT

```

        </xsd:sequence>
    </xsd:complexType>
    <!-- Operation types -->
    <xsd:element name="subscribe">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="responseURI" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                <xsd:element name="subscription" type="m2m_subs_s1t:SubscriptionType"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="subscribeResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="result" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="unsubscribe">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="responseURI" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                <xsd:element name="subscriptionLogicalName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                <xsd:element name="outgoingConnectionId" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="unsubscribeResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="result" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

```

PUBLIC, SMARTSANTANDER PROJECT

```

        </xsd:complexType>
    </xsd:element>
    <xsd:element name="disconnect">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="responseURI" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                <xsd:element name="outgoingConnectionId" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="disconnectResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="result" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

UNICA_API_SOAP_m2m_subscriptionresponse_services_v1_1.wsdl

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- October, 2010 -->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:m2m_subsresp_s1="http://www.telefonica.com/wsdl/UNICA/SOAP/m2m/subscriptionresponse/v1/services"
xmlns:m2m_subsresp_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscriptionresponse/v1/types"
xmlns:uch="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v1/security_headers"
xmlns:ucf="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v1/faults"
xmlns:utih="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v2/transaction_info_header"
targetNamespace="http://www.telefonica.com/wsdl/UNICA/SOAP/m2m/subscriptionresponse/v1/services">
    <wsdl:import namespace="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v1/faults"
location="UNICA_API_SOAP_common_faults_v1_0.wsdl"/>
    <wsdl:import namespace="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v1/security_headers"
location="UNICA_API_SOAP_common_security_headers_v1_0.wsdl"/>
    <wsdl:import namespace="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v2/transaction_info_header"
location="UNICA_API_SOAP_common_transaction_info_header_v2_0.wsdl"/>
    <wsdl:types>

```

PUBLIC, SMARTSANTANDER PROJECT

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.telefonica.com/wsdl/UNICA/SOAP/m2m/v1/" elementFormDefault="qualified">
    <xsd:import
namespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscriptionresponse/v1/types"
schemaLocation="UNICA_API_SOAP_m2m_subscriptionresponse_types_v1_1.xsd"/>
    </xsd:schema>
</wsdl:types>
<wsdl:message name="SubscribeResponseRequest">
    <wsdl:part name="parameters" element="m2m_subsresp_s1t:subscribeResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="SubscribeResponseResponse">
    <wsdl:part name="parameters" element="m2m_subsresp_s1t:subscribeResponseResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="UnsubscribeResponseRequest">
    <wsdl:part name="parameters" element="m2m_subsresp_s1t:unsubscribeResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="UnsubscribeResponseResponse">
    <wsdl:part name="parameters" element="m2m_subsresp_s1t:unsubscribeResponseResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="DisconnectResponseResponse">
    <wsdl:part name="parameters" element="m2m_subsresp_s1t:disconnectResponseResponse">
</wsdl:part>
</wsdl:message>
<wsdl:message name="DisconnectResponseRequest">
    <wsdl:part name="parameters" element="m2m_subsresp_s1t:disconnectResponse">
</wsdl:part>
</wsdl:message>
<wsdl:portType name="SubscriptionResponsePort">
    <wsdl:operation name="subscribeResponse">
        <wsdl:input message="m2m_subsresp_s1:SubscribeResponseRequest"/>
        <wsdl:output message="m2m_subsresp_s1:SubscribeResponseResponse"/>
        <wsdl:fault name="ClientException" message="ucf:ClientException"/>
    </wsdl:operation>
</wsdl:portType>

```

PUBLIC, SMARTSANTANDER PROJECT

```

        <wsdl:fault name="ServerException" message="ucf:ServerException"/>
    </wsdl:operation>
    <wsdl:operation name="unsubscribeResponse">
        <wsdl:input message="m2m_subsresp_s1:UnsubscribeResponseRequest"/>
        <wsdl:output message="m2m_subsresp_s1:UnsubscribeResponseResponse"/>
        <wsdl:fault name="ClientException" message="ucf:ClientException"/>
        <wsdl:fault name="ServerException" message="ucf:ServerException"/>
    </wsdl:operation>
    <wsdl:operation name="disconnectResponse">
        <wsdl:input message="m2m_subsresp_s1:DisconnectResponseRequest"/>
        <wsdl:output message="m2m_subsresp_s1:DisconnectResponseResponse"/>
        <wsdl:fault name="ClientException" message="ucf:ClientException"/>
        <wsdl:fault name="ServerException" message="ucf:ServerException"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="SubscriptionResponseSOAPBinding" type="m2m_subsresp_s1:SubscriptionResponsePort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="subscribeResponse">
        <soap:operation soapAction="urn:subscribeResponse"/>
        <wsdl:input>
            <soap:header
                message="utih:UplinkTransactionInfoHeader"
                part="uplinkTransactionInfoHeader" use="literal"/>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="ClientException">
            <soap:fault name="ClientException" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ServerException">
            <soap:fault name="ServerException" use="literal"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="unsubscribeResponse">
        <soap:operation soapAction="urn:unsubscribeResponse"/>

```



PUBLIC, SMARTSANTANDER PROJECT

```
<wsdl:input>
    <soap:header message="utih:UplinkTransactionInfoHeader"
part="uplinkTransactionInfoHeader" use="literal"/>
    <soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
    <soap:body use="literal"/>
</wsdl:output>
<wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="disconnectResponse">
    <soap:operation soapAction="urn:disconnectResponse"/>
    <wsdl:input>
        <soap:header message="utih:UplinkTransactionInfoHeader"
part="uplinkTransactionInfoHeader" use="literal"/>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ClientException">
        <soap:fault name="ClientException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="ServerException">
        <soap:fault name="ServerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="M2MSubscriptionResponseService">
    <wsdl:port name="SubscriptionResponse"
binding="m2m_subsresp_s1:SubscriptionResponseSOAPBinding"/>
</wsdl:service>
</wsdl:binding>
```

PUBLIC, SMARTSANTANDER PROJECT

```
<soap:address location="http://localhost/UNICA_SDP/M2M/SubscriptionResponse"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

UNICA_API_SOAP_m2m_subscriptionresponse_types_v1_1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- October, 2010 -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:m2m_subs_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscriptionresponse/v1/types"
  xmlns:uct="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
  targetNamespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/subscriptionresponse/v1/types" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xsd:import namespace="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
    schemaLocation="UNICA_API_SOAP_common_types_v1_0.xsd"/>
  <!-- Business data -->
  <!-- Operation types -->
  <xsd:element name="subscribeResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="subscriptionLogicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="outgoingConnectionId" type="xsd:int" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="errorCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="errorText" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="subscribeResponseResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="result" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="disconnectResponse">
    <xsd:complexType>
```


PUBLIC, SMARTSANTANDER PROJECT

```

        <xsd:sequence>
            <xsd:element name="errorCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="errorText" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="disconnectResponseResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="result" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="unsubscribeResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="subscriptionLogicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="errorCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="errorText" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="unsubscribeResponseResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="result" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

4. USN Notification WSDL specification

UNICA_API_SOAP_m2m_notification_services_v1_1.wsdl

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- October, 2010 -->
<wSDL:definitions          xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"          xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"

```



PUBLIC, SMARTSANTANDER PROJECT

```
xmlns:m2m_notif_s1="http://www.telefonica.com/wsd/UNICA/SOAP/m2m/notification/v1/services"
xmlns:m2m_notif_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/notification/v1/types"
xmlns:uch="http://www.telefonica.com/wsd/UNICA/SOAP/common/v1/security_headers"
xmlns:ucf="http://www.telefonica.com/wsd/UNICA/SOAP/common/v1/faults"
xmlns:utih="http://www.telefonica.com/wsd/UNICA/SOAP/common/v2/transaction_info_header"
targetNamespace="http://www.telefonica.com/wsd/UNICA/SOAP/m2m/notification/v1/services">

    <wsdl:import
        namespace="http://www.telefonica.com/wsd/UNICA/SOAP/common/v1/faults"
        location="UNICA_API_SOAP_common_faults_v1_0.wsdl"/>

    <wsdl:import
        namespace="http://www.telefonica.com/wsd/UNICA/SOAP/common/v1/security_headers"
        location="UNICA_API_SOAP_common_security_headers_v1_0.wsdl"/>

    <wsdl:import
        namespace="http://www.telefonica.com/wsd/UNICA/SOAP/common/v2/transaction_info_header"
        location="UNICA_API_SOAP_common_transaction_info_header_v2_0.wsdl"/>

    <wsdl:types>

        <xsd:schema
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.telefonica.com/wsd/UNICA/SOAP/m2m/v1/" elementFormDefault="qualified">

            <xsd:import
                namespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/notification/v1/types"
                schemaLocation="UNICA_API_SOAP_m2m_notification_types_v1_1.xsd"/>

            </xsd:schema>

        </wsdl:types>

        <wsdl:message name="NotifyRequest">

            <wsdl:part name="parameters" element="m2m_notif_s1t:notify">

        </wsdl:part>

        </wsdl:message>

        <wsdl:message name="NotifyResponse">

            <wsdl:part name="parameters" element="m2m_notif_s1t:notifyResponse">

        </wsdl:part>

        </wsdl:message>

        <wsdl:portType name="NotificationPort">

            <wsdl:operation name="notify">

                <wsdl:input message="m2m_notif_s1:NotifyRequest"/>

                <wsdl:output message="m2m_notif_s1:NotifyResponse"/>

                <wsdl:fault name="ClientException" message="ucf:ClientException"/>

                <wsdl:fault name="ServerException" message="ucf:ServerException"/>

            </wsdl:operation>

        </wsdl:portType>

        <wsdl:binding name="NotificationSOAPBinding" type="m2m_notif_s1:NotificationPort">

            <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

        </wsdl:binding>

    </wsdl:service>

</wsdl:service>
```



PUBLIC, SMARTSANTANDER PROJECT

```
<wsdl:operation name="notify">
  <soap:operation soapAction="urn:notify"/>
  <wsdl:input>
    <soap:header message="utih:UplinkTransactionInfoHeader" part="uplinkTransactionInfoHeader" use="literal"/>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="M2MNotificationService">
  <wsdl:port name="Notification" binding="m2m_notif_s1:NotificationSOAPBinding">
    <soap:address location="http://localhost/UNICA_SDP/M2M/Notification"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

UNICA_API_SOAP_m2m_notification_types_v1_1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- October, 2010 -->
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:m2m_notif_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/notification/v1/types"
  xmlns:uct="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
  targetNamespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/notification/v1/types"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified">
  <xsd:import
    namespace="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
    schemaLocation="UNICA_API_SOAP_common_types_v1_0.xsd"/>
  <!-- Business data -->
  <xsd:simpleType name="EventKindType">
    <xsd:restriction base="xsd:string">
```

PUBLIC, SMARTSANTANDER PROJECT

```

        <xsd:enumeration value="Register"/>
        <xsd:enumeration value="Observation"/>
    </xsd:restriction>
</xsd:simpleType>
<!-- Types of operations -->
<xsd:element name="notify">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="subscriptionLogicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="eventKind" type="m2m_notif_s1t:EventKindType" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="xmlRegister" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="notifyResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="result" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

5. USN Queries WSDL specification

UNICA_API_SOAP_m2m_sensordata_services_v1_1.wsdl

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- January, 2011 -->
<wSDL:definitions
    xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
    xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"
    xmlns:m2m_sdata_s1="http://www.telefonica.com/wSDL/UNICA/SOAP/m2m/sensordata/v1/services"
    xmlns:m2m_sdata_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/sensordata/v1/types"
    xmlns:uch="http://www.telefonica.com/wSDL/UNICA/SOAP/common/v1/security_headers"
    xmlns:ucf="http://www.telefonica.com/wSDL/UNICA/SOAP/common/v1/faults"
    xmlns:utih="http://www.telefonica.com/wSDL/UNICA/SOAP/common/v2/transaction_info_header"
    targetNamespace="http://www.telefonica.com/wSDL/UNICA/SOAP/m2m/sensordata/v1/services">
    <wSDL:import
        namespace="http://www.telefonica.com/wSDL/UNICA/SOAP/common/v1/faults"
        location="UNICA_API_SOAP_common_faults_v1_0.wsdl"/>
    <wSDL:import
        namespace="http://www.telefonica.com/wSDL/UNICA/SOAP/common/v1/security_headers"

```

PUBLIC, SMARTSANTANDER PROJECT

```
location="UNICA_API_SOAP_common_security_headers_v1_0.wsdl"/>
  <wsdl:import
    namespace="http://www.telefonica.com/wsdl/UNICA/SOAP/common/v2/transaction_info_header"
location="UNICA_API_SOAP_common_transaction_info_header_v2_0.wsdl"/>
  <wsdl:types>
    <xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.telefonica.com/wsdl/UNICA/SOAP/m2m/v1/" elementFormDefault="qualified">
      <xsd:import
        namespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/sensordata/v1/types"
schemaLocation="UNICA_API_SOAP_m2m_sensordata_types_v1_1.xsd"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="GetDevicesByLogicalGroupRequest">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getDevicesByLogicalGroup"/>
  </wsdl:message>
  <wsdl:message name="GetDevicesByLogicalGroupResponse">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getDevicesByLogicalGroupResponse"/>
  </wsdl:message>
  <wsdl:message name="GetDataByServiceRequest">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getDataByService"/>
  </wsdl:message>
  <wsdl:message name="GetDataByServiceResponse">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getDataByServiceResponse"/>
  </wsdl:message>
  <wsdl:message name="GetDeviceDataRequest">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getDeviceData"/>
  </wsdl:message>
  <wsdl:message name="GetDeviceDataResponse">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getDeviceDataResponse"/>
  </wsdl:message>
  <wsdl:message name="GetConcentratorDataRequest">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getConcentratorData"/>
  </wsdl:message>
  <wsdl:message name="GetConcentratorDataResponse">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:getConcentratorDataResponse"/>
  </wsdl:message>
  <wsdl:message name="AddDataByServiceRequest">
    <wsdl:part name="parameters" element="m2m_sdata_s1t:addDataByService"/>
```

PUBLIC, SMARTSANTANDER PROJECT

