COSMOS

Cultivate resilient smart Objects for Sustainable city applicatiOnS

Grant Agreement Nº 609043

D7.3.3 Smart events and protocols for smart public transport (Year 3)

WP7: Use cases Adaptation, Integration and Experimentation

Version: 1.0
Due Date: 31/08/2016
Delivery Date: 31/08/2016
Nature: Report
Dissemination Level: Public

Lead partner: EMT
Authors: Andrés Recio, Sergio Fernández

Internal reviewers: ALL

www.iot-cosmos.eu

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 609043

Version Control:

Version	Date	Author	Author's Organization	Changes
0.1	10/07/2016	Andrés Recio / Sergio Fernández	EMT	ToC
0.2	12/07/2016	Andrés Recio / Sergio Fernández	EMT	Adding new subchapters
0.3	16/07/2016	Andrés Recio / Sergio Fernández	EMT	Completing chapters
0.4	03/08/2016	Andrés Recio / Sergio Fernández	EMT	First full draft
0.5	10/08/2016	Andrés Recio / Sergio Fernández	EMT	New additions to the document
0.6	12/8/2016	Andrés Recio / Sergio Fernández	EMT	First EMT final draft review
0.7	19/08/2016	Juan Sancho	ATOS	Document Review
0.8	26/8/2016	Andrés Recio / Sergio Fernández	EMT	Comments addressed
0.9	28/08/2016	Juan Sancho	ATOS	Additional comments
1.0	1/09/2016	Andrés Recio / Sergio Fernández	EMT	Final version

Table of Contents

Table of Figures

List of tables

Acronyms

Acronym	Meaning
ASPX	Active Server Pages Extended File
BSON	Binary JSON
CEP	Complex Event Processing
CG	Care Giver
CPU	Central Processing Unit
CRTM	Consorcio Regional de Transportes de Madrid (Madrid Regional Transport Authority)
D	Deliverable
DoW	Description of Work
DDP	Data Distribution Protocol
DSO	Domain Specific Ontology
EMT	Madrid Public Transportation Company (Empresa Municipal de Transportes de Madrid)
EMTIng	EMT Gamification Platform
ETA	Estimated Time of Arrival
FC	Functional Component
GPS	Global Positioning System
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IIS	Internet Information Services
IoT	Internet of Things
IP	Integration Point
JSON	Java-Script Object Notation
LD	Linked Data
MB	Message Bus
MS	Milestone
OS	Operating System
PE	Physical Entity
RB	Reactive Box
RB SAE	Reactive Box which data source is the SAE
REST	Representational State Transfer
SAE	EMT operating aid system (Servicio de Ayuda a la Explotación)
SDK	Software Development Kit
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Person with special needs
SSH	Secure Shell
SSID	Service Set Identifier
SVN	Subversion
TCP	Transmission Control Protocol
UC	Use Case

UI	User Interface
URL	Uniform Resource Locator
UTM	Universal Transverse Mercator
VE	Virtual Entity
VEProt	Virtual Entity Process for Reactive and Ontological Things
VM	Virtual Machine
WP	Work Package
WS	WEB SERVICE or WEB SOCKET (as apply in the text)
XML	Extensible Mark-up Language

1. Introduction

1.1. Overview and scope

One of the key aspects of a system is to demonstrate its effectiveness and functionality under specific items that reveal what's really behind it and understand its operation. In high abstraction systems based on IoT this is fundamental because different components and systems require a concrete basis on which to stand, being the use cases the essential mechanism for this.

In order to provide to Madrid Mobility scenario these particular elements where parties and required functions will be based for the COSMOS model of integration and functionality, two different use cases are considered:

- Route Monitoring Use Case
- Traffic Analysis Use Case

Those use cases will serve as a demonstration for the overall Madrid Mobility scenario, involving the whole set of different elements of the COSMOS ecosystem.

As indicated in D7.1.1, the first use case of Madrid Mobility scenario was initially oriented to citizens which require protection and assistance (what has been called in this project “Person with special needs”; that is, people with reduced mobility or special mobility needs such as elderly, handicapped people or children). The idea is to provide a service of routing and indications for this people to help them using the bus, including the possibility of making a check-in once on board the bus and monitoring by a caregiver or responsible for ensuring the trip or the collection on arrival. A simple example could be: “A kid takes a bus to go to school, and his VE (device) connects with the VE bus to notify when he gets on and where and when he gets off the bus, and notifying that to his parents if he has reached his final destination”.

However, the aforementioned use case involves not only an important challenge to the different actors and technicians responsible for the COSMOS consortium but also to the transport and traffic infrastructure of Madrid, as for supporting COSMOS elements, it is required to create a support infrastructure based on real-time events, which is non-existent so far, and whose motivation for development within the area of mobility has been raised in the context of the current project. Therefore, the second use case included within the Madrid Mobility scenario is the Traffic Analysis.

Finally, although some key parts require the degree of refinement necessary to consider themselves as fully operational, the design and prototyping is serving for the entire set to be tested at all levels.

This final document includes a compendium of the work developed within the whole duration of the project, including the complete specification result concluding with a complete specification which relates how Events and Protocols have been implemented within the field of intelligent mobility.

1.2. Motivation

As it has been specified within the previous deliverables, the Madrid mobility oriented virtual entities architecture has been developed under a specific multi-purpose architecture called VEProt. This system contains various components which have required the implementation of

specific developments with different software architectures and infrastructures that will be put into operation for the prototype. Moreover, data exchange standards and specific communication models have been defined in order to build a stable platform that supports the system even beyond the proposed use cases. Therefore, the prototype developed for the Madrid scenario has three clear purposes:

1. Check the right functioning of the integration model and the VEProt inter-operability platform which will be explained herein.
2. Create an optimal environment to verify the correct functioning of the COSMOS infrastructure.
3. Offering an opendata model oriented to events within the mobility field that allows developers and integratos to create new Virtual Entities and exchange information in a collaborative growing system.

2. Definition of prototype components

2.1. Technical basis for the prototype

2.1.1. Setting a laboratory for mobility

Due to the plurality of objectives and the complexity of the infrastructure that was needed by the use cases, in November 2014 a laboratory for mobility was implemented in Madrid, from which to gather ideas and initiatives to help building the final platform.

This lab, supported not only by COSMOS project partners but for various companies, universities and the city of Madrid itself, is coordinating tasks mainly oriented to the definition of standards and semantic elements for urban mobility, especially in the transport and traffic management sector.

One of the outputs of this lab is the definition of the data model that supports Madrid UC, and which specification is published in this link:

https://github.com/madridopenlabmobility/MOBILITY-MADRID-virtual-entities

Within this laboratory, in collaboration with various companies, the RB system model has been evolved and some software has been built for the Madrid scenario, especially in the event-driven management through DDP or optimizing indexes and NO-SQL databases, which are essential for both the local cloud and for the high availability system that require the watching events elements through Meteor.[i]

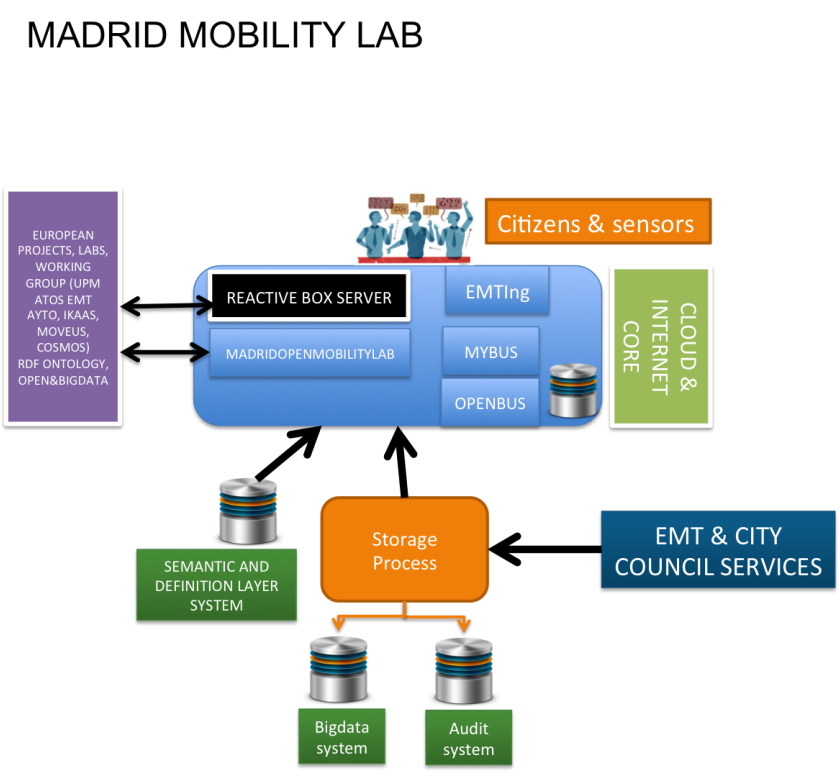


Figure 1. Madrid Mobility Lab infrastructure scheme

2.1.2. Unification of message formats in different areas of the mobility related virtual entities

The creation of a dynamic, flexible and adaptive model of virtual entities is complex; even more if based on schemes whose pattern does not follow criteria that can be evaluated by a system. As it was recorded in the initial requirements for the design of virtual entities within the COSMOS project plan, certain essential attributes for autonomy and interaction of objects within the paradigm of IoT are needed. The specification VEProt meet that goal, allowing simultaneously:[ii]

- Communicate two virtual entities together expressing its attributes and its context.
- Communicate an entity with a platform and vice versa. Transferring files and data from a virtual entity to derived systems as storage models, actuators or analytical elements.
- Indicate operations or actions to build collaborative models of entities that allow functionality scaling in SmartCity environments.
- Building an scalable platform in which, in a simple way, any developer can get information in public mode or build new heterogeneous Virtual Entities' models based in the same rules and structures.

2.1.2.1 VEProt datagram

The VEProt scheme has five blocks of data for storage and exchange of data between VEs:

1. vep:header. Defines the general characteristics of the virtual entity, among others, their geographical location, who is the owner of the showed data, information about the appearance and disappearance of data and its iteration within the data managed by the VE.


```
{     "vep:versionProtocol" : "1.00",
"vep:geometry" : {
"type" :    "Point",
"coordinates" : ["-3.69138338267", "40.4211505276"]
},
"vep:iteration" : {
"autoUpdateExpiration" : "2"
},
"_id" : "a8f4d6d7-3b81-11e5-9a82-406c8f10d363",
"vep:owner" : "SERVERSENSORMANAGER.WASP.1999",
"vep:utcExpiration" : "2015-10-04 14:52:57.619549",
"vep:statusLocked" : 0,
"vep:utcGeneration" : "2015-08-05 14:52:57.619540",
"vep:utcAsignation" : "2015-08-05 14:52:57.619561"
```


2. vep:notification: It contains information about where to receive notifications when seeking communication with the VE.


```
{     "vep:port" : "16003",
   "vep:address" : "localhost",
 "vep:modelistening" : "SKT" }
```


3. vep:source: Specifies which modules, functions and features made the operation of the VE at every moment or particular message within the scope of each VE. It also defines the semantic context of the VE.

```
{     "vep:id" : "DEVICEEVENTMAD.WASP.1999",
    "vep:version" : "1.00",
 "vep:function" : "MANAGEMESSAGES",
  "vep:subsystem" : "WASPSERVERMODULE",
   "vep:system" : "LOCALSENSORMANAGER",
    "vep:context" :
{ "layer" : "VEPROTMADRID",
        "vep" : "http://mobilitylabmadrid.emtmadrid.es/vep"     } }
```

4. vep:target: Describe, if there were, which is the recipient or consumer of information that revealed by the the VE when a message or data is requested to this VE.

```
{     "vep:system" : "SERVEREVENTSMANAGER",
    "vep:subsystem" : "LOADDATALAYERS" }
```

5. vep:body: Defines the values to exchange or the information that, at every moment, manifests a particular VE and is capable of being read by any element that communicates with it, either directly through the VEProt communication elements or through the RB. It is subdivided into four sections:

    1. vep:plan: Specifies a plan or execution flow when a VE is prompted to processing a task if there is capacity to do so
    2. vep:requisites: Indicates which contextual elements are involved in the execution of a sub-process within a VE.
    3. vep:type: Shows the type of values contained by the data area of the datagram, in relation to the set of types that are specified in the documentation of the VEProt datagram.
    4. vep:data: Contains the data or parameters of a specific datagram message containing an observation or execution unit of a VE.

```
{     "vep:plan" : {
   "typePlan" : "noPlan"     },
   "vep:data" :
{ "busData" :
{"status" : "5",
  "direction" : "1",
  "bus" : "8811",
  "stop" : "1518",
  "line" : "150",
  "trip" : "14",
  "name" : "####",
 "geometry" :
 {"type":"Point","coordinates":["-3.69138338267",
"40.4211505276"]},
"altitude" : "626.372",
"delay" : "-346",
"offSet" : "570" }}],
"vep:requisites" : {
"requisites" : "noRequisites"},
"vep:type" :
{"typeContent" : "busPosition"     } }
```

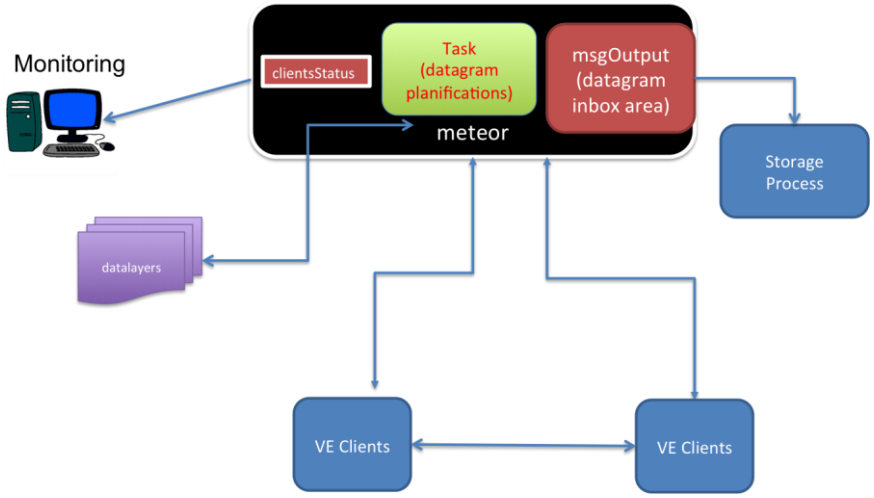


Figure 2. Exchange of VEProt datagrams high level architecture

2.1.2.1.1 *Managing security in the context of COSMOS SmartMobility (Year 3)*

Within the framework of the implementation in public and open mode of the COSMOS infrastructure which is visible through MobilityLabs, it has been necessary to develop a security model within the RB to serve as a logical firewall between the plug-in systems through the RB and the different available Virtual Entities:

- Creating a connection system that ensures blocking against accesses that violate safety or against inadequate or abusive use of the systems housing infrastructure. To do this, connections have been not only protected by a secure tunnel (SSL), but a logon system has been defined and implemented.
- Defining the access level to the Virtual Entities. There are VE that may not be visible in public and open mode, or whose restriction means that only certain sub-elements of the information contained in the VE are accessible depending on the profile or role of the connected system. This involves the administration of roles and permissions to know, for each connected system, what information of which VE is accesible.
- Defining the property level of Virtual Entities. This level allows any connected system to know who has defined and who feeds the data that a VE represents a VE and defines the responsibility for them.
- Defining the change level in the Virtual Entities. This way, reading accesses can be separated from writing accesses, insulating layers or functions that allow changes to the information contained in the VE, even though its information is of public character, letting the ability to write only to the owner or to whom has been granted permission to write.

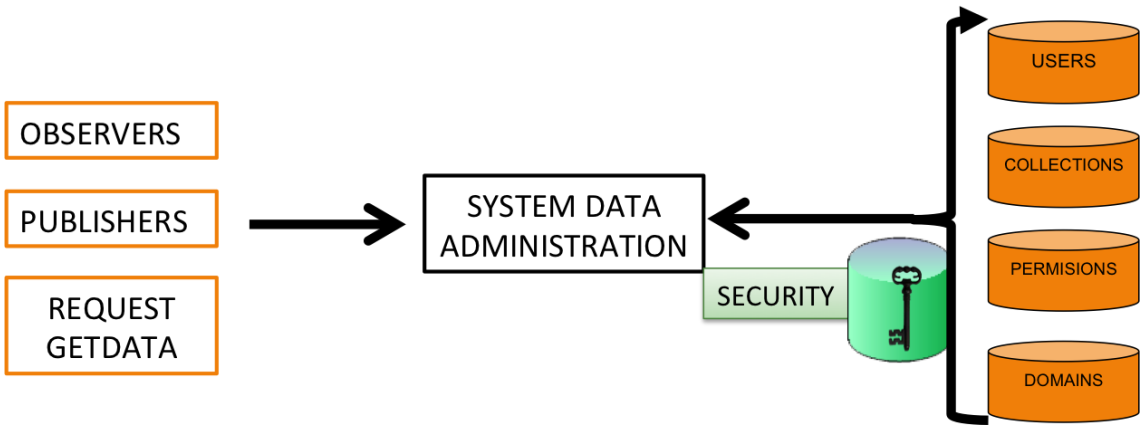


Figure 3. Security model scheme

The security model for MobilityLabs-RB has been called SDA (System Data Administration) and it has been designed as a model separated from any infrastructure, so it can be integrated into different subsystems. In summary, it is independent and functions as an isolated system. Its high level data model is the following:

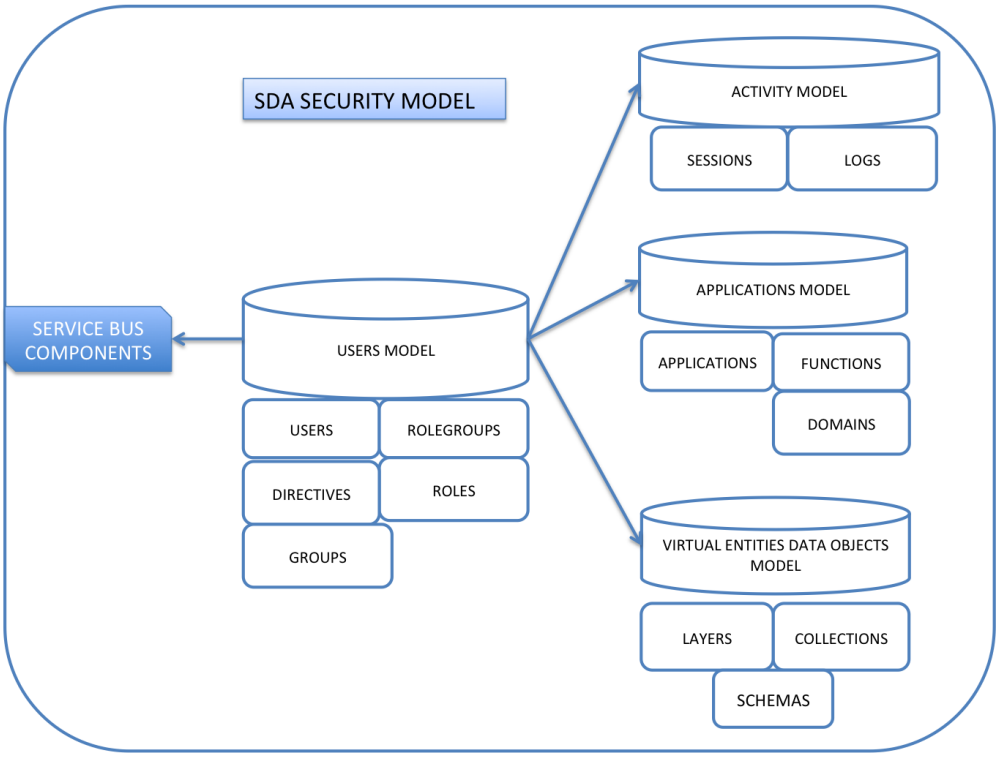


Figure 4. High level detailed security model scheme

For its management, in addition, there is a Web Services infrastructure whose level of accessibility depends on the public or private level of access to the information model. There is an API REST that connects directly and internally with the data access services which is managed by creating a temporary cache, returning a ticket granting access which expires at the conclusion of the time set, requiring starting a new session. This allows keeping connection activities of different customers in asynchronous mode for a limited time period if it is not used or extended by the different uses to its full limit activity defined at the application level.