```
</wsdl:message>
<wsdl:message name="AddDataByServiceResponse">
  <wsdl:part name="parameters" element="m2m_sdata_s1t:addDataByServiceResponse"/>
</wsdl:message>
<wsdl:message name="UpdateDataByServiceRequest">
  <wsdl:part name="parameters" element="m2m_sdata_s1t:updateDataByService"/>
</wsdl:message>
<wsdl:message name="UpdateDataByServiceResponse">
  <wsdl:part name="parameters" element="m2m_sdata_s1t:updateDataByServiceResponse"/>
</wsdl:message>
<wsdl:message name="DeleteDataByServiceRequest">
  <wsdl:part name="parameters" element="m2m_sdata_s1t:deleteDataByService"/>
</wsdl:message>
<wsdl:message name="DeleteDataByServiceResponse">
  <wsdl:part name="parameters" element="m2m_sdata_s1t:deleteDataByServiceResponse"/>
</wsdl:message>
<wsdl:portType name="SensorDataPort">
  <wsdl:operation name="getDevicesByLogicalGroup">
    <wsdl:input message="m2m_sdata_s1:GetDevicesByLogicalGroupRequest"/>
    <wsdl:output message="m2m_sdata_s1:GetDevicesByLogicalGroupResponse"/>
    <wsdl:fault name="ClientException" message="ucf:ClientException"/>
    <wsdl:fault name="ServerException" message="ucf:ServerException"/>
  </wsdl:operation>
  <wsdl:operation name="getDataByService">
    <wsdl:input message="m2m_sdata_s1:GetDataByServiceRequest"/>
    <wsdl:output message="m2m_sdata_s1:GetDataByServiceResponse"/>
    <wsdl:fault name="ClientException" message="ucf:ClientException"/>
    <wsdl:fault name="ServerException" message="ucf:ServerException"/>
  </wsdl:operation>
  <wsdl:operation name="getDeviceData">
    <wsdl:input message="m2m_sdata_s1:GetDeviceDataRequest"/>
    <wsdl:output message="m2m_sdata_s1:GetDeviceDataResponse"/>
    <wsdl:fault name="ClientException" message="ucf:ClientException"/>
    <wsdl:fault name="ServerException" message="ucf:ServerException"/>
  </wsdl:operation>
  <wsdl:operation name="getConcentratorData">
```

PUBLIC, SMARTSANTANDER PROJECT

```
<wsdl:input message="m2m_sdata_s1:GetConcentratorDataRequest"/>
<wsdl:output message="m2m_sdata_s1:GetConcentratorDataResponse"/>
<wsdl:fault name="ClientException" message="ucf:ClientException"/>
<wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
<wsdl:operation name="addDataByService">
  <wsdl:input message="m2m_sdata_s1:AddDataByServiceRequest"/>
  <wsdl:output message="m2m_sdata_s1:AddDataByServiceResponse"/>
  <wsdl:fault name="ClientException" message="ucf:ClientException"/>
  <wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
<wsdl:operation name="updateDataByService">
  <wsdl:input message="m2m_sdata_s1:UpdateDataByServiceRequest"/>
  <wsdl:output message="m2m_sdata_s1:UpdateDataByServiceResponse"/>
  <wsdl:fault name="ClientException" message="ucf:ClientException"/>
  <wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
<wsdl:operation name="deleteDataByService">
  <wsdl:input message="m2m_sdata_s1>DeleteDataByServiceRequest"/>
  <wsdl:output message="m2m_sdata_s1>DeleteDataByServiceResponse"/>
  <wsdl:fault name="ClientException" message="ucf:ClientException"/>
  <wsdl:fault name="ServerException" message="ucf:ServerException"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="SensorDataSOAPBinding" type="m2m_sdata_s1:SensorDataPort">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getDevicesByLogicalGroup">
    <soap:operation soapAction="urn:getDevicesByLogicalGroup"/>
    <wsdl:input>
      <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:binding>
</wsdl:service>
```



PUBLIC, SMARTSANTANDER PROJECT

```
<wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getDataByService">
    <soap:operation soapAction="urn:getDataByService"/>
    <wsdl:input>
        <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
<soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ClientException">
        <soap:fault name="ClientException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="ServerException">
        <soap:fault name="ServerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getDeviceData">
    <soap:operation soapAction="urn:getDeviceData"/>
    <wsdl:input>
        <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
<soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ClientException">
        <soap:fault name="ClientException" use="literal"/>
    </wsdl:fault>
```


PUBLIC, SMARTSANTANDER PROJECT

```
</wsdl:fault>
  <wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getConcentratorData">
  <soap:operation soapAction="urn:getConcentratorData"/>
  <wsdl:input>
    <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
    <soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="ServerException">
    <soap:fault name="ServerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="addDataByService">
  <soap:operation soapAction="urn:addDataByService"/>
  <wsdl:input>
    <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
    <soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="ClientException">
    <soap:fault name="ClientException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="ServerException">
```

```
        <soap:fault name="ServerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="updateDataByService">
    <soap:operation soapAction="urn:updateDataByService"/>
    <wsdl:input>
        <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
        <soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ClientException">
        <soap:fault name="ClientException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="ServerException">
        <soap:fault name="ServerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="deleteDataByService">
    <soap:operation soapAction="urn:deleteDataByService"/>
    <wsdl:input>
        <soap:header message="uch:SimpleOAuthHeader" part="simpleOAuthHeader" use="literal"/>
        <soap:header message="utih:TransactionInfoHeader" part="transactionInfoHeader" use="literal"/>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ClientException">
        <soap:fault name="ClientException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="ServerException">
        <soap:fault name="ServerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>
```

PUBLIC, SMARTSANTANDER PROJECT

```
</wsdl:operation>

</wsdl:binding>

<wsdl:service name="M2MSensorDataService">
  <wsdl:port name="SensorData" binding="m2m_sdata_s1:SensorDataSOAPBinding">
    <soap:address location="http://localhost/UNICA_SDP/M2M/Sensordata"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

UNICA_API_SOAP_m2m_sensordata_types_v1_1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- January, 2011 -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:m2m_sdata_s1t="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/sensordata/v1/types"
xmlns:uct="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
targetNamespace="http://www.telefonica.com/schemas/UNICA/SOAP/m2m/sensordata/v1/types" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xsd:import namespace="http://www.telefonica.com/schemas/UNICA/SOAP/common/v1"
schemaLocation="UNICA_API_SOAP_common_types_v1_0.xsd"/>
  <!-- Business data -->
  <xsd:complexType name="ServiceIdType">
    <xsd:sequence>
      <xsd:element name="logicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ConcentratorIdType">
    <xsd:sequence>
      <xsd:element name="logicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SimType">
    <xsd:sequence>
      <xsd:element name="sim" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SimsType">
    <xsd:sequence>
```

PUBLIC, SMARTSANTANDER PROJECT

```

        <xsd:element name="sim" type="m2m_sdata_s1t:SimType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UniversalConcentratorType">
    <xsd:sequence>
        <xsd:element name="universalConcentrator" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="UniversalConcentratorsType">
    <xsd:sequence>
        <xsd:element name="universalConcentrator" type="m2m_sdata_s1t:UniversalConcentratorType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AuthenticationProfileType">
    <xsd:sequence>
        <xsd:element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="password" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="NetAuthenticationProfilesType">
    <xsd:sequence>
        <xsd:element name="netAuthenticationProfile" type="m2m_sdata_s1t:AuthenticationProfileType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LogicalGroupIidType">
    <xsd:sequence>
        <xsd:element name="logicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DeviceIidType">
    <xsd:sequence>
        <xsd:element name="globalIdentifier" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

```

PUBLIC, SMARTSANTANDER PROJECT

```

<xsd:complexType name="ConcentratorType">
  <xsd:sequence>
    <xsd:element name="concentratorId" type="m2m_sdata_s1t:ConcentratorIdType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="logicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="location" type="m2m_sdata_s1t:LocationType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="ipAddress" type="uct:IpAddressType" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ConcentratorsType">
  <xsd:sequence>
    <xsd:element name="concentrator" type="m2m_sdata_s1t:ConcentratorType" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DeviceType">
  <xsd:sequence>
    <xsd:element name="deviceId" type="m2m_sdata_s1t:DeviceIdType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="creationTime" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="registrationTime" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="status" type="xsd:int" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DevicesType">
  <xsd:sequence>
    <xsd:element name="device" type="m2m_sdata_s1t:DeviceType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LogicalGroupType">
  <xsd:sequence>
    <xsd:element name="logicalName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LogicalGroupsType">
  <xsd:sequence>

```

PUBLIC, SMARTSANTANDER PROJECT

```

                <xsd:element name="logicalGroup" type="m2m_sdata_s1t:LogicalGroupType" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="BusinessDeviceType">
            <xsd:sequence>
                <xsd:element name="deviceURN" type="xsd:string" minOccurs="1" maxOccurs="1"/>
                <xsd:element name="typeOfURN" type="xsd:int" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="detailedBusinessDevice" type="m2m_sdata_s1t:TypedAvPairType" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="BusinessDevicesType">
            <xsd:sequence>
                <xsd:element name="businessDevice" type="m2m_sdata_s1t:BusinessDeviceType" minOccurs="0"
maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:simpleType name="AggregatorType">
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="Max"/>
                <xsd:enumeration value="Min"/>
                <xsd:enumeration value="Average"/>
                <xsd:enumeration value="Count"/>
                <xsd:enumeration value="Sum"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="AggregatorExtendedType">
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                    <xsd:attribute name="aggregator" type="m2m_sdata_s1t:AggregatorType" use="optional"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
        <xsd:complexType name="AttributesType">
    
```

PUBLIC, SMARTSANTANDER PROJECT

```
<xsd:sequence>
  <xsd:element name="name" type="m2m_sdata_s1t:AggregatorExtendedType" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ServiceAttributesType">
  <xsd:sequence>
    <xsd:element name="businessDevice" type="m2m_sdata_s1t:AttributesType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="detailedNotice" type="m2m_sdata_s1t:AttributesType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AvPairType">
  <xsd:sequence>
    <xsd:element name="name" type="m2m_sdata_s1t:AggregatorExtendedType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="DataType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Integer"/>
    <xsd:enumeration value="String"/>
    <xsd:enumeration value="Float"/>
    <xsd:enumeration value="SerialNumber"/>
    <xsd:enumeration value="Time"/>
    <xsd:enumeration value="Date"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="TypedAvPairType">
  <xsd:complexContent>
    <xsd:extension base="m2m_sdata_s1t:AvPairType">
      <xsd:sequence>
        <xsd:element name="type" type="m2m_sdata_s1t:DataType" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

PUBLIC, SMARTSANTANDER PROJECT

```
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="FilterAvPairType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="operator" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FilterAttributesType">
  <xsd:sequence>
    <xsd:element name="attribute" type="m2m_sdata_s1t:FilterAvPairType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FromToType">
  <xsd:sequence>
    <xsd:element name="from" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="to" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FilterType">
  <xsd:sequence>
    <xsd:element name="from" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="to" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="attributes" type="m2m_sdata_s1t:FilterAttributesType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FilterAccumulateDateType">
  <xsd:sequence>
    <xsd:element name="from" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="to" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```


PUBLIC, SMARTSANTANDER PROJECT

```

        <xsd:element name="dateAttributeName" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MeasureDataType">
    <xsd:sequence>
        <xsd:element name="attribute" type="m2m_sdata_s1t:AvPairType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MeasureType">
    <xsd:sequence>
        <xsd:element name="date" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="data" type="m2m_sdata_s1t:MeasureDataType" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MeasuresFilterType">
    <xsd:sequence>
        <xsd:element name="filter" type="m2m_sdata_s1t:FilterType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="attributes" type="m2m_sdata_s1t:AttributesType" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AccumulatedMeasuresFilterType">
    <xsd:sequence>
        <xsd:element name="filter" type="m2m_sdata_s1t:FromToType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="detailedNoticeAttribute" type="m2m_sdata_s1t:AttributesType" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AccumulatedMeasurementsType">
    <xsd:sequence>
        <xsd:element name="accumulatedMeasurement" type="m2m_sdata_s1t:MeasureType" minOccurs="0"
maxOccurs="unbounded"/>

```

PUBLIC, SMARTSANTANDER PROJECT

```

        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AccumulatedMeasurementsFilterType">
        <xsd:sequence>
            <xsd:element name="businessDeviceAttribute" type="m2m_sdata_s1t:AvPairType" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="filterDate" type="m2m_sdata_s1t:FilterAccumulateDateType" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="interval" type="m2m_sdata_s1t:IntervalType" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="attributes" type="m2m_sdata_s1t:AttributesType" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="LocationType">
        <xsd:sequence>
            <xsd:element name="latitude" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="longitude" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="IntervalType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Hour"/>
            <xsd:enumeration value="Day"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- Types of operations -->
    <xsd:element name="getDevicesByLogicalGroup">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="logicalGroup" type="m2m_sdata_s1t:LogicalGroupIDType" minOccurs="1"
maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

```

PUBLIC, SMARTSANTANDER PROJECT

```

<xsd:element name="getDevicesByLogicalGroupResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="device" type="m2m_sdata_s1t:DeviceType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="getDataByService">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
      <xsd:choice>
        <xsd:element name="devices" type="xsd:anyType"/>
        <xsd:element name="concentrators" type="xsd:anyType"/>
        <xsd:element name="logicalGroups" type="xsd:anyType"/>
        <xsd:element name="businessDevices" type="xsd:anyType"/>
        <xsd:element name="serviceAttributes" type="xsd:anyType"/>
      </xsd:choice>
      <xsd:element name="accumulatedMeasurements" type="m2m_sdata_s1t:AccumulatedMeasurementsFilterType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="getDataByServiceResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
      <xsd:choice>
        <xsd:element name="devices" type="m2m_sdata_s1t:DevicesType"/>
        <xsd:element name="concentrators" type="m2m_sdata_s1t:ConcentratorsType"/>
        <xsd:element name="logicalGroups" type="m2m_sdata_s1t:LogicalGroupsType"/>
        <xsd:element name="businessDevices" type="m2m_sdata_s1t:BusinessDevicesType"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