Sessions Collection		
fieldName	type	description
_id	Key, object	Unique id of this document (id of session)
dateUpdated	Datetime, mandatory, index autodelete	Datetime of last updated
dateCreated	Datetime, mandatory	Datetime of creation data session
dateEnd	Datetime, mandatory, index, autodelete	Datetime of end of data session.
idApplication	String	Id of application using this session
IP	String	Ip direction of user app
idUser	String	Id of current user
dataCache	Object	Object data with parts of related collections

Table 1. Sessions VE Data Model

Another feature of the Security Model is that it identifies the different systems or connected Customers who have access to it. This is especially important to know features like accessibility or health risks. As it is defined in the following table, the ratio of elements of the entity "User" contains attributes that allow health preferences (healthPreferences) and special needs it may have (needAids).

Users Collection		
fieldName	type	description
_id	Key, String	Unique value of item (is the user id or login code)
firstName	String, mandatory	First user name
lastName	String, mandatory	Last user name
dateIni	Datetime, mandatory	Date of Creation in security system
dateEnd	Datetime, mandatory	Date end in security system (31/12/9999 is equal to infinite value)
dateLastUpdate	Datetime, mandatory	Date Last Updated of any data
dateLastLogin	Datetime, mandatory	Date last login
password	String, mandatory	Password encrypted value
timeRestrictions	Object	
daysAllow	Int	1 byte for representing days of week which the user can Access to system. The value representation is: 0111 1111. Bit less significative is Sunday and seventh bite is Saturday. 1 = allow Access 0.- deny Access if last bit is locked (= 1) everyday will be locked (full

		locked)
timeAccess	Object Array	Object to represent slot of times for Access or denies. Contains this format: { "timeAccess":[{“allow”:1:"from":"secs","to":"secs"},{“allow”:0,"from":"secs","to":"secs"}] }]}
phone	String, optional	Phone number
Email	String, mandatory	Mail
dateExpirePassw	Datetime, mandatory	Date of expiration password
idDirective	String, mandatory	Directive of user
idGroups	Object, optional	Array of ids of User groups.
Picture	String, optional	url of photograph of user
needAids	Int, optional	Defines special needs using bits of one byte 1.- Visual disability 1.- Hearing disability 1.- Cognitive impairment 1.- infant 1.- ederly 1.- motion disability 1.- Wheel chair 1.- RFU
healthPreferences	Int, optional	Defines healthy preferences usings bits of one byte 1.- Pollen Allergy 1.- Respiratory disese 1.- Heart disese 1.- Skin disese 1.- RFU 1.-RFU 1.-RFU 1.-RFU
idDomains	Object, mandatory	Array of Domain which this users belongs to

Table 2. Users VE Data Model

In short, the security model is not only a function to connect systems, but for letting users from connected systems to log on and let applications to know their special needs.

In addition, the system should also allow roles relationship between users, which is essential for the management of Assisted routes, so that the Caregiver can establish a relationship with the supervised users and therefore this can be known by the system. To do this, one of the entities defined in the SDA develops this relationship:

RoleGroups Collection		
fieldName	type	description
_id	Key, identity	Unique value of item
idUser	String, mandatory	User identification, registered in Users collection
dateIni	Datetime, mandatory	Date from in this relationship

dateEnd	Datetime, mandatory	Date to in this relationship, (31/12/9999 = infinite)
idRole	String, mandatory	Id of Role
idUsers	Object, mandatory	Array of users belongs to idUsers

Table 3. Relationships between Users VE Data Model

The whole scheme of SDA Security Model using MobilityLabs for the integration of Smarts VE in the Madrid scenario is the following:

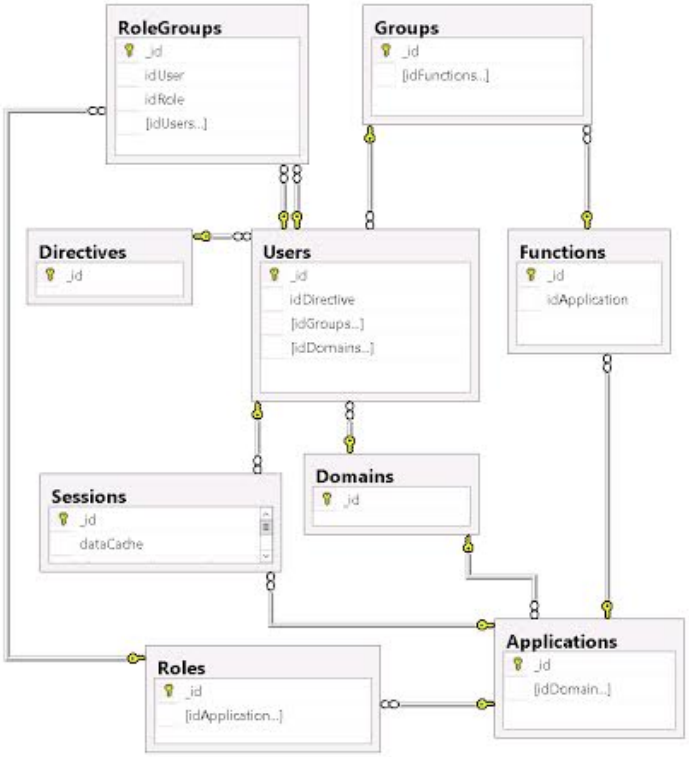


Figure 5. Complete SDA Security model scheme

In addition, the set of implemented API REST FULL services allows access control to the system from the part of system architecture, once the connected system gets its session ticket. At a high level, this is the services scheme:

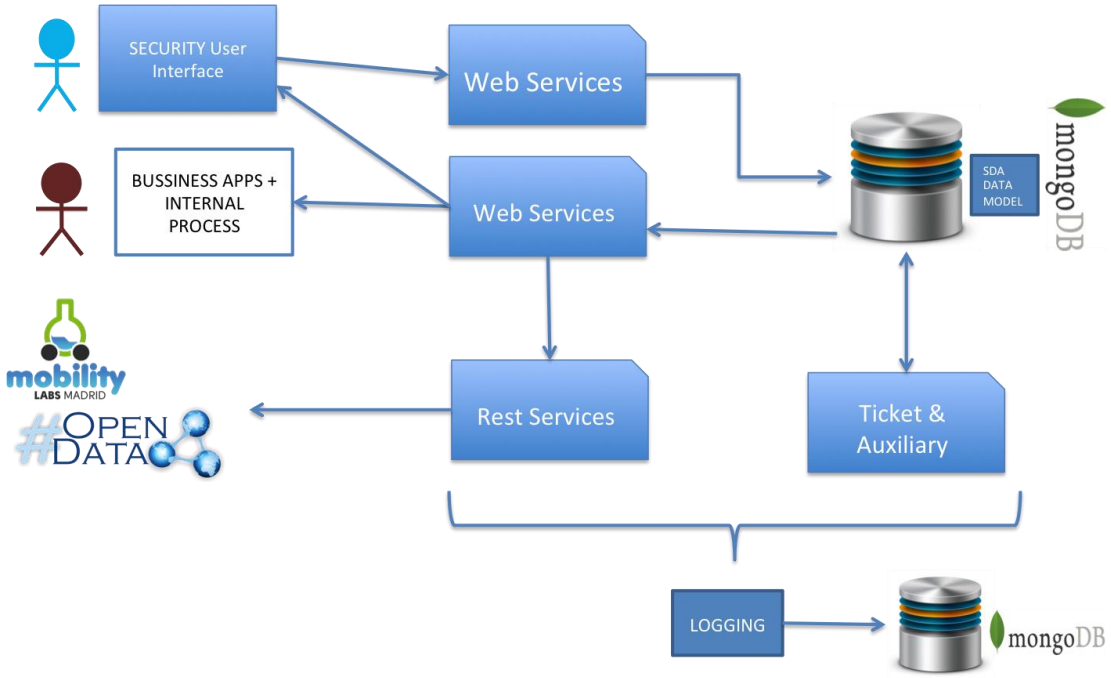


Figure 6. API REST FULL services scheme

Finally, indicate that any action of access or modification within the scope of system security is stored to be audited later. For the registration and traceability of all accesses (LOGGING) the Rabbit server itself is used, conferring the queue management storing usage and activity in new Audit Virtual Entities which can obtain subsequently, KPI of usage. These KPIs provide information for periods (hour, day, month, year):

- Number of starts of session (correct / incorrect)
- Accesses by application.
- Functions used
- Data read and written by Virtual Entity.
- Other KPI of interest.

2.1.2.1.2 New features and functionality of VEProt datagram in Year 3

During the deployment phase under the Opendata mode, the new requirements involved in managing and maintaining the lifecycle of the information associated with the Virtual Entities have been analyzed. Specifically, the problem arises from the relationship of an external system with COSMOS environment using the Reactive Box architecture as the SmartMobility gateway. For this reason it is necessary, once established the security elements defined in the previous chapter, to define the various functions that allow not only the reading of historical data or the observation of real time phenomena, but also the load, change and deletion of data within the logical structures that support the data defined in the virtual entities.

The four FUNCTIONS defined in the datagram in order to create or change any information hosted by Virtual Entities are defined in the following figure. This allows to connected systems that have specific change privileges, to make changes to the VE information. The system, through the security model, controls that no data of any VE is modified by a user / system that does not have ownership of it. In short, the information can only be modified if:

- Any information can only be inserted into the logic data structure of a VE if the system (User) has writing privileges on the information of that VE.
- Any information can only be modified or deleted in the logic data structure of a VE if the system (User) is the owner of that specific data.

A logical access and action scheme of each of the each functions implemented during Y3 is:

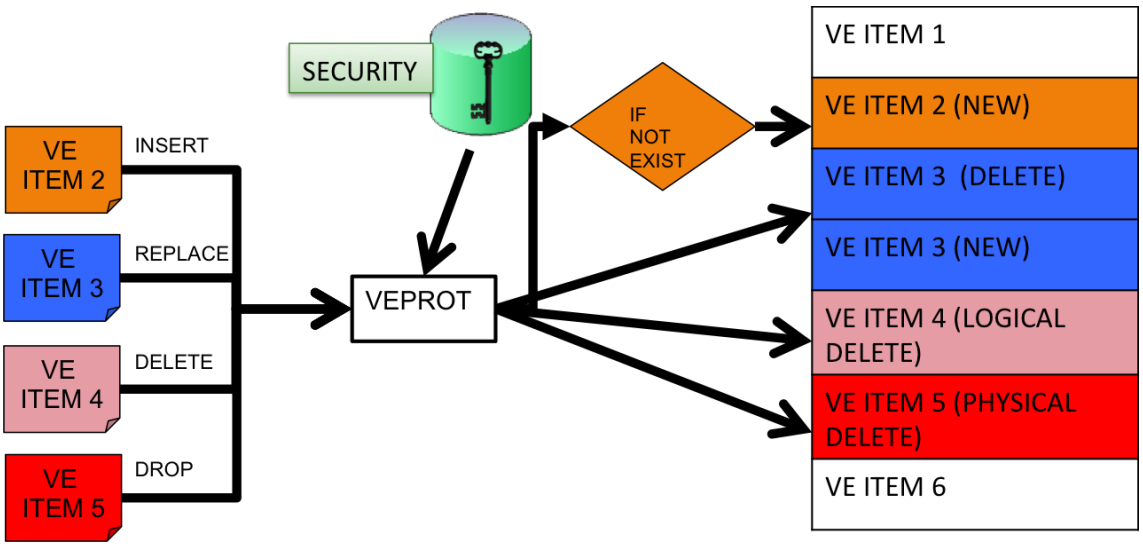


Figure 7. New functions defined within the datagram in Y3

The changes required in the VEProt datagram to perform this functionality are defined in the next chapter.

2.1.2.1.3 Changes in datagram VEPROT in Year 3

One of the biggest challenges has been to simplify the complex datagram that comprises the complete communication infrastructure in order to simplify it so it could be used by any connectable system. To do so, a major effort has been made to eliminate from the RB Opendata service model any difficulty for the integrators, while not loosing any element of compatibility with the Protocol requirements at a security level and flow of information .

Currently, a simple datagram model has been achieved to be used openly over the Internet, which enables the entire lifecycle of information within the SmartMobility, consisting of:

- Register new VEs or new versions of them.
- Retrieve and know what information structure have the elements obtained by direct observation or historical mode.
- Deliver / modify data for the logical storage of information contained in VEs.
- To know who, when and how a data has been entered or modified in the system.

The simplification of the model can be easily checked by comparing the internal models (flow MOBILITYLABS/COSMOS) with the public model:

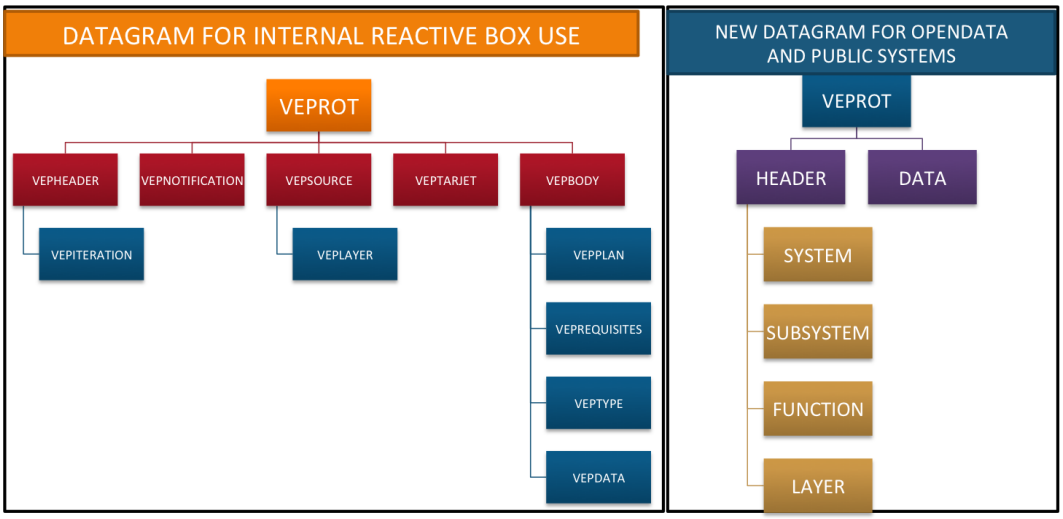


Figure 8. Flow comparison between Internal and Public model

To this end, it has been necessary to provide certain subelements to the new VEProt Datagram structure, while others have been eliminated.

The Opendata simplified format of the VEProt datagram

Although there are multiple combinations within the VEProt datagram, it is necessary to comply specific formatting rules to allow the validation of a message by the system and, eventually, to become part of the MobilityLabs collections from Opendata. Once the data is validated and accepted, it is available for use through the observation mechanisms such as the Reactive Box or the Historical Collections Web Services.

The minimum values required for the insertion, modification and deletion of VEs are:

- General Header: It expresses the intention to introduce a datagram in the message to perform an action using the datagram's server. Its definition is "target": "datagramServer"
- Data Section: It contains arrays with the definition of each data section (document) for the datagram's server. It is defined as: "vep_data": []
- Layer Section: It contains the definition of the collection (array item of the Data Section) containing information of the collection (layer) with which you want to work. Its definition is "layerData": {}

It contains the following relevant elements:

- _id (Required). Element unique value: The value identified as a key attribute. Each element with this identifier is treated as an individual unit within the real time collection and a key subattribute within the historical collection. Each time a document in the collection is changed, the system automatically stores the previous version in the history, so that all elements are accessible.
- System (Required). Fixed value "LAYERS".
- Subsystem (Required). Fixed value PUTDATA.
- Function (Required). It can contain the following value types:
  - INSERT: It builds data collection should not exist and then insert the document into the collection with the key (_id). If the document defined with that _id already exists, it will ignore insertion. Once inserted, it builds a parallel Item with the key

ObjectId within the Historic collection; the _id value will be stored it in the "keyid" element.
    - REPLACE. It builds the data collection should not exist. If the document defined with that _id does not exist then it will create it; in the event that it already exists it will substitute it with the new Item defined by that _id. It builds a parallel Item with key ObjectId in the Historic collection; the _id value will be stored in the "keyid" element.
    - DELETE. It deletes the current value from the collection defined by that _id (should it exist). If the Elimina el valor actual de la colección definido con ese _id en caso de existir. If the operation is successful, it stores the item with the erase operation within the Historic collection.
    - DROP: Removes the defined value with that _id if it exists in the collection; in addition to that, it also removes all items with keyid equal to that _id from the Historic collection.
- Layer (Obligatorio): Estructura que define la colección con la que se desea trabajar. El usuario que accede deberá tener permisos para realizar las acciones descritas anteriormente sobre la colección. Contiene los siguientes ítems.
    - Owner: It defines the user that makes the operation and, as such, registers itself as the documetn builder
    - Type: It defines the way of sharing a collection; it can contain the following types:
        - Public: It defines that the ítem of that collection is open and shared for any user that has access to the collection. The other users can read the item but not modify it.
        - Shared: It defines that the ítem of that collection is open and shared for any user that has access to the collection and other users with writing privileges can take over property of the ítem; that is, update or delete it.
        - Private: it defines that the item of that collection is restricted to the solely use of the user that has built the data.
        - Name: Name of the collection on which you want to act. The user must have privileges over this collection. The collections are composed of two distinct parts. The first one is represented in capital letters and it refers to a group or family of the collection. It is recommended not to use names longer than 12 characters. The second one is represented in lowercase letters and contains one or two words that reference the data type of the collection; for each word a maximum of 8 characters is recommended. In case there are two words they must be separated by a point.
- Geometry (Opcional): It serves to define the data in a geographical location. The Geometry property allows the representation system to automatically project the data on a map using the website http://rbmobility.emtmadrid.es:3333 . All formats used in this structure must be in GEOJSON format (http://geojson.org/)
- Shape (Requested if a Geometry object is included). It contains the aspect definition It contains the definition of the aspect with which the item will be shown on the map. You can set marker type attributes (see http://fontawesome.io/).
- State (Requested if a Geometry object is included). It contains the information that appears in the vignette pressing or mousing over the map item. HTML can be used.
- Instant (Requested): Value of the data instant. String representing timing in format YYYY-MM-DD HH:mm:ss:nnnnnn.

In addition to these data, any elements or attributes desired in the document can be sent to complete the information that the user wishes to include within the collection.

Datagram example.

This datagram example shows the creating or modifying action of a document in the VE TRAFFICMAD.alarms, fed by the COSMOS system. This collection contains a forecast of traffic flow at a point of the city defined by its coordinates. As it can be seen, in addition to the mandatory values, it contains the structures that the COSMOS system defines for its convenience and use (userid: COSMOS.SERVICIOS.TRAFFICMAD), such as the traffic intensity, speed, alarm codes to be used in an observation model through Reactive Box, etc.

An example of this VE data is as follows:


```
{
    "_id": "PM20952",
    "subsystem": "PUTDATA",
    "function": "REPLACE",
    "layer": {
        "owner": "EMT.SERVICIOS.TRAFFICMAD",
        "type": "public",
        "name": "TRAFFICMAD.alarms"
    },
    "levelAlarm": "I",
    "instant": "2016-08-18 16:40:49.867",
    "rating": "-1",
    "geometry": {
        "type": "Point",
        "coordinates": [-3.666663, 40.401878]
    },
    "textStatus": "GoodTraffic",
    "trafficState": {
        "codeStation": "PM20952",
        "speed": "75",
        "intensity": "2400"
    },
    "system": "LAYERS",
    "serviceLevel": "0",
    "conditionAlarm": {
        "factorSpeed": "0.500000",
        "factorIntensity": "0.500000",
        "meanSpeed": "30",
        "meanIntensity": "1000"
    },
    "codeAlarm": "40",
    "shape": {
        "type": "marker",
        "options": {
            "shape": "circle",
            "markerColor": "green",
            "prefix": "fa",
            "icon": "fa-exclamation-triangle"
        }
    },
    "state": {
        "color": "white",
        "instant": "2016-08-18 16:40:49.867",
        "description": "<b><p>Station:
PM20952</p><p>State:GoodTraffic</p><p>Time:2016-08-18
16:40:49.867</p></b>",
       "value": "1",
       "format": "text"
    },
    "alertReceived": "0"
}
```

2.1.2.2 Defining an interoperable data model for intelligent transport environments.

The specification of the data model, as indicated in the previous point, was carried out with the clear intention to equip the whole system with a semantic scheme based on JSON-LD. Both the scheme and the information is stored in the same NoSQL database. Thus, it is possible to extract information as well as data in its native format. [3]

The chosen database engine is MongoDB.[4]

The documents are symbolically defined as layers and contain a logic structured as follows:

- A document defining layers (_layers) in which the characteristics of each family of data are specified.
- A semantic specification document called [layername].ontology in which objects and their semantic annotation are described.
- A document called [layername].things in which the general description of each virtual entity is contained.
- One or more derived documents in which the transactional elements or events related to each layer collection are stored.