PUBLIC, SMARTSANTANDER PROJECT

```

<xsd:element name="serviceAttributes" type="m2m_sdata_s1t:ServiceAttributesType"/>
    <xsd:element
type="m2m_sdata_s1t:AccumulatedMeasurementsType"/>
        name="accumulatedMeasurements"
    </xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="getDeviceData">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="device" type="m2m_sdata_s1t:DeviceIDType" minOccurs="1" maxOccurs="1"/>
                    <xsd:choice>
                        <xsd:element name="sensorML" type="xsd:anyType"/>
                        <xsd:element name="lastMeasure" type="xsd:anyType"/>
                        <xsd:element name="measures" type="m2m_sdata_s1t:MeasuresFilterType"/>
                    </xsd:choice>
                    <xsd:element
type="m2m_sdata_s1t:AccumulatedMeasuresFilterType"/>
                        name="accumulatedMeasurements"
                    </xsd:choice>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="getDeviceDataResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
                    <xsd:element name="device" type="m2m_sdata_s1t:DeviceIDType" minOccurs="1" maxOccurs="1"/>
                    <xsd:choice>
                        <xsd:element name="sensorML" type="xsd:string"/>
                        <xsd:element
maxOccurs="unbounded"/>
                            name="measure"
                            type="m2m_sdata_s1t:MeasureType"
                            minOccurs="0"
                        </xsd:element>
                    </xsd:choice>
                    <xsd:element
type="m2m_sdata_s1t:AccumulatedMeasurementsType"/>
                        name="accumulatedMeasurements"
                    </xsd:choice>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

```

PUBLIC, SMARTSANTANDER PROJECT

```

</xsd:element>

<xsd:element name="getConcentratorData">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="concentratorID" type="m2m_sdata_s1t:ConcentratorIDType" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="getConcentratorDataResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="concentrator" type="m2m_sdata_s1t:ConcentratorType" minOccurs="1"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="addDataByService">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
      <xsd:choice>
        <xsd:element name="sims" type="m2m_sdata_s1t:SimsType"/>
        <xsd:element name="universalConcentrators" type="m2m_sdata_s1t:UniversalConcentratorsType"/>
        <xsd:element name="netAuthenticationProfiles" type="m2m_sdata_s1t:NetAuthenticationProfilesType"/>
        <xsd:element name="businessDevices" type="m2m_sdata_s1t:BusinessDevicesType"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="addDataByServiceResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>

```

PUBLIC, SMARTSANTANDER PROJECT

```
<xsd:choice>
    <xsd:element name="sims" type="xsd:anyType"/>
    <xsd:element name="universalConcentrators" type="xsd:anyType"/>
    <xsd:element name="netAuthenticationProfiles" type="xsd:anyType"/>
    <xsd:element name="businessDevices" type="xsd:anyType"/>
    </xsd:choice>
    <xsd:element name="errorCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="errorText" type="xsd:string" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="updateDataByService">
    <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
    <xsd:choice>
    <xsd:element name="sims" type="m2m_sdata_s1t:SimsType"/>
    <xsd:element name="universalConcentrators" type="m2m_sdata_s1t:UniversalConcentratorsType"/>
    <xsd:element name="netAuthenticationProfiles" type="m2m_sdata_s1t:NetAuthenticationProfilesType"/>
    <xsd:element name="businessDevices" type="m2m_sdata_s1t:BusinessDevicesType"/>
    </xsd:choice>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>

    <xsd:element name="updateDataByServiceResponse">
    <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
    <xsd:choice>
    <xsd:element name="sims" type="xsd:anyType"/>
    <xsd:element name="universalConcentrators" type="xsd:anyType"/>
    <xsd:element name="netAuthenticationProfiles" type="xsd:anyType"/>
    <xsd:element name="businessDevicess" type="xsd:anyType"/>
    </xsd:choice>
    <xsd:element name="errorCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    </xsd:complexType>
    </xsd:element>
</xsd:element>
</xsd:complexType>
</xsd:element>
```