The defined layers for the prototype are:

- LINESMAD: Madrid bus lines.
  - LINESMAD.things. Contains the structural elements of those bus lines
  - LINESMAD.ontology. Contains the ontological scheme of the bus lines structure.
- BUSMADRID: Contents related to the Madrid urban buses.
  - BUSMADRID.thing. Definition of the general characteristics of buses.
  - BUSMADRID.ontology. Ontological scheme of buses.
  - BUSMADRID.dataevent. Status of each bus in real time with its geographical position.
- STOPMAD: Madrid bus stops.
  - STOPMAD.things. Definition of the bus stop object.
  - STOPMAD. Ontology. Ontological scheme of bus stops.
  - STOPMAD.events. Status of each bus stop and bus arrival time of each bus line stoping by each bus stop.
- APPLICATIONMAD: List of applications connected to the system.
  - APPLICATION.things. Specification of each app object.
  - APPLICATION.ontology. Ontological scheme of app.
  - APPLICATION.events. Status of each app and its uses.
- USERMAD: List of users acceding to the system and their roles or profiles.
  - USERMAD.things. Specification of the user entity.
  - USERMAD.ontology. Ontological scheme of the user entity.
  - USERMAD.events. Activity of users in relation with the application use.
- DRIVERMAD: Specifications related to bus drivers.
  - DRIVERMAD.things. List of bus drivers that can provide transport services.
  - DRIVERMAD.ontology. Ontological scheme of the bus driver entity.
  - DRIVERMAD.events. Activity of the bus driver during the service provided at a certain bus line.
  - DRIVERMAD.messages. Messages sent and received by bus drivers at the onboard console.

- ROUTEMAD: Design of the person with special needs usual routes.
  - ROUTEMAD.things. This has no use.
  - ROUTEMAD.ontology. Ontological scheme of the elements involved in planning routes.
  - ROUTEMAD.userplan. Contains the designs and planning schemes fo the person with special needs routes.
  - ROUTEMAD.bustracking. Records every position of the vehicles belonging to each route with the periods each route planning belongs.
  - ROUTEMAD.usertracking. Records the position of each person with special needs which is being monitored.
- MSGOUT: Defines the RB message queue.
  - MSGOUT.things This has no use.
  - MSGOUT.ontology. Defines the Ontological scheme of the message exchange VEProt datagram.
  - MSGOUT.messages. Contains the message pool and the activity of the system.

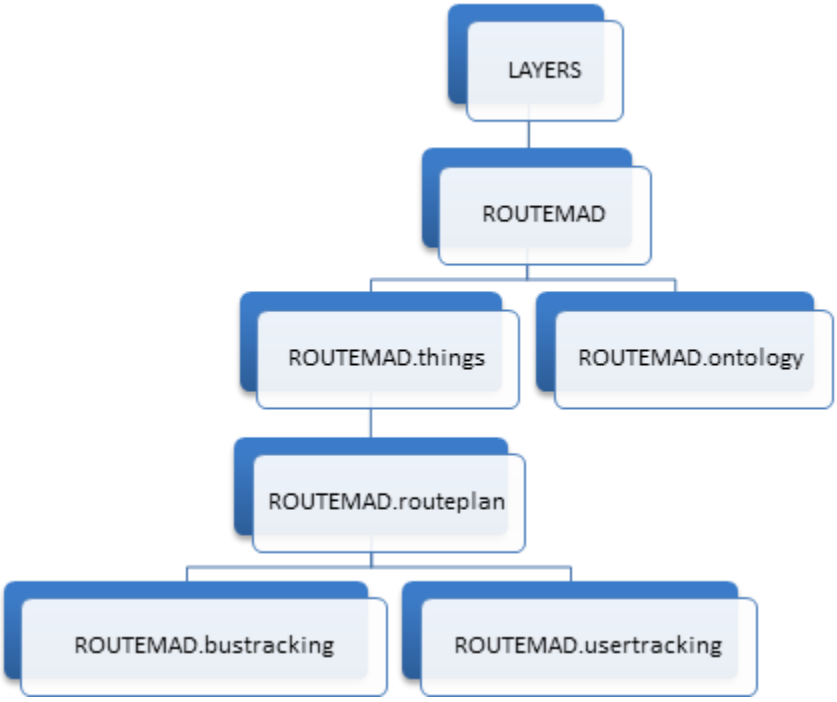


Figure 9. Example of hierarchy of the ROUTEMAD layer

Moreover, there is a special collection named “users” that specifies the users and access privileges to the RB. For each user connected to the RB it is indicated wich layers have access to and therefore is granted access privileges (superuser) either to read or write. Finally, the hierarchy of the own collections included into the database that contains the layers, allow to establish sets of elements in "private" or "public" mode so that each item of each layer can hide or show part of its content independently and in function of the security role of the user connected to the RB.

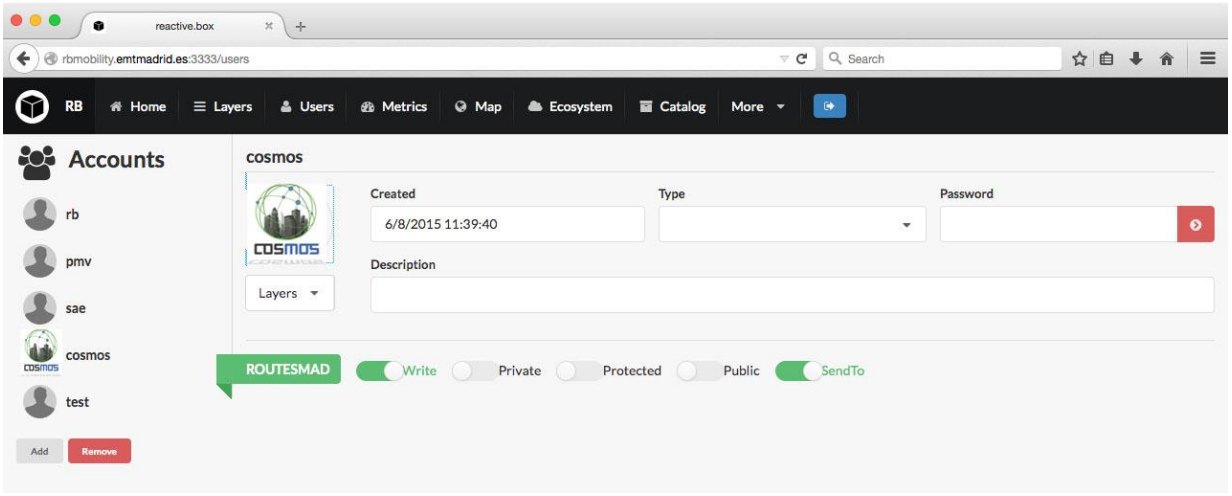


Figure 10. User maintenance schedule in the Reactive Box management portal

2.1.2.3 New public Virtual Entities for SmartMobility in Y3.

Another area in which an special effort has been made during Y3 has been to integrate different data sources that can be useful from the point of view of systems to be connected via COSMOS or even in native mode using MobilityLabs. To this end, data load engines have been developed connected to various public data sources, performing a standardization of data in the Reactive Box.

Thus, the number of public VE has greatly increased, and there is currently a wealth of information that allows building observation models with complex opticals. In addition to that, the sources of information of the systems that are starting to get connected are also in process of being incorporated, especially after the hackathon held during July and August 2016. This is enabling not only the dissemination of knowledge on using COSMOS-CEP, but how to feed the infrastructure itself with new data.

The new VE families with updated information that have been incorporated during Y3 can be seen in the following figure.

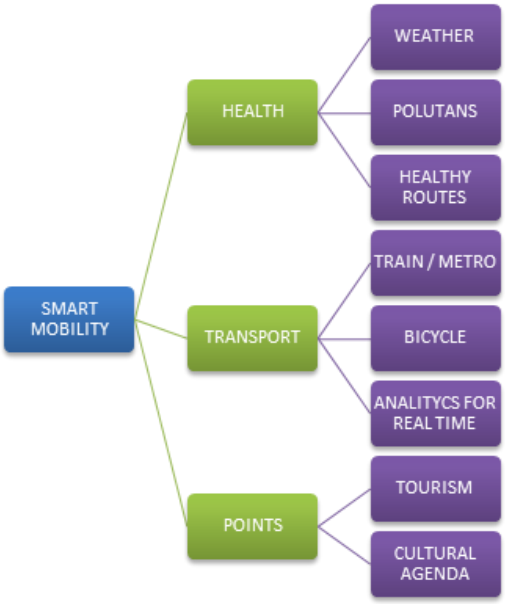


Figure 11. New VE families incorporated during Y3

2.1.3. Building interoperable and scalable platforms for managing events

The Reactive Box platform aims to exchange messages and status across the system. Its architecture -polymorphic and scalable- is adaptive to any scheme or definition of message exchange. It is based on Meteor under Node.js and is supported with a MongoDB database. Several DDP connectors have been developed to observe status changes of the system in a way that each RB reports in real time the status changes of the whole system. Thanks to this technology it is possible to deploy the user monitoring by their caregivers, as it is raised within the COSMOS UC.

Among the different RB being deployed in the field of urban intelligent transportation, we will explain the two ones involved in Madrid UC:

- RB SAE: The RB SAE is responsible for registering and exchanging status of buses, bus stops and bus lines during operation. To update the RB SAE, the system performs observations of the fleet management system and records the various changes of status that occur.[5]

In order to feed the RB SAE there is a real time connector which is watching changes in the SQL Server of the system. Each change detected is automatically transferred to the MongoDB of the RB SAE, in such a way that it will be always updated with the changes that occur in the EMT fleet control system. For the Madrid UC, the important collections available in the RB SAE, are those related to the position and activity of the buses and their relationship with the bus lines and bus stops of the specific route the bus is doing.

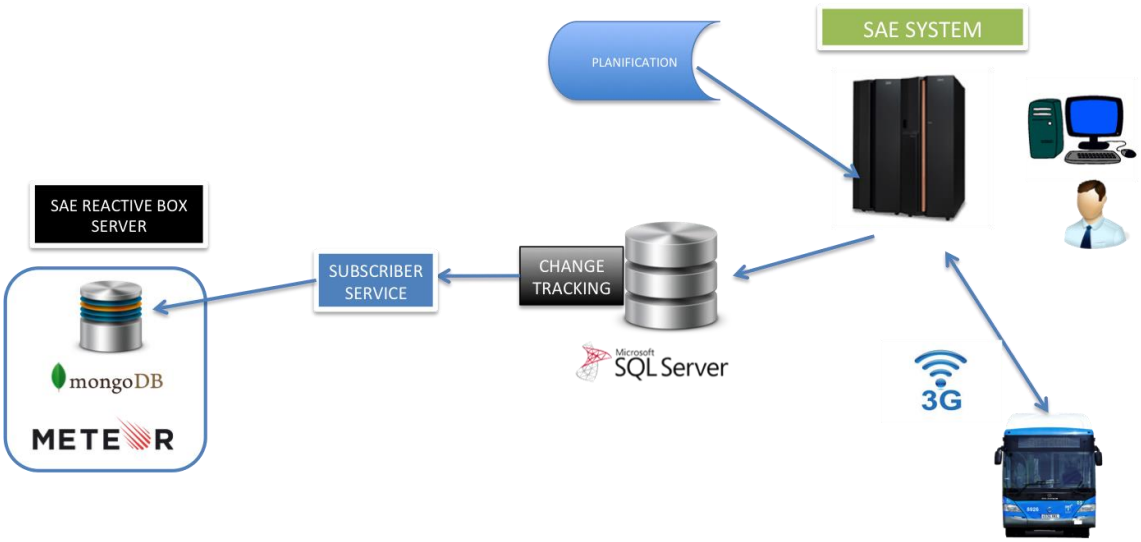


Figure 12. Data charge scheme from the EMT Fleet control system towards the RB SAE

- This RB aims to provide real-time events related to virtual entities in the area of IoT. The design, conceived and inspired to serve as an integrator element between mobility events and COSMOS architecture, provides, through its abstract layers, information on the activity of traffic and transport, allowing observing any changes through the site http://rbmobility.emtmadrid.es:3333.
- The RB Mobility, core service of this prototype is constantly fed with real data, having developed several connectors and services, much designed for Madrid UC:

- Things of LINEBUSMAD, DRIVERSBUSMAD, BUSSTOPMAD: The loading is done directly from the EMT information systems, using real values and keeping the changes through the SQL Server Change Tracking service.
- Things of APPLICATION, USERS: Loading is done by observing the EMTIng Gamification database of EMT. This way, any subscribed application allows the registration, tracking and can receive PUSH notifications.
- Events of BUSSTOPMAD. Is fed in real tim by a DDP observer connected to the RB SAE thorugh the position layers of buses and bus stops.

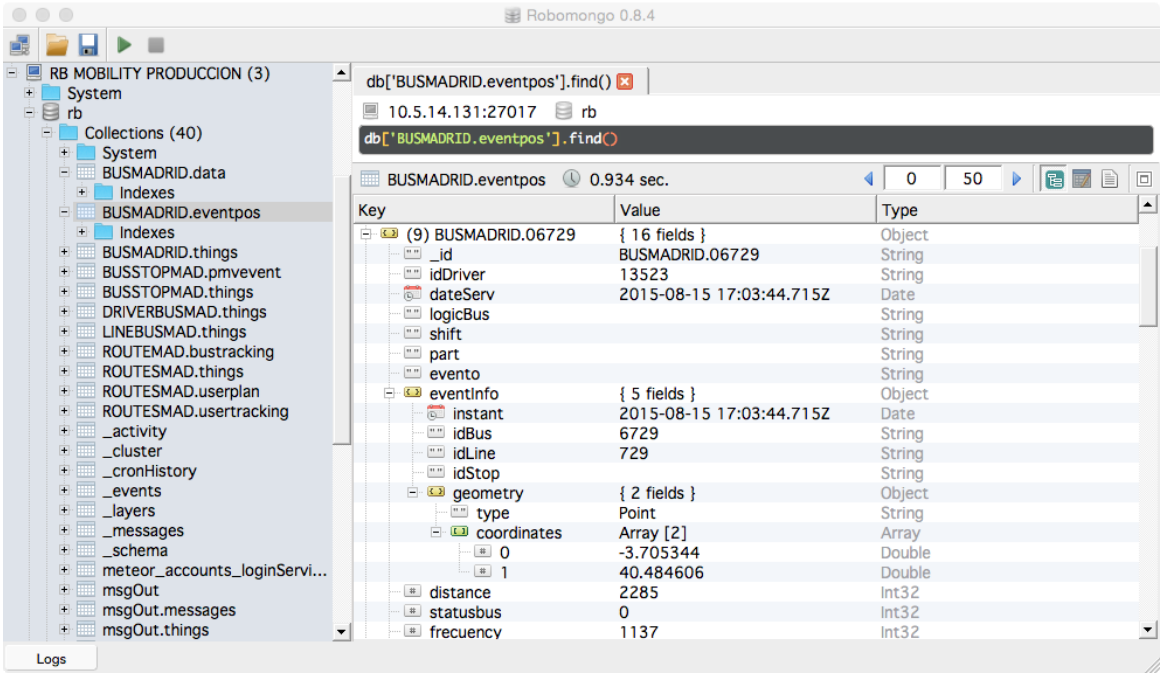


Figure 13. Organizational scheme of RB layers in MongoDB

2.1.3.1 New features for Y3. On Demand Core for autoload data in VE

2.1.3.1.1 System architecture for scheduling data loader

One of the problems encountered during the platform evolution of Y3 has been how to provide extended functionality to allow the system to be autonomous when making dynamic registrations of new VE or how to feed them at specific times of day, depending on specific characteristics or situations.

To achieve this, a service engine that allows accepting tasks in VEProt datagram format has been developed. These special datagrams contain a format that defines autonomous functions to run within the system and can also be requested from the outside, inside the Opendata model.

Thus, certain data calculation demand application functions are performed by the core of process instances at the request of an external system and for a specified time. An example of this may be to make a studio on the location of EMT buses of a bus line for a certain time (i.e. one hour per day). This process would be carried out at the request of an external system that wishes to measure how city traffic influence the misuse of the bus service regularity. All data is

stored by the system in the bigdata system and can be recovered through the massive data recovery system and be used for further analysis.

To do this, once again, it is used the only system point of entry and validation, which is the Rabbit server. Through it, and in agenda process format of the VEProt datagram a task is injected into the Reactive Box and is used from that time to perform iterative processes.

An outline of the acceptance and processing tasks can be seen below:

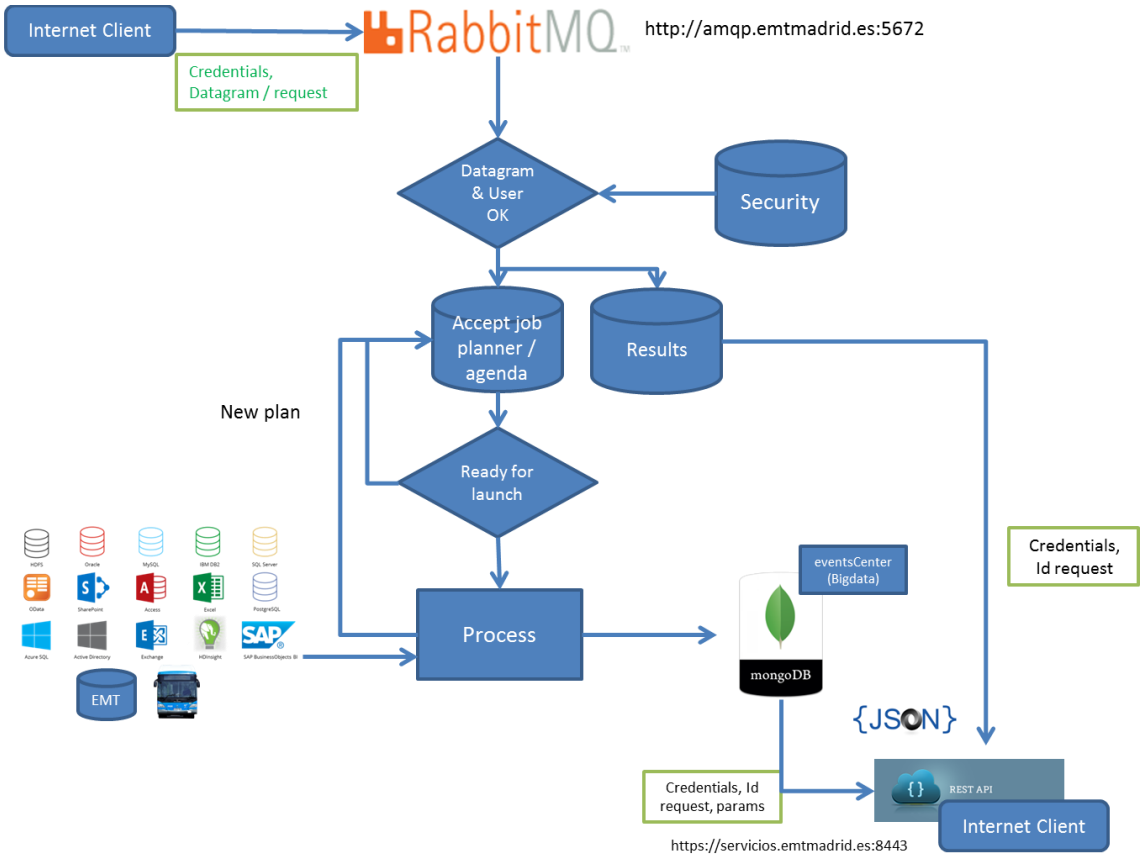


Figure 14. Aceptance and processing of tasks scheme

The stored data belong to the user that has processed them, though, as any data stored in a VE can be defined in PUBLIC or SHARED mode, to be shared.