PUBLIC, SMARTSANTANDER PROJECT

```
<xsd:element name="errorText" type="xsd:string" minOccurs="1" maxOccurs="1"/>

</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="deleteDataByService">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
<xsd:choice>
<xsd:element name="sims" type="m2m_sdata_s1t:SimsType"/>
<xsd:element name="universalConcentrators" type="m2m_sdata_s1t:UniversalConcentratorsType"/>
<xsd:element name="netAuthenticationProfiles" type="m2m_sdata_s1t:NetAuthenticationProfilesType"/>
<xsd:element name="businessDevices" type="m2m_sdata_s1t:BusinessDevicesType"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="deleteDataByServiceResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="serviceID" type="m2m_sdata_s1t:ServiceIDType" minOccurs="0" maxOccurs="1"/>
<xsd:choice>
<xsd:element name="sims" type="xsd:anyType"/>
<xsd:element name="universalConcentrators" type="xsd:anyType"/>
<xsd:element name="netAuthenticationProfiles" type="xsd:anyType"/>
<xsd:element name="businessDevices" type="xsd:anyType"/>
</xsd:choice>
<xsd:element name="errorCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
<xsd:element name="errorText" type="xsd:string" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```



PUBLIC, SMARTSANTANDER PROJECT

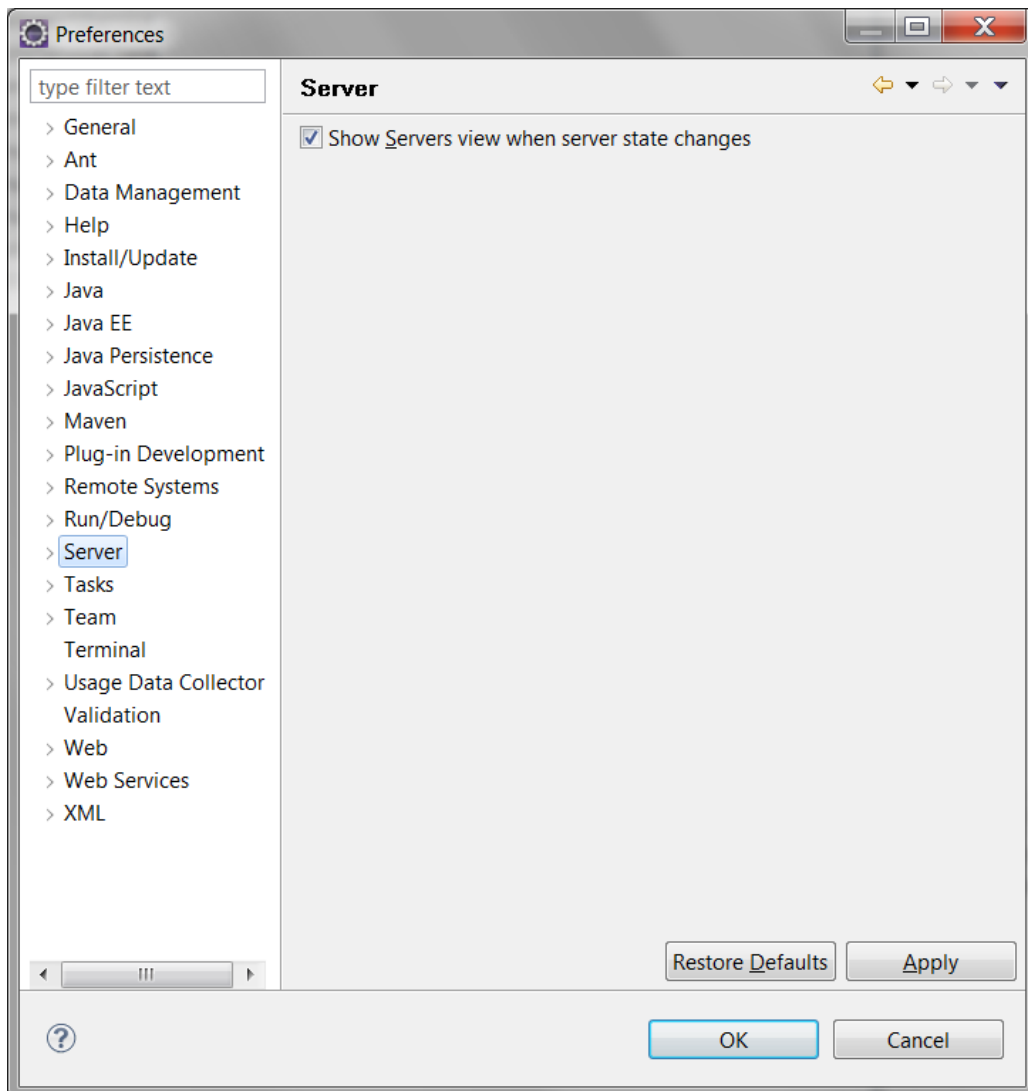
6. Building a service using Eclipse IDE

The minimum software requirements needed are the followings:

- JDK 1.5 or later
- Eclipse for JEE developers
- Ant version 1.6.5 or higher
- Apache Tomcat (version 6)
- Axis2

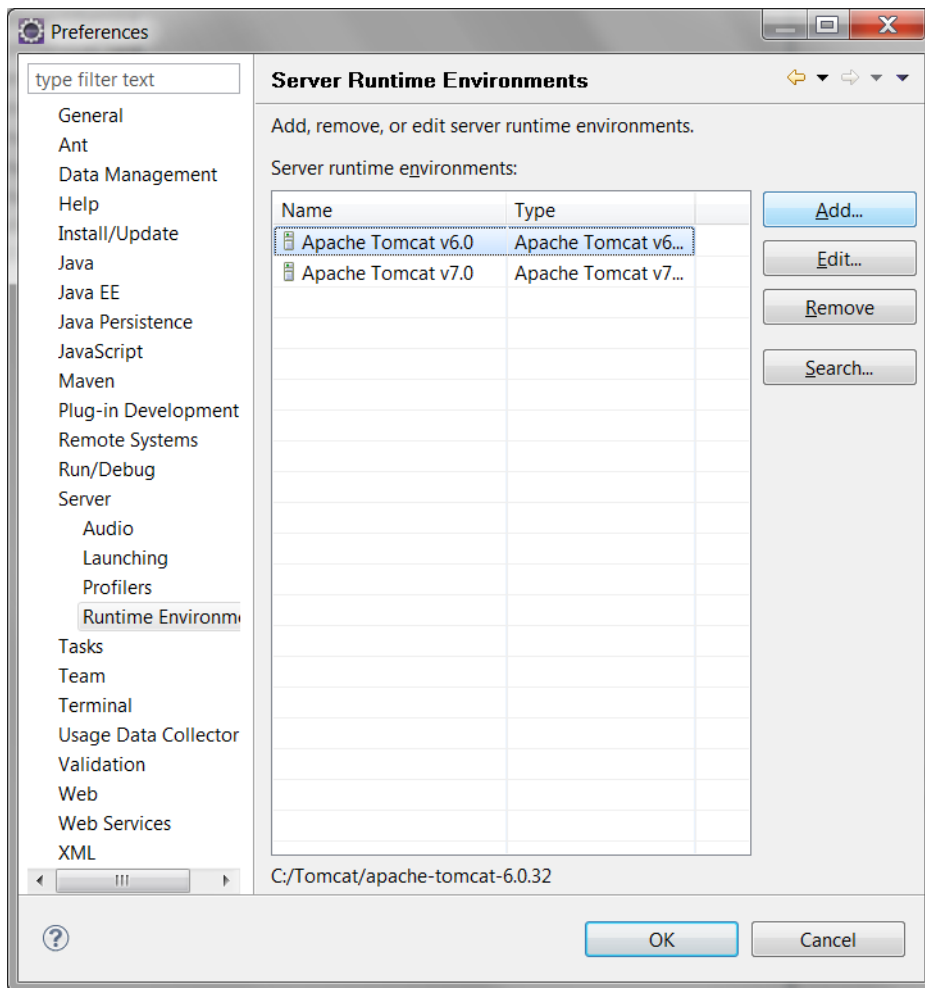
First of all you will need to setup your Eclipse to use Tomcat and Axis2 home directories.

To do that, go under Window->Preferences:



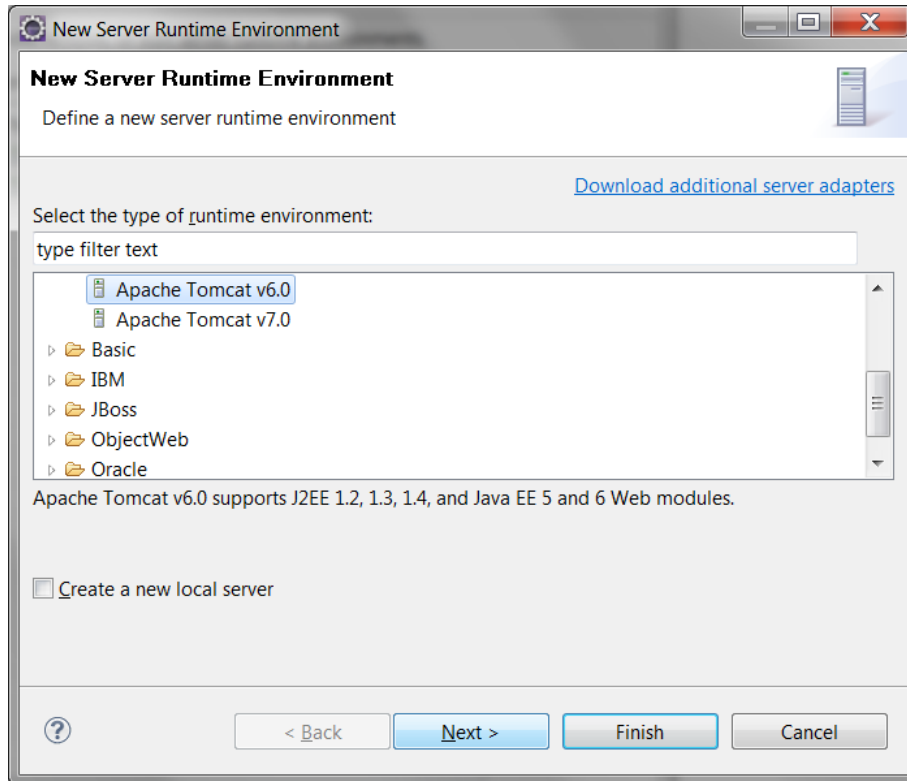
For setting up Tomcat click on Server->Runtime Environments and click Add...

PUBLIC, SMARTSANTANDER PROJECT

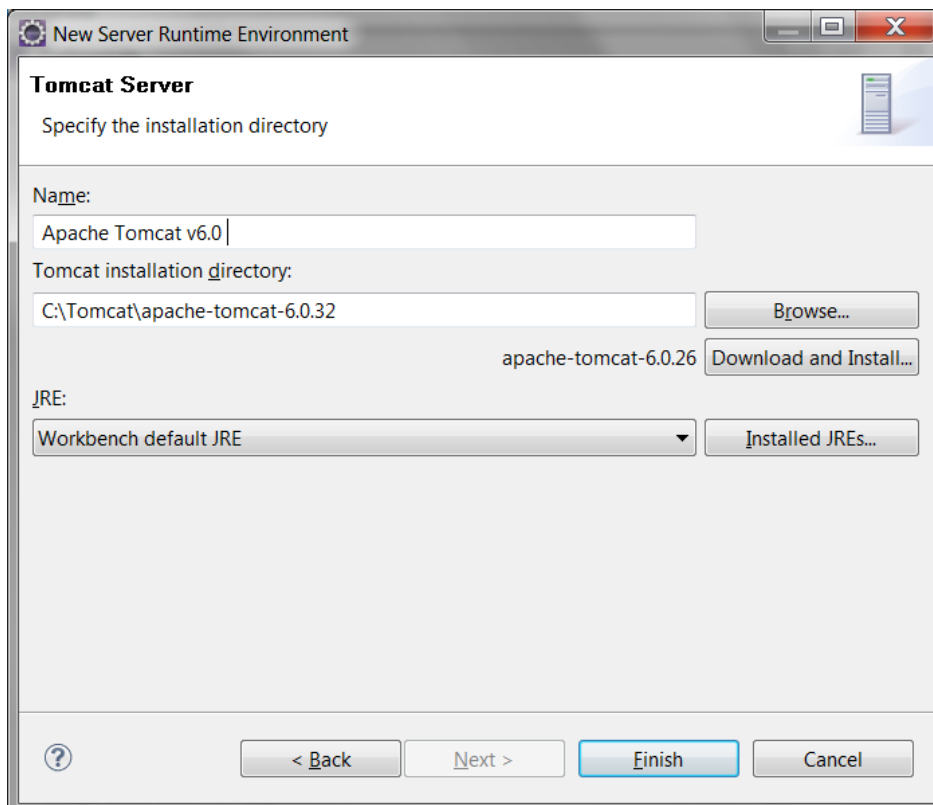