Thanks to a special architecture, the system locates available resources on the Intranet in which the server infrastructure is installed. If user equipment of a network (end customer) is detected (and are registered as capable of being used to manage processes), the system use them to distribute the processing load, allowing parallel processing of multiple machines with concurrent processes in each one, depending on available resources, memory load and processor provided to the RB-COSMOS infrastructure.

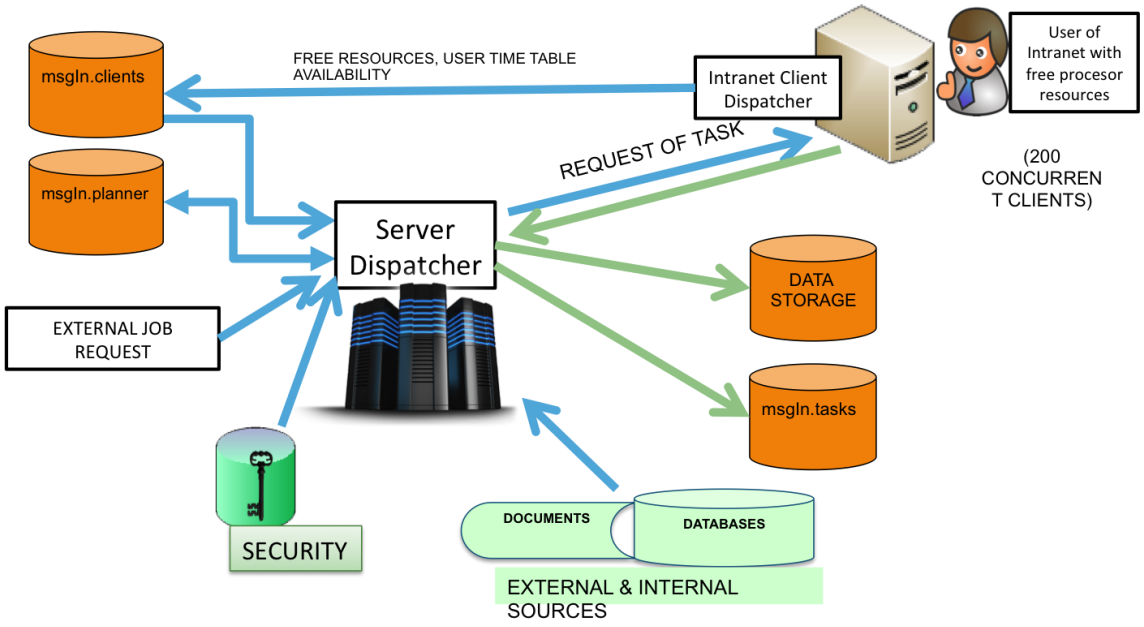


Figure 15. Scheme of background Server Dispacher

2.1.3.1.2 *System architecture for asynchronous recovery of massive data*

As was stated in previous deliverables, the COSMOS - Madrid SmartMobility architecture not only makes data management for observing the change of state in different VE observed by a client. It also stores state changes in a bigdata storage model. This implies that there is a mechanism of information retrieval that the logical storage of a VE supports.

The problem and the challenge presented and that has been solved during Y3 is how to use an standard API REST to retrieve dynamically and randomly huge volumes of information from a system connected from the outside. This has been solved by MongoDB DUMP mechanisms using the acceptance criteria by BSON selectors from the database format. These BSON filter selectors are accepted in API REST FULL to obtain data from any collection to which the connected client has access.

The DataProvider service through which massive data can be recovered is posted at the following URL:

https://rbdata.emtmadrid.es:8443/DataProvider

It contains a set of methods oriented to the interaction with the MobilityLabs systems. The most important of them is getCollection, which is published in:

https://rbdata.emtmadrid.es:8443/DataProvider/api/dmz/getCollection

In the COSMOS-MobilityLabs systems developed during this project, the first prerequisite to access to a VE is to have a user with access privileges to that VE. Regarding the rest of the parameters, those are used in the usual manner of the API REST, including the layer (VE) and name.

For example, the call to a collection called TRAFFICMAD.alarms would be:

```
https://rbdata.emtmadrid.es:8443/DataProvider/api/dmz/getCollection/YOUR_idClient/YOUR_passKey/Layers/TRAFFICMAD.alarms/
```

In addition, within the body of the call the parameters to be received in BSON format should be noted, and they must contain specific call filters.

For example, if the collection TRAFFICMAD.alarms would like to extract the value of the collection corresponding to the item with key "keyid" equal to PMV333 the MONGODB format filter notation would be { "keyid": "PMX333"}.

The filter values are combinable with all possibilities that MongoDB offers when extracting documents collections, as the system actually applies internally the filter for extraction once validated.

The general outline of the recovery system of massive data can be seen below:

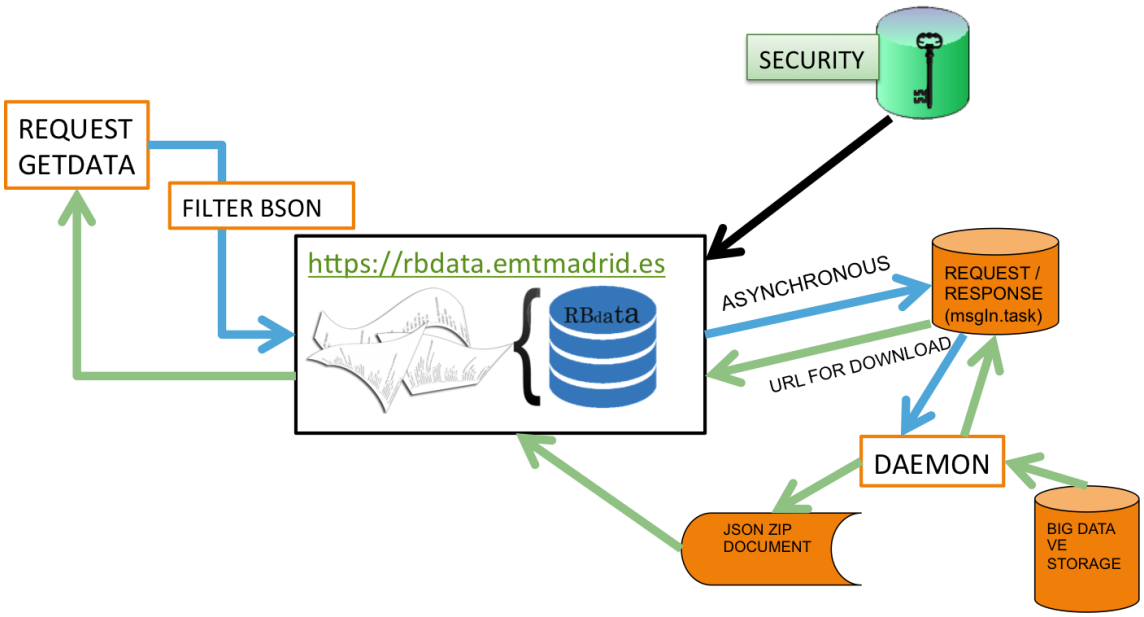


Figure 16. System architecture for asynchronous recovery of massive data

Since the information is retrieved asynchronously, the system stores the request record and the download url of the compressed file in which to store the data to retrieve, so that the connected system may apply for its information as many times as desired using the identifier of the request. That allows to store a compressed data repository in JSON format available as often as desired.

2.1.4. Registering Virtual Entities for SmartMobility

2.1.4.1 Introduction and requirements

Although the semantic model that will support the definition of VE has not been polished completely, a theoretical analysis of how the system will be developed it is presented. This task will be part of future deployment of COSMOS-MOBILITYLABS system within the scope of the Madrid SmartMobility.

First of all, we will conduct an analysis of requirements that justify why the VE registration is done this way, and why is based on the semantic model that is subsequently expressed.

- The information extracted through both the observation mechanism of the Reactive Box and the massive data extraction systems is dynamic and has not an associated Web, but associated service models.
- The information extracted is always in JSON format.
- In order to simplify the registration methods, it is forescasted the possibility of using the same methods to insert data into the VE using the VEProt datagram.
- Collecting this three types of information must be considered:
    - Pure data from the VE logic stores by observers or extractors.
    - VE semantic structures or schemes definitions by observers or extractors.
    - Heterogeneous data; that means information of VE stored data along with their semantic annotation.
- It should allow to know who is the owner of a particular VE and the data version which is stored.
- It should allow scaling to a semantic portal.

Because of these characteristics, the model under development will be represented by JSON-LD (http://json-ld.org/), as it will allow to store the VE ontology with the same structure and rules in which the different VE data are stored.

2.1.4.2 Registry system

As stated before, it will be possible to use the same use rules of VEProt datagram in order to register the semantic definition of a VE. To do this, simply define the base structure and insert it by calling through RabbitMQ. Schematically, we could define it as follows:

Attribute	Value	Property	URI	Description
layer	ROUTESMAD.alarms	mobilitymadrid:layers	http://mobilitylab.emtmadrid.es/def/layers#usereventmad	kind of layer into RB or Mobility Servers. Layer ontology describes data domain Reactive Box and other elements of this entity
_id	[_id]	Schema:text	http://schema.org/Text	Unique id of item
instant	[date time value]	dcterms:date	xsd:datetime	Instant of data generation
geometry	GEOJSON	point object	http://geojson.org/geojson-spec.html#id2	Geo-point of current position of user
idRoute	[_id]	Schema:text	http://schema.org/Text	Unique id of route plan
idSesion	[value]	Schema:text	http://schema.org/Text	Current session of user track
busData	[value]	JSON structure	JSON structure	data of bus when the user device is linked to

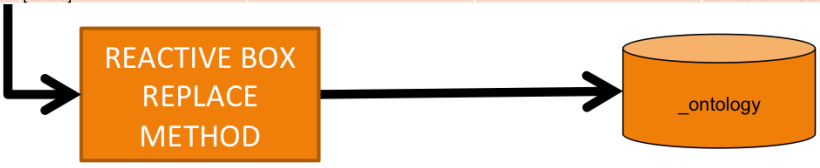


Figure 17. Conceptual model for registering new VE

As defined above, the same REPLACE method of the new VEProt Opendata datagram can be now used to register the semantics of a VE. Thus, any system can not only provide data to a VE but also register their definition, under the same rules.