Under Apache select the Tomcat version you have previously installed on your machine and click Next.

PUBLIC, SMARTSANTANDER PROJECT

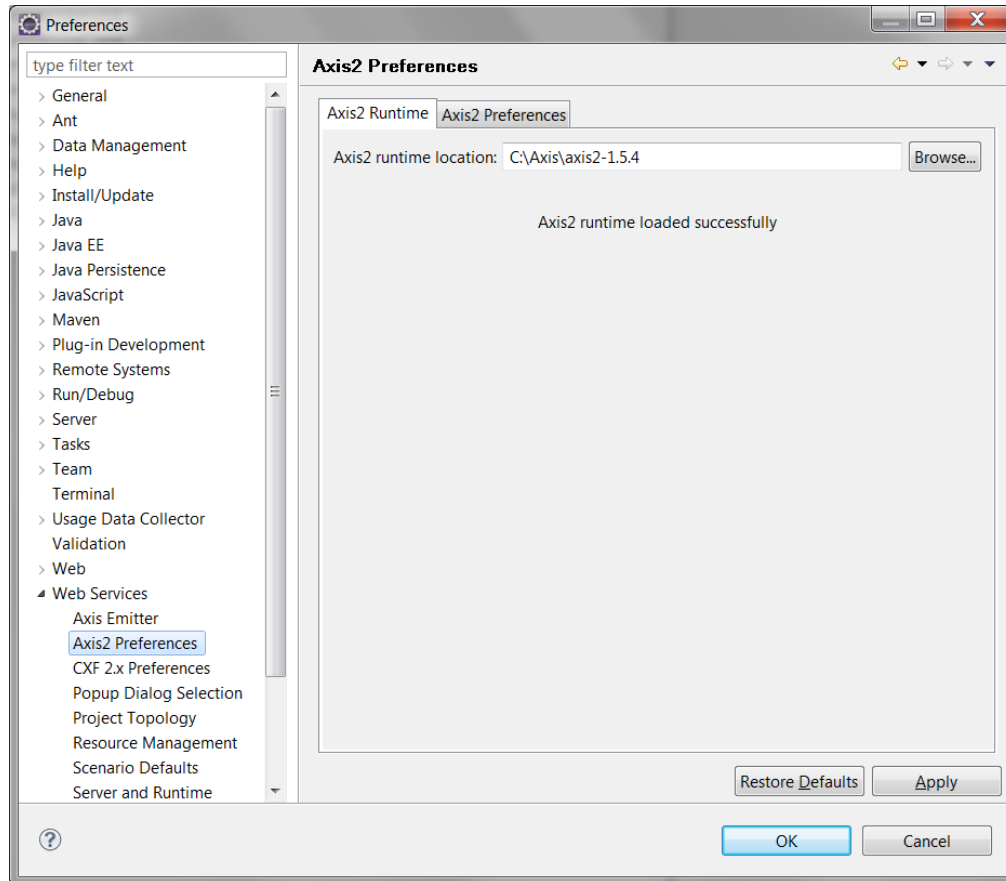


Under Tomcat installation directory browse the home folder of your Tomcat installation and click Finish.

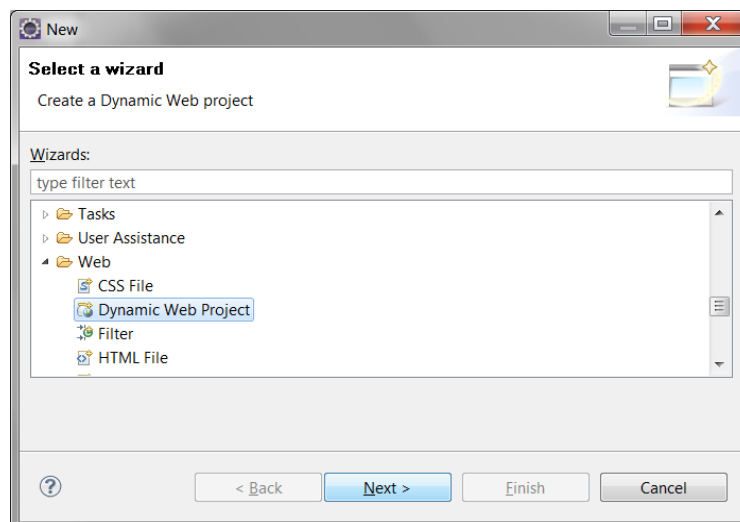


PUBLIC, SMARTSANTANDER PROJECT

The next step is to configure the Axis2, go again to Window->Preferences, select Webservices>Axis2 Preferences and specify the home directory of your Axis2 installation. Finally click on Apply and OK.

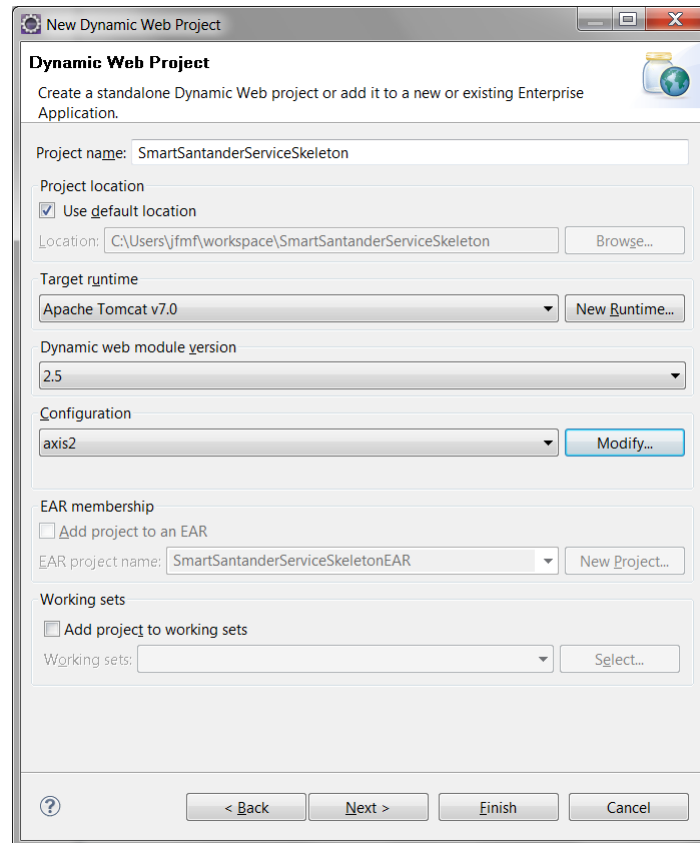


Both Tomcat and Axis2 are now configured in your Eclipse installation. The next step is to create the project, go under File->New->Other... under Web select Dynamic Web Project and click Next.

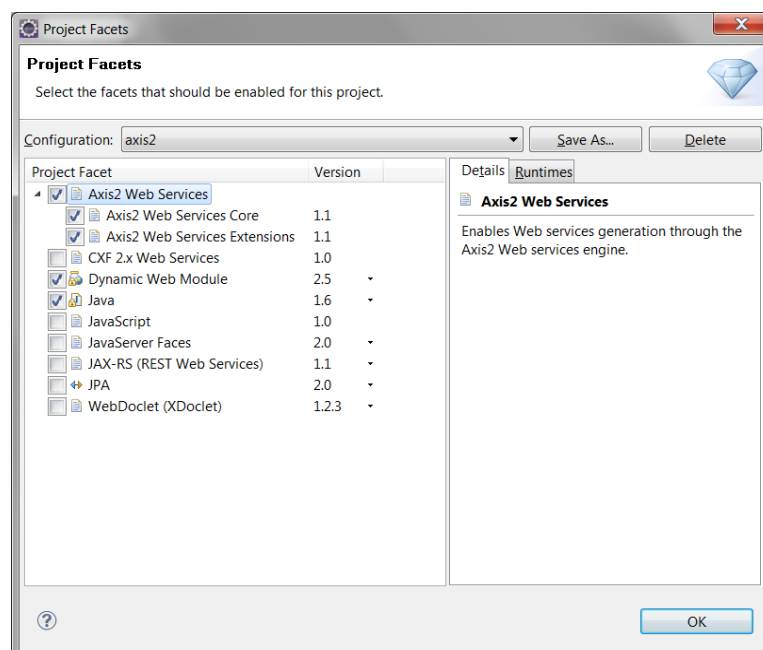


PUBLIC, SMARTSANTANDER PROJECT

Write a project name, set the target runtime to the Apache Tomcat version you have and set the Dynamic web module version to 2.5. On the Configuration click on Modify...



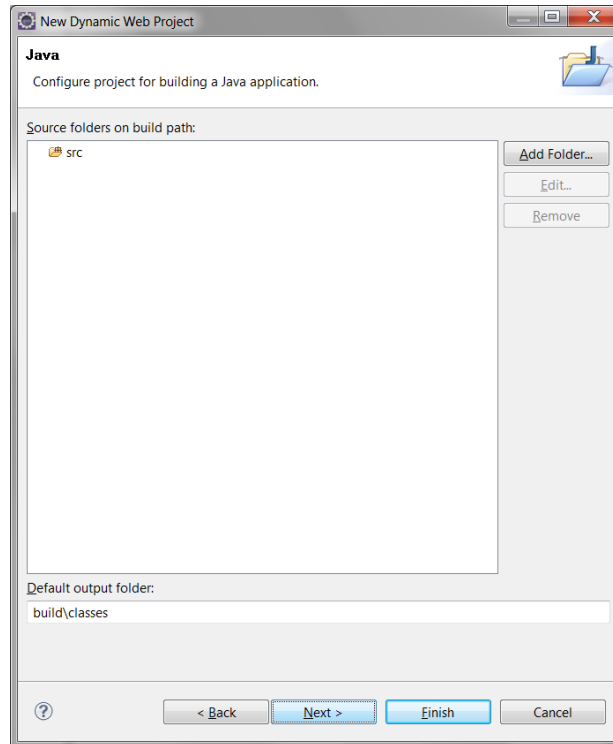
Select the Axis2 Web Services and click Save As... giving a name to this new configuration. Click on Next.



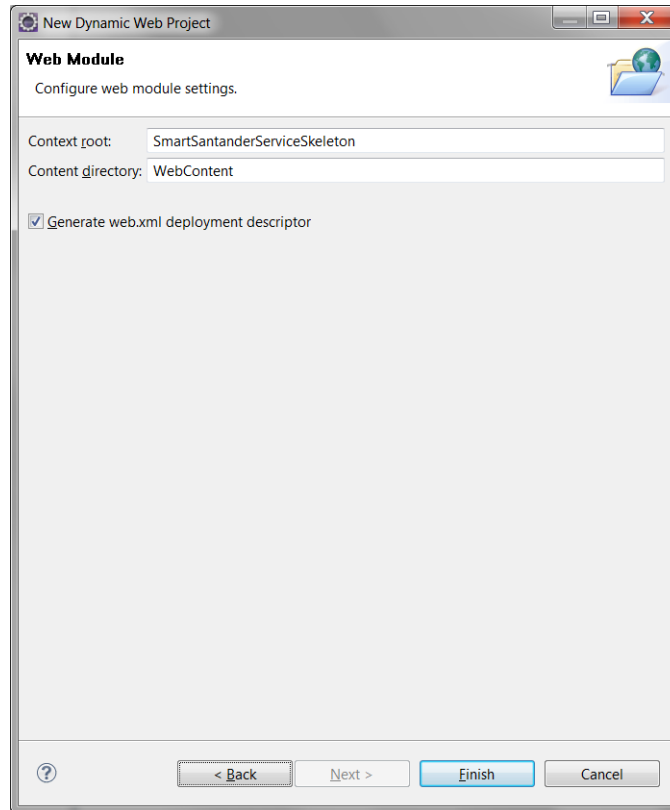


PUBLIC, SMARTSANTANDER PROJECT

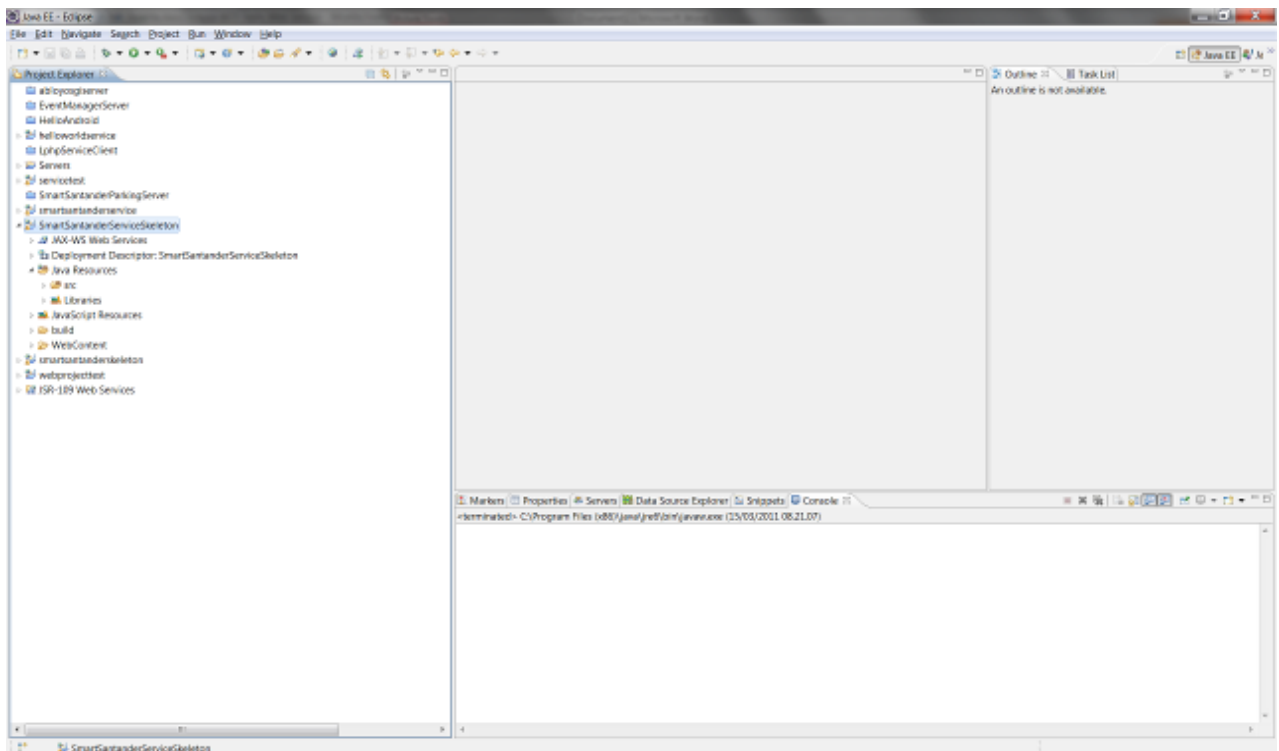
The Next step is to set the source and output folders for the project. In this case we will use the default configuration, so click Next.



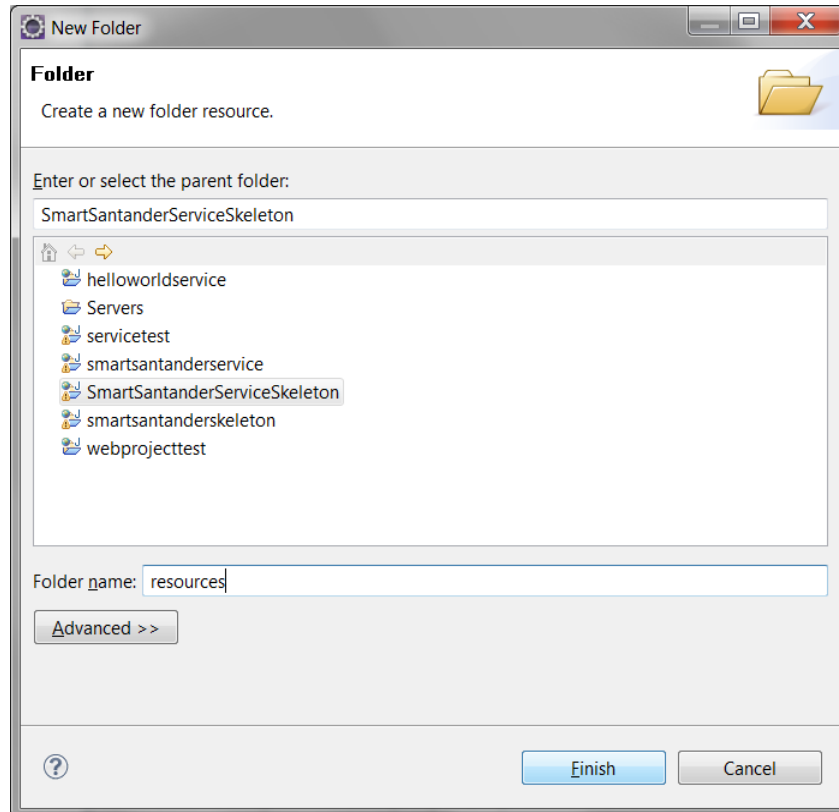
The final step is to configure the web module settings we will use also the default configuration, with the Context root with the name of the project and the content directory is WebContent. Click on Finish.



The `SmartSantanderServiceSkeleton` is now created.



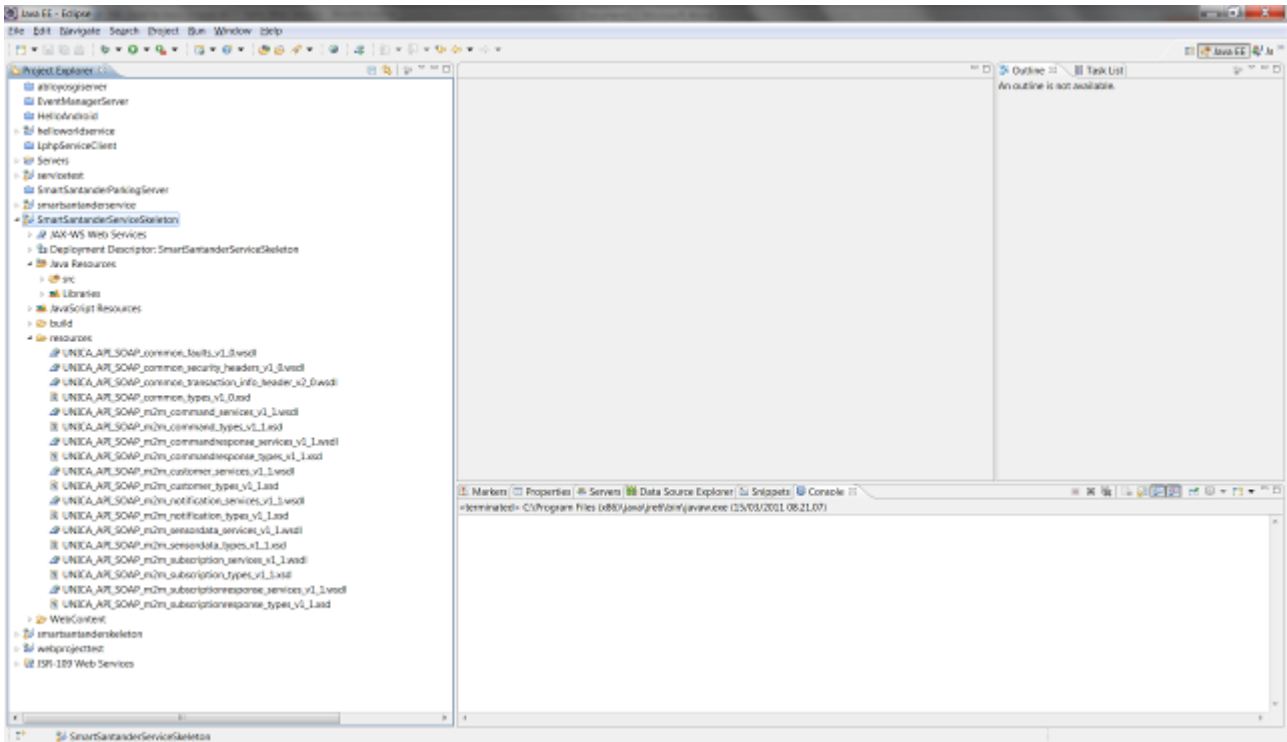
The first step is to create a new folder inside the project called resources and place all the wsdl and xsd files of the UNICA platform.



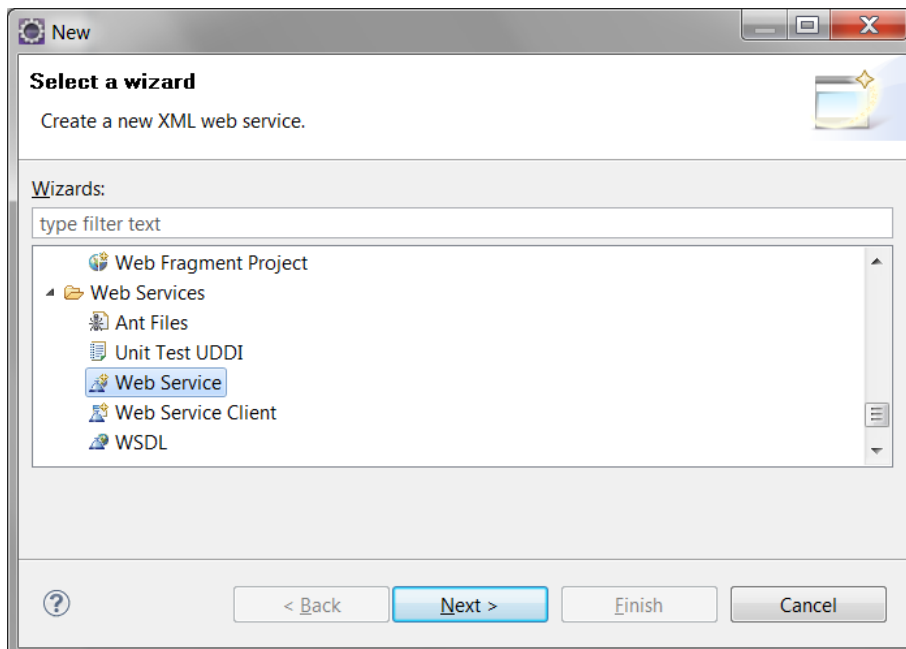
Now that we have the web service description files in the project we will generate the web service code for some of the descriptions. For that right click on the project name and select New->Other...



PUBLIC, SMARTSANTANDER PROJECT

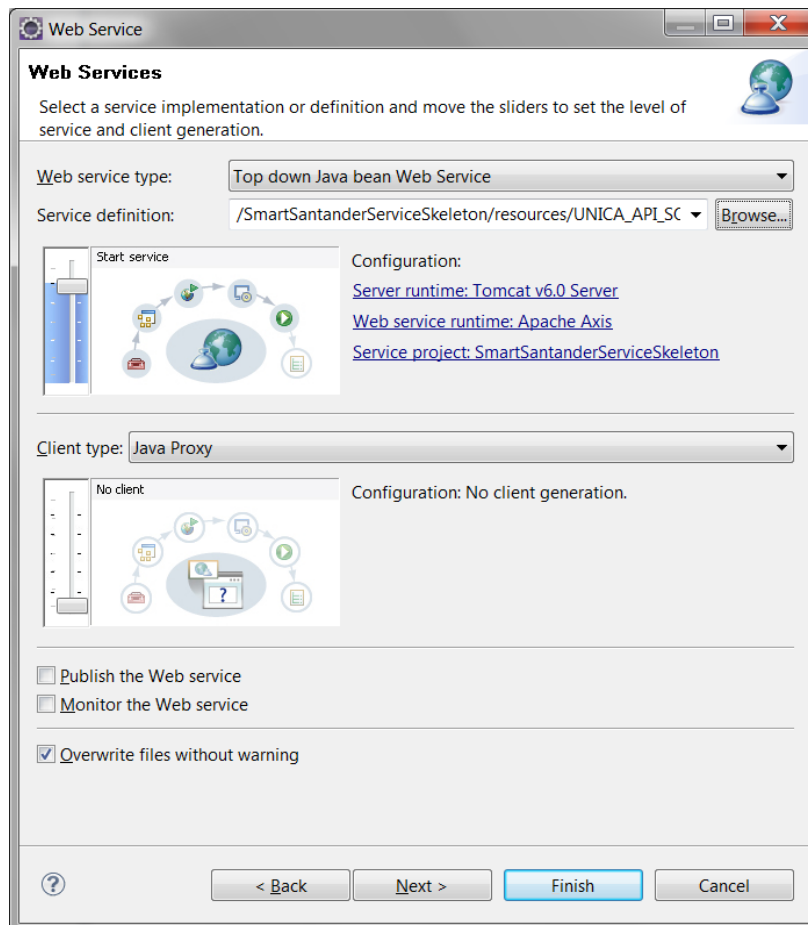


Under Web Services select Web Service and click Next.

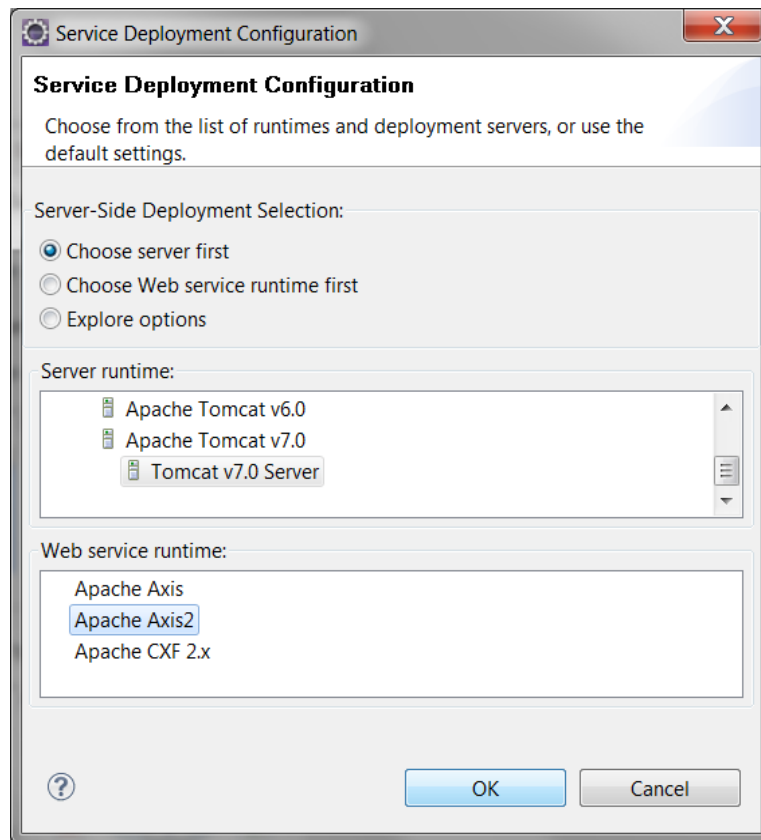


On the Web Service type select “Top down Java Bean Web Service” and browse the wsdl file for the notification service under the project resources folder. Then Click on the Server runtime.

PUBLIC, SMARTSANTANDER PROJECT

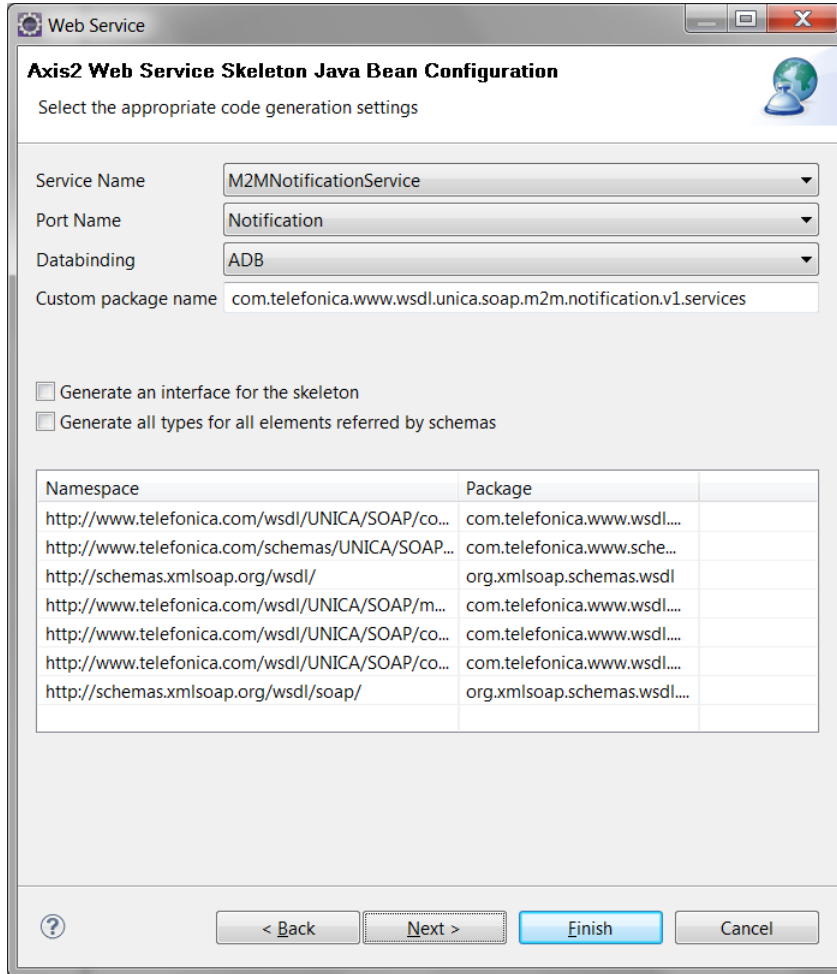


On Server runtime select the Tomcat Server with the version of Tomcat you have and on the Web service runtime select Apache Axis2, click OK. Click Next.



Select the Generate an interface for the skeleton and generate all types for all elements referred by schemas and click on Next.

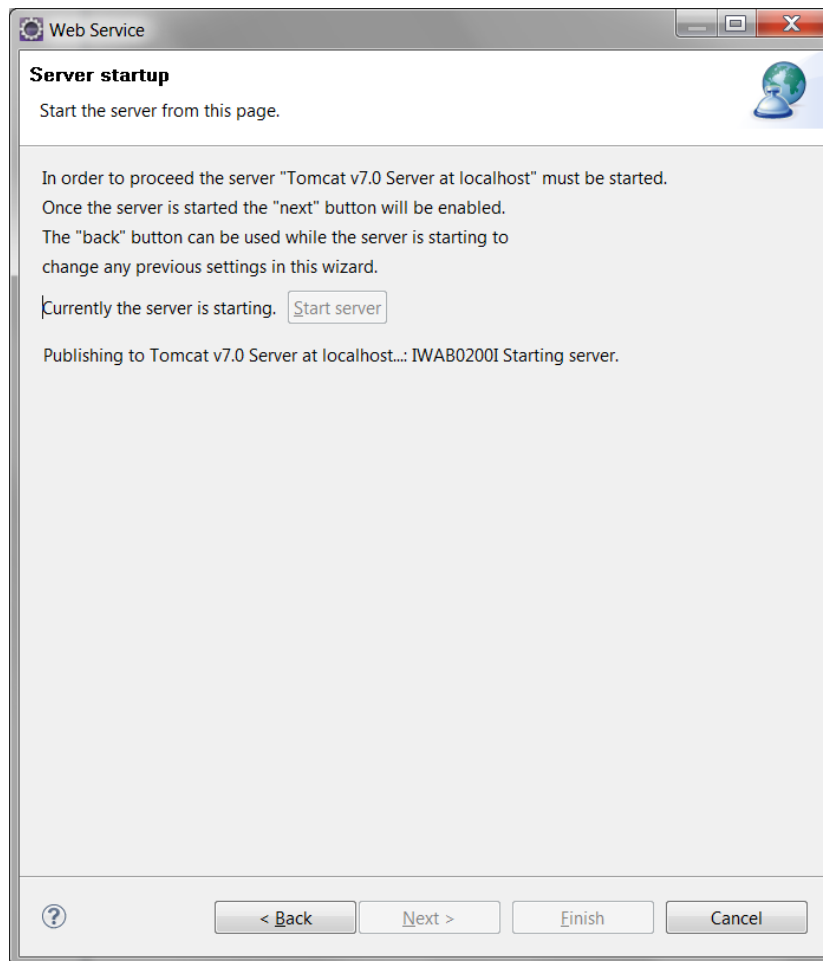
PUBLIC, SMARTSANTANDER PROJECT



The screenshot shows the 'Axis2 Web Service Skeleton Java Bean Configuration' dialog box. It includes fields for Service Name (M2MNotificationService), Port Name (Notification), Databinding (ADB), and Custom package name (com.telefonica.www.wsdl.unica.soap.m2m.notification.v1.services). There are also checkboxes for 'Generate an interface for the skeleton' and 'Generate all types for all elements referred by schemas'. A table lists namespaces and their corresponding packages.

| Namespace | Package |
|---|-----------------------------|
| http://www.telefonica.com/wsdl/UNICA/SOAP/co... | com.telefonica.www.wsdl... |
| http://www.telefonica.com/schemas/UNICA/SOAP... | com.telefonica.www.sche... |
| http://schemas.xmlsoap.org/wsdl/ | org.xmlsoap.schemas.wsdl |
| http://www.telefonica.com/wsdl/UNICA/SOAP/m... | com.telefonica.www.wsdl... |
| http://www.telefonica.com/wsdl/UNICA/SOAP/co... | com.telefonica.www.wsdl... |
| http://www.telefonica.com/wsdl/UNICA/SOAP/co... | com.telefonica.www.wsdl... |
| http://schemas.xmlsoap.org/wsdl/soap/ | org.xmlsoap.schemas.wsdl... |

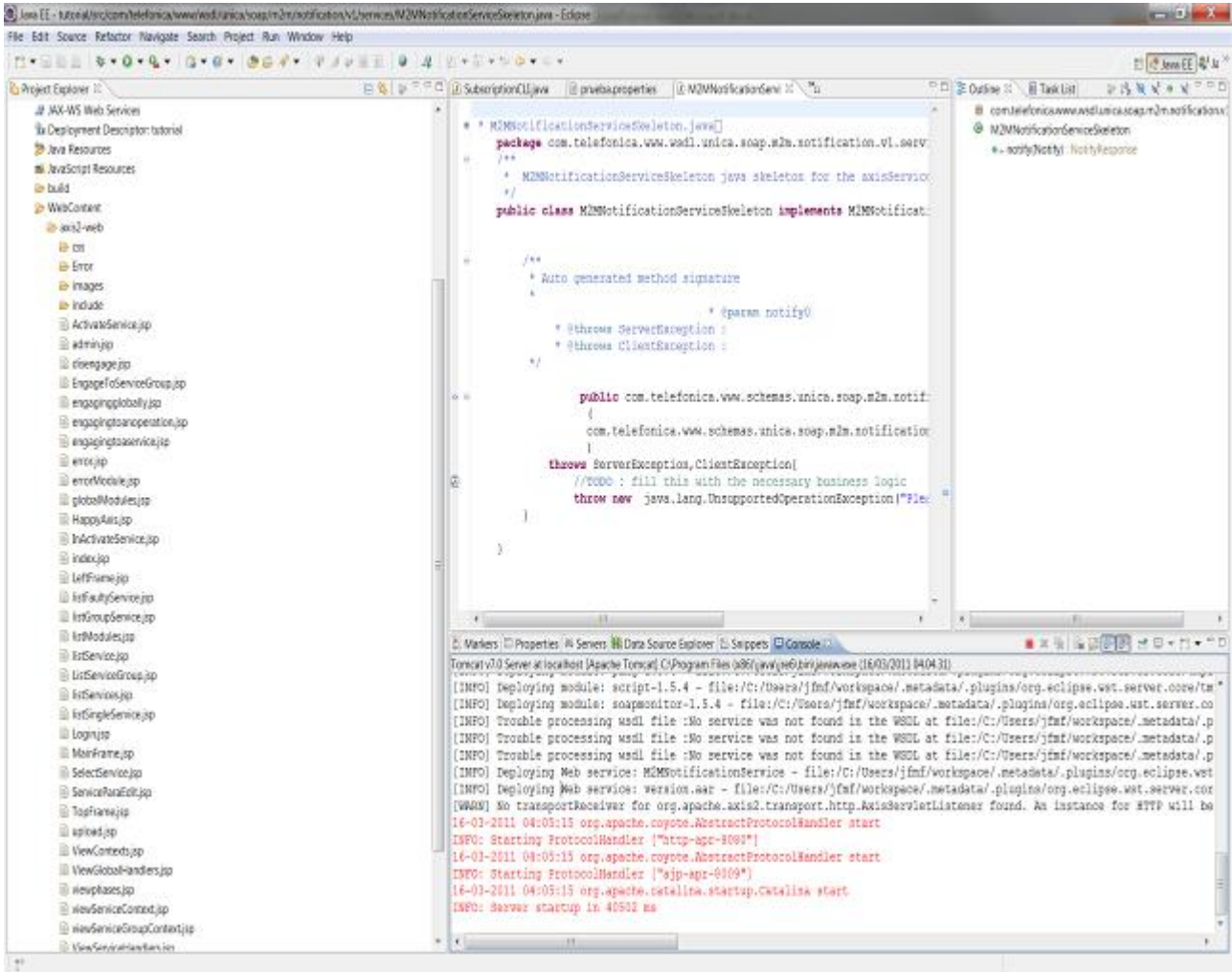
Finally click on Start server.



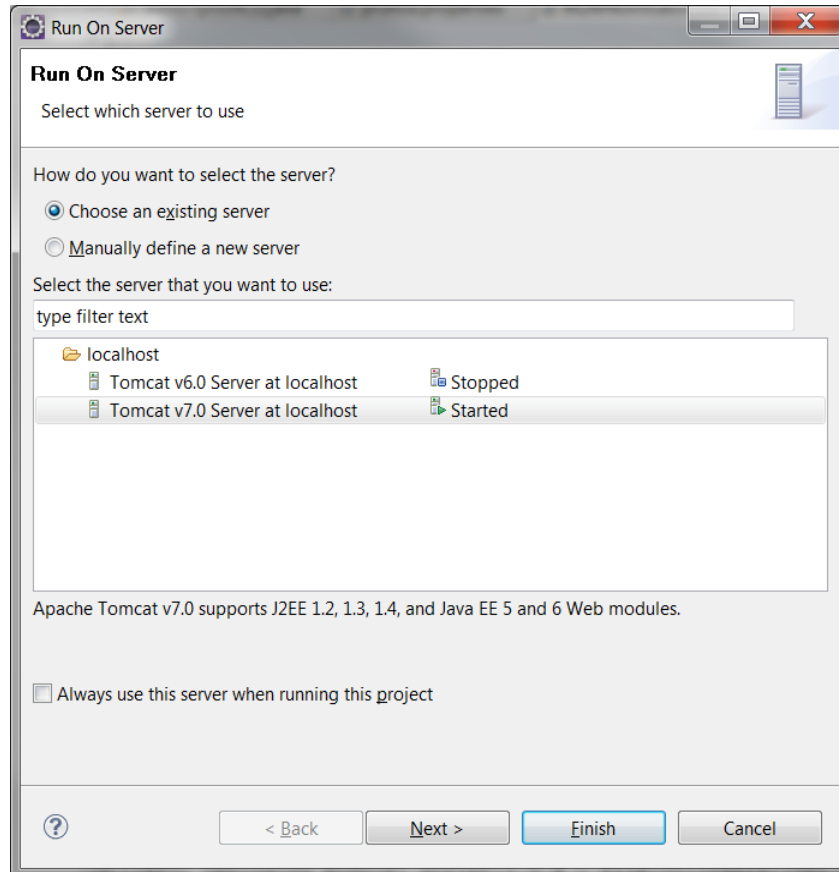
You should get some output on the Console, click Finish. Your web service should be now running. 16-03-2011 04:05:15 org.apache.coyote.AbstractProtocolHandler? start INFO: Starting ProtocolHandler? http-apr-8080? 16-03-2011 04:05:15 org.apache.coyote.AbstractProtocolHandler? start INFO: Starting ProtocolHandler? ajp-apr-8009? 16-03-2011 04:05:15 org.apache.catalina.startup.Catalina start INFO: Server startup in 40502 ms To check that the service is correctly running go to the Web Content folder->axis2-web right click on the index.jsp file and select Run As...->Run On Server.



PUBLIC, SMARTSANTANDER PROJECT



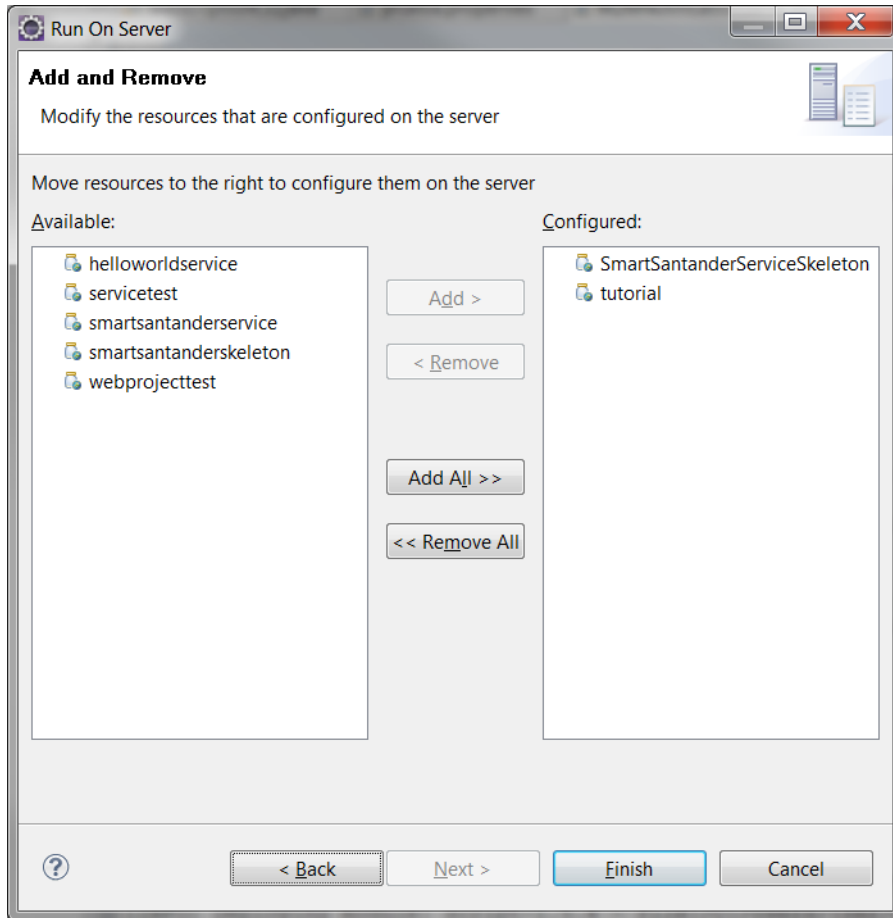
Select the server that is currently running and click Next.



Make sure your project resources are under the Configured list and click Finish.



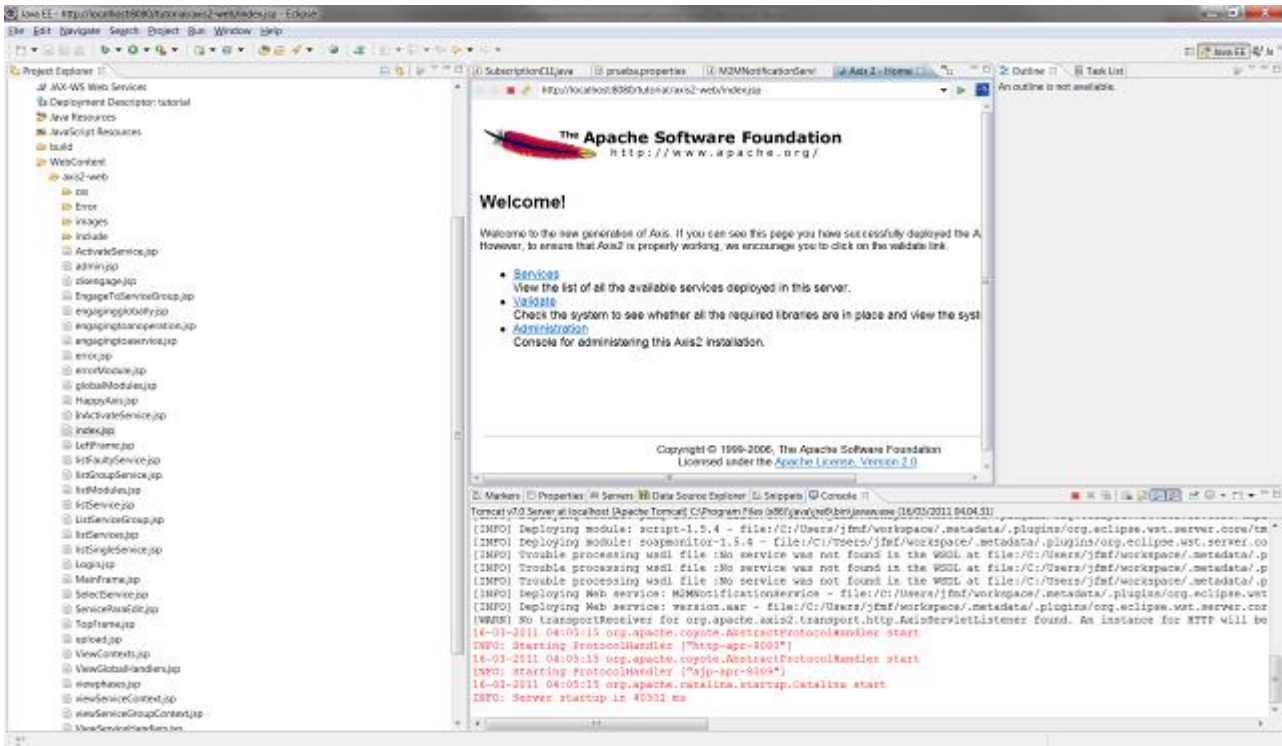
PUBLIC, SMARTSANTANDER PROJECT



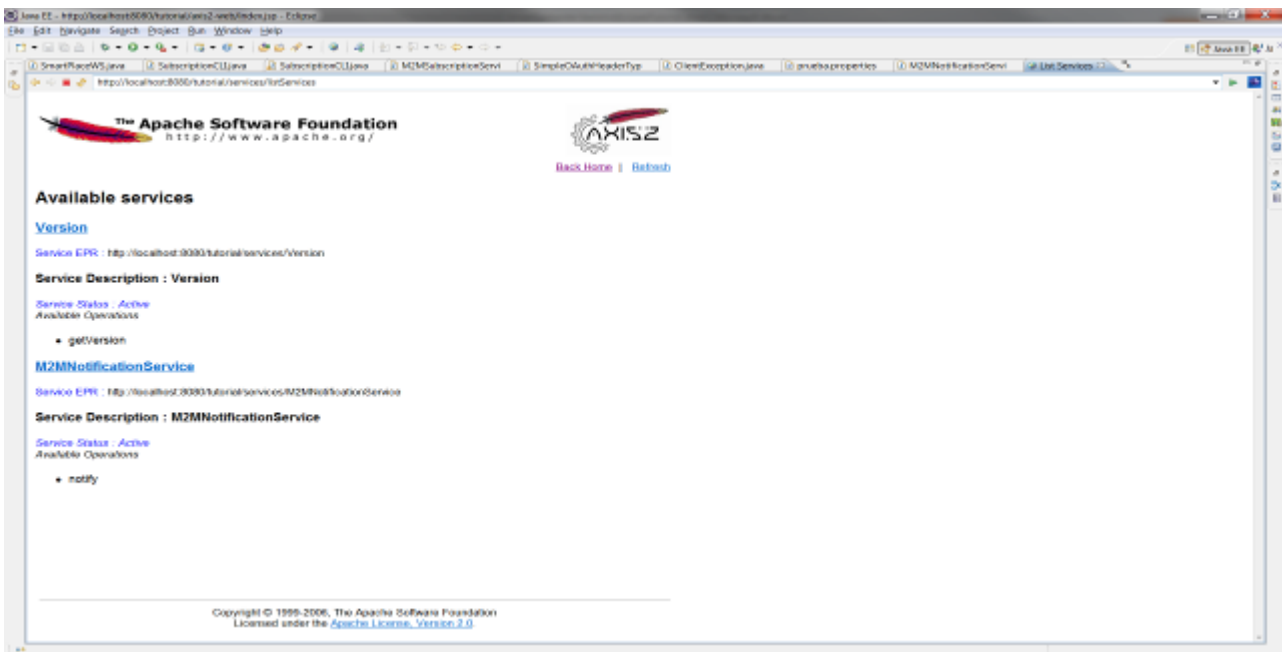
Apache's page will appear, click on Services.



PUBLIC, SMARTSANTANDER PROJECT



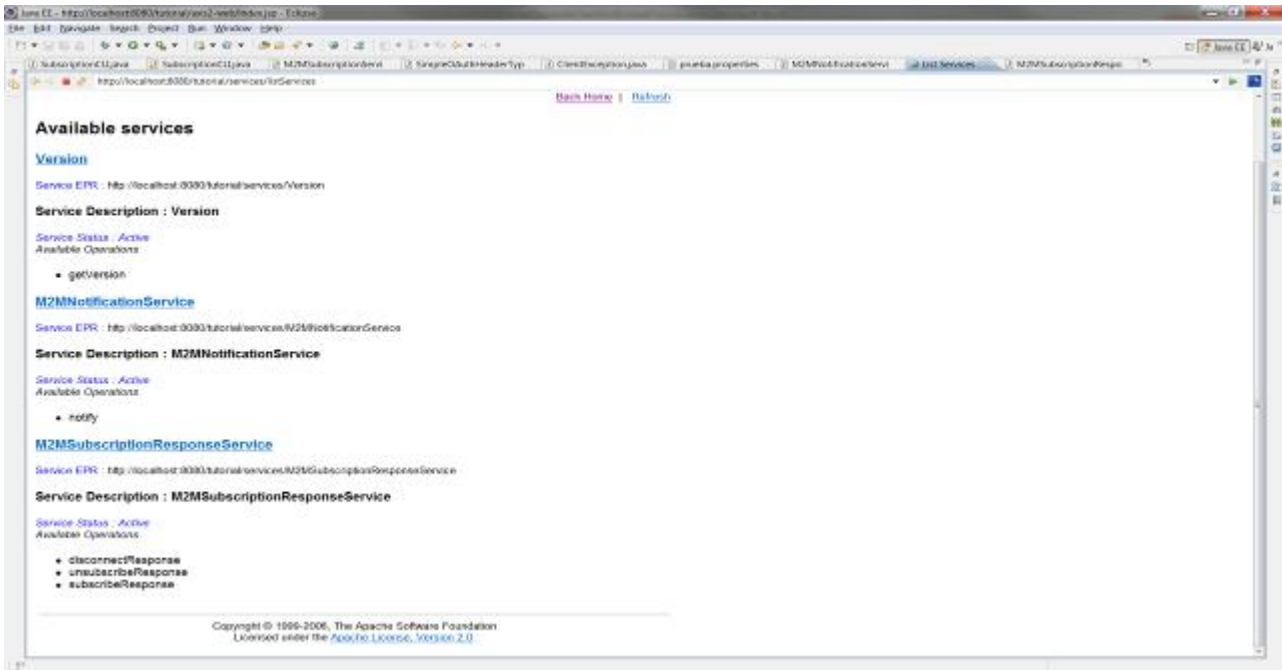
The M2MNotificationService should appear in the Services page.



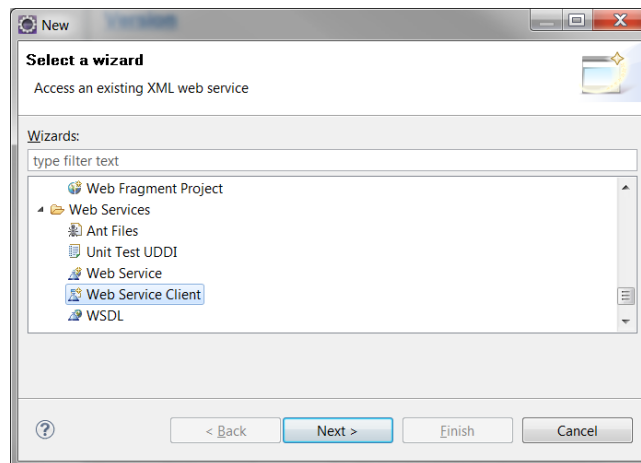
Repeat the same process of creating a web service but this time for the M2MSubscriptionResponse service and check that you now have to services deployed.



PUBLIC, SMARTSANTANDER PROJECT



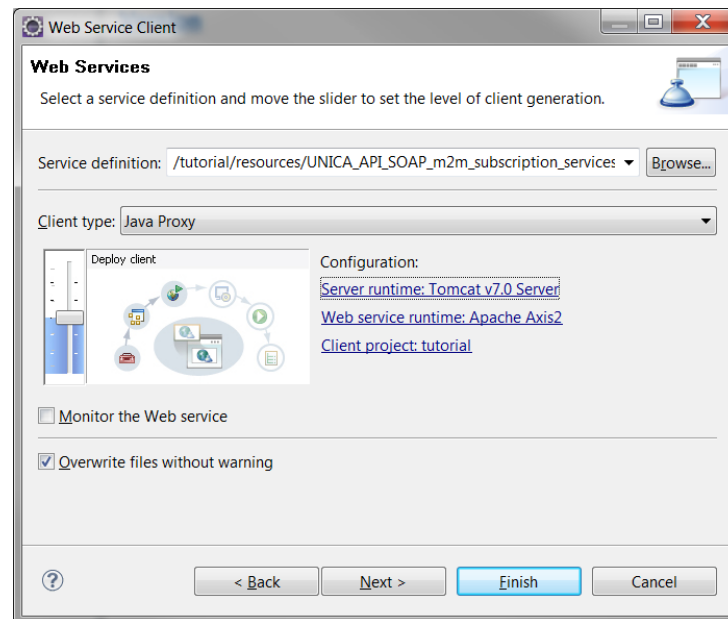
Finally we need to create a client for the M2MSubscriptionService, for that right click on the project name and select New->Other... This time select under Web Services the Web Service Client and click on Next.



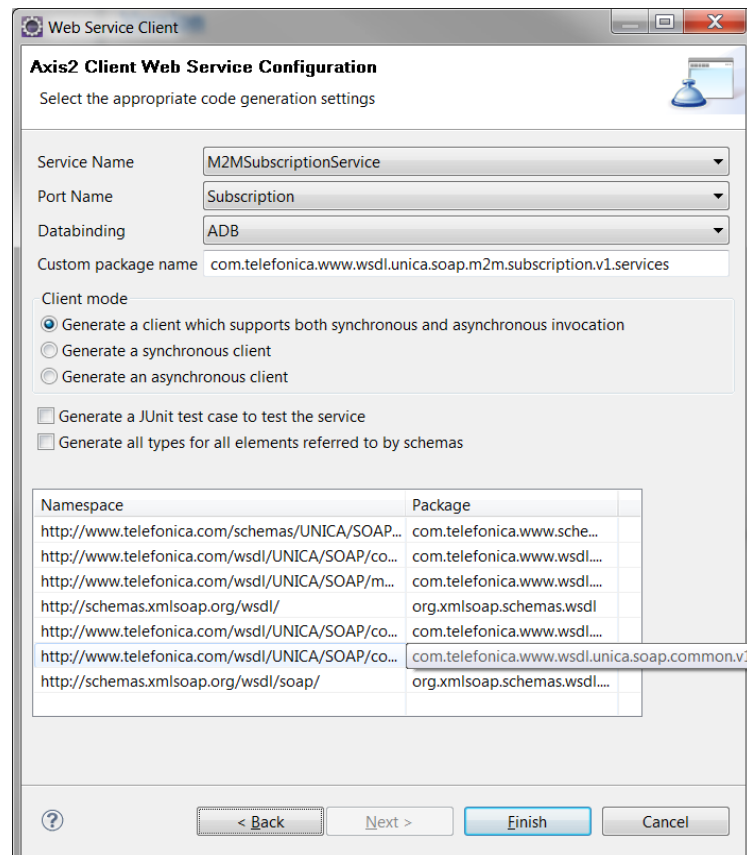
Browse the Subscription service wsdl file and select the Tomcat version and Axis2 as the service runtime. Click on Next.

D4.2 DESCRIPTION OF IMPLEMENTED IOT SERVICES

PUBLIC, SMARTSANTANDER PROJECT



Finally click on Finish.



The web service client code is now generated. The next step is to code the notify method of the notification service in the M2MNotificationServiceSkeleton and the subscription methods in the



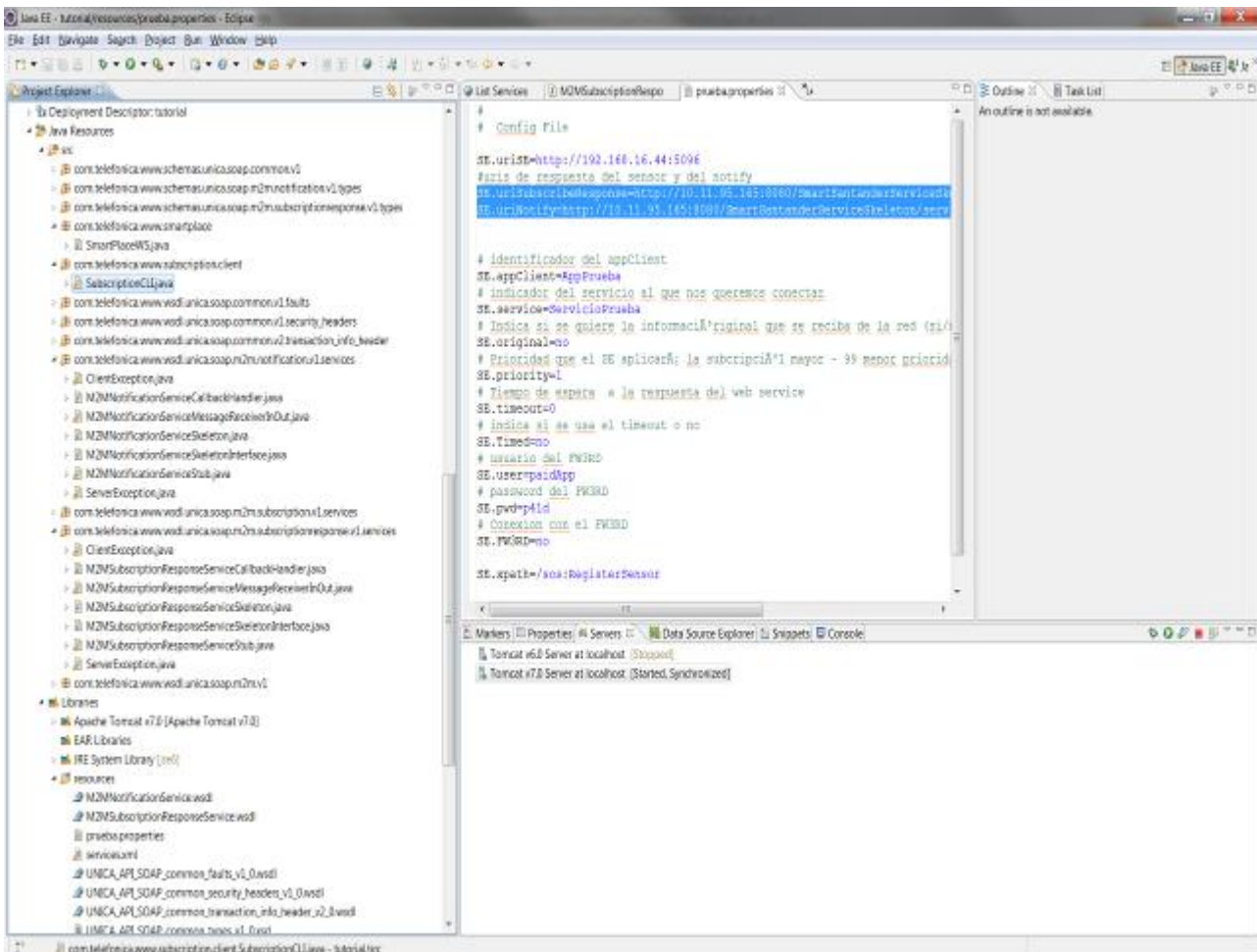
PUBLIC, SMARTSANTANDER PROJECT

M2MSubscriptionResponseServiceSkeleton (see the attached code). Include also the SubscriptionCLI.java and SmartPlaceWS.java sources in your project and finally include also the prueba.properties file under resources folder of your project and edit the properties of uriSubscribeResponse and uriNotify adding the endpoints of both of the services:

SE.uriSubscribeResponse= http://10.11.95.165:8080/SmartSantanderServiceSkeleton/services/M2MSubscriptionResponseService

SE.uriNotify= http://10.11.95.165:8080/SmartSantanderServiceSkeleton/services/M2MNotificationService.

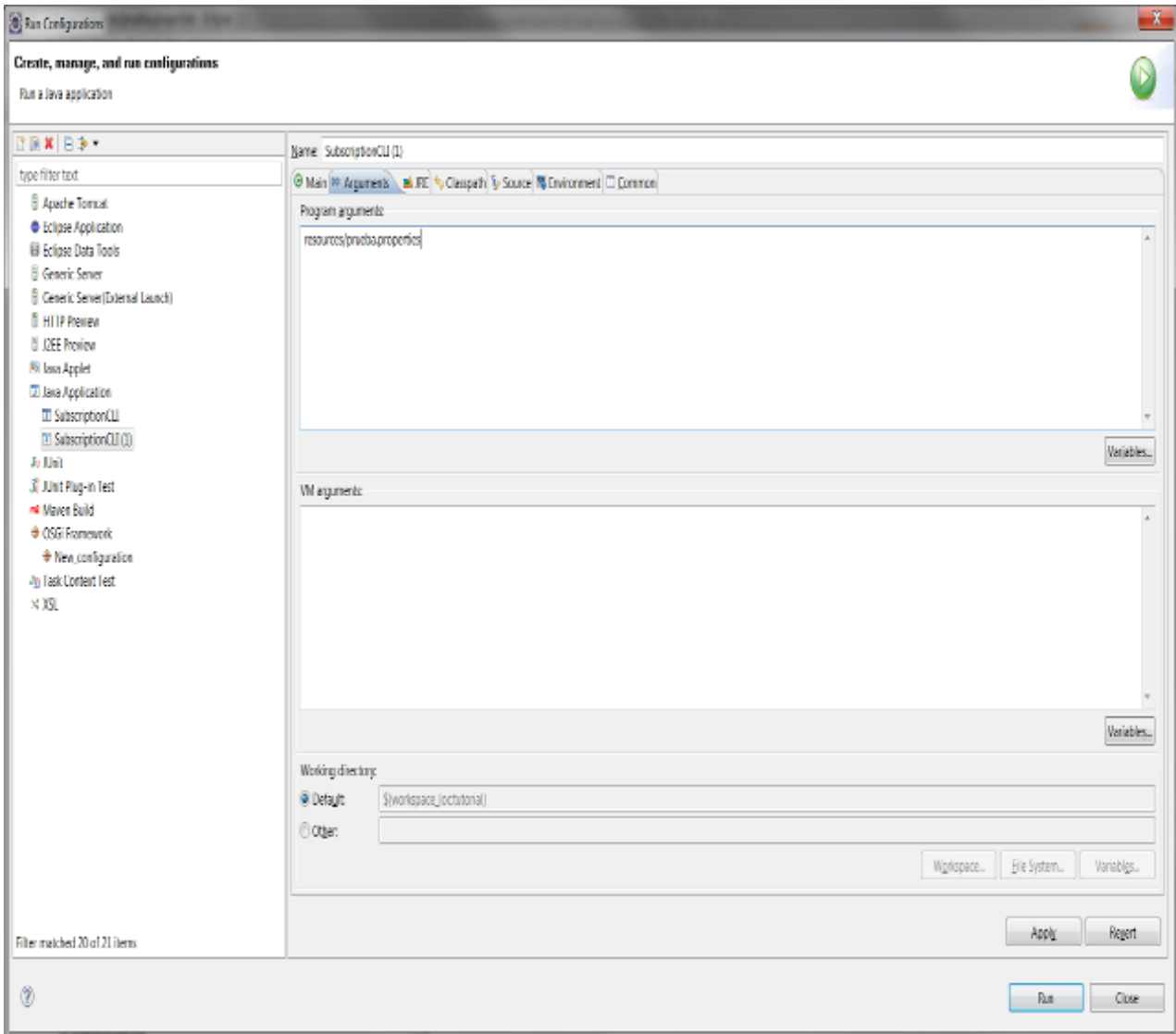
Now we just need to run the SubscriptionCLI.java. Start your Tomcat server with the notification and subscription services. Then go to the SubscriptionCLI.java and right click on it. Select Run As->Run Configurations.



On the program arguments add the path to the prueba.properties file and click on Run.



PUBLIC, SMARTSANTANDER PROJECT



You should get the following output:

SubscriptionCLI:http://192.168.16.44:5096

uriSubscribeResponse:http://10.11.95.165:8080/SmartSantanderServiceSkeleton/services/M2MSubscriptionR
esponseService subscribeObservation:AppPrueba service:ServicioPrueba

uriNotify:http://10.11.95.165:8080/SmartSantanderServiceSkeleton/services/M2MNotificationService

seconds:0 original:false priority:1 xpath:/sos:RegisterSensor [INFO] calling to subscribeRegister

AppPrueba:ServicioPrueba ...