The information is recorded in a RB logical store called _ontology and its structure is the following:

FiedName	Description	Format
_id	Unique Id (assigned by the system RB)	ObjectId
Collection	General Name of Virtual Entity	String
subCollection	Specific Name of Virtual Entity	String
Description	Description of Virtual Entity-Version	String
Datetimegen	Date time of Registry	Datetime
Owner	Owner of this version of VE (must be exists into user definitions)	String
Type	Mode of use (public = everybody read, shared=everybody read/write, private= only for use by the owner)	String
Version	Version/subversion of current datatype in VE	String
Uri	Main uri point for getting data or definition	String
Attributes	Array Object which contains de full structure of VE	(Container)
Attibute	Item of Attributes object. Defines the name of each item	String
Value	Item of Attributes object. Defines the fix value or the specific datatype of current attribute	String
Property	Item of Attributes object. Defines the schema definition in RDF semantic	String
URI	Item of Attributes object. Defines the URI of current attribute	String
Description	Item of Attributes object. Description of current attribute	String

Table 4. Data model for semantic definitions of VE

In the following example, an alleged ontological scheme for a VE called BICISENS.temperature is defined, in which temperature values of a sensor connected to one of the public bikes of Madrid are stored, being managed by EMTMADRID.

```
{
  "_id": ObjectId("576b103b213ef41bb57c4cd1"),
  "collection": "BICISENS",
  "subcollection": "temperature",
  "Description": "This layer contains a example of how you can put and get
data into MOBILITYLABS systems. The layer BICISENS.temperature",
  "datetimegen": "2016-06-15T09:00:00",
  "owner": "EMTMADRID",
  "type": "share",
  "version": "1.00",
  "uri": "http://resources.emtmadrid.es/bicisens/temperature/",
  "attributes": [
    {
      "Attribute": "_id",
      "Value": "[value]",
      "Property": "Schema:text",
      "URI": "http://schema.org/Text",
      "Description": "Unique id of sensor into system"
    },{
      "Attribute": "instant",
      "Value": "[date time value]",
      "Property": "dcterms:date",
      "URI": "xsd:datetime",
```

```
      "Description": "Instant of data generation"
    },{
      "Attribute": "geometry",
      "Value": "GEOJSON",
      "Property": "point object",
      "URI": "http://geojson.org/geojson-spec.html#id2",
      "Description": "Geo-point of current position of sensor"
    },{
      "Attribute": "accuracy",
      "Value": "[value]",
      "Property": "schemas:Text",
      "URI": "https://schemas.org/Text",
      "Description": "Class or characteristics of value"
    },{
      "Attribute": "shape",
      "Value": "[value]",
      "Property": "structure object",
      "URI": "object",
      "Description": "for painting in map (icon size color)"
    },{
      "Attribute": "acelerometer",
      "Value": "value",
      "Property": "schemas:Number",
      "URI": "https://schemas.org/Number",
      "Description": "Section of route where bus is position in current time"
    },{
      "Attribute": "battery",
      "Value": "[idSection]",
      "Property": "schemas:Number",
      "URI": "https://schemas.org/Number",
      "Description": "Section of route where bus is position in current time"
    },{
      "Attribute": "socket",
      "Value": "[value]",
      "Property": "schemas:Text",
      "URI": "https://schemas.org/Text",
      "Description": "Name of device or sensor"
    },{
      "Attribute": "value",
      "Value": "[value]",
      "Property": "ssn:MeasurementCapability",
      "URI": "http://purl.oclc.org/NET/ssnx/ssn#MeasurementCapability",
      "Description": "Characteristics of meassures and context conditions"
    },{
      "Attribute": "state",
      "Value": "[value]",
      "Property": "structure object",
      "URI": "object",
      "Description": "for painting in map (popup container)"
    },{
      "Attribute": "busData",
      "Value": "[value]",
      "Property": "JSON structure",
      "URI": "JSON structure",
      "Description": "data of bus when the user device is linked to"
    }
  ]
}
```


As exposed, the defined format for the semantic definition of VE in the field of Madrid SmartMobility can follow the same registration rules that any subsequently data feed.

Integration MobilityLabs-VE-Registry into COSMOS-Semantic-VE-Registry

One issue to be analyzed in detail within the full integration model of general publication of the semantic model of COSMOS is to establish rules that automaticly feed the COSMOS semantic model using, for the case of Madrid, the _ontology scheme already defined. To do this, the main objective now is to define the charging system (bulk copy or massive script) to allow generating, as a background process, the ontological definition that COSMOS requires in its operative scheme.

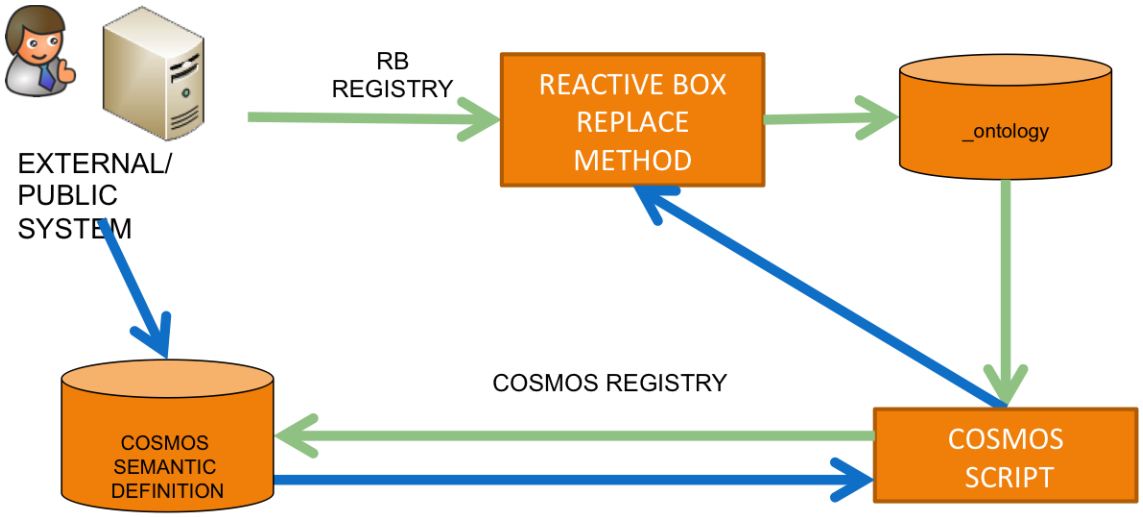


Figure 18. Conceptual flow of how integrate RB VE Registry and COSMOS VE Registry

3. Definition and design of the different components of Madrid UC prototype

3.1. Introduction

This section describes the different components that have been used in the prototype, either by technological reuse, indirect developments or adhoc developments for the prototype. All the development has involved technology, hardware and software that can be conceived as a unit about the overall vision of the prototype.

First of all, to make the development, it has been necessary to make a studio of existing technology and data sources within the mobility infrastructure of Madrid city, and to check the feasibility of its integration, analyzing contents, formats and availability of information. Finally, specific connectors have been developed for those currently existing subsystems to be integrated.

The final result is a set of programs and servers that integrate existing technologies and data into new developments designed for COSMOS infrastructure, without losing perspective that this system may act as a platform for additional future systems. Hence, the importance of taking into account from the beginning the requirement to reuse and open information.

3.2. Work team

The need for collaboration across multiple disciplines to prepare the prototype has meant extra work beyond the conception of a purely formal work team. Surely, the information found through specialized Internet sites is an essential support without which it would be hard to achieve developments of this nature. Moreover, the contribution of experts -vision and advice- through the Madrid mobility laboratory has also been essential.

Regarding the software, it has been mainly developed by EMT Madrid team, who brings its expertise in developments (C#NET[6], Python 2.7[7], Transact SQL[8], JavaScript[9]) and infrastructure management (MongoDB, SQL Server[10], Internet Information Server[11], Apache[12], Meteor), as well as thorugh the technological experts of COSMOS consortium, whose developments in the field of CEP + Message BUS and Machine Learning models are essential for the UC.

3.3. Components used in COSMOS Madrid UC prototype

3.3.1. EMT MADRID opendata platform[13]

The EMT MADRID Opendata platform consists of a set of SOA and JSON technologies web services that provide data sets based on:

1. Information related to the planning of service (timetables, routes, bus lines, bus stops, etc.). This information is available at:
   https://openbus.emtmadrid.es:9443/emt-proxy-server/last/bus
   http://servicios.emtmadrid.es:8443/bus
2. Information related to the geographical position, such as nearby bus stops, streets, as well as some advanced planning services and the estimated arrival time of a bus (real time) to a certain bus stop:
   https://openbus.emtmadrid.es:9443/emt-proxy-server/last/geo
   http://servicios.emtmadrid.es:8443/geo

3. Contents with specific value, such as a service to estimate the arrival of buses to bus stops in real time with extended information about incidents and a method for obtaining planned walking and bus routes between two points of the city.
https://openbus.emtmadrid.es:9443/emt-proxy-server/last/servicemedia
http://servicios.emtmadrid.es:8443/servicemedia

   All the associated documents to the aforementioned services is found in
http://opendata.emtmadrid.es/Documentos/Opendata-v-1-12.aspx

   To develop the prototype of the monitoring portal for people with special needs several connectors have been developed with .NET[14] technology towards the services of the EMT Opendata platform in order to obtain planned routes for the time periods selected. It has also been necessary to create connectors for calendar services and times of bus lines in service.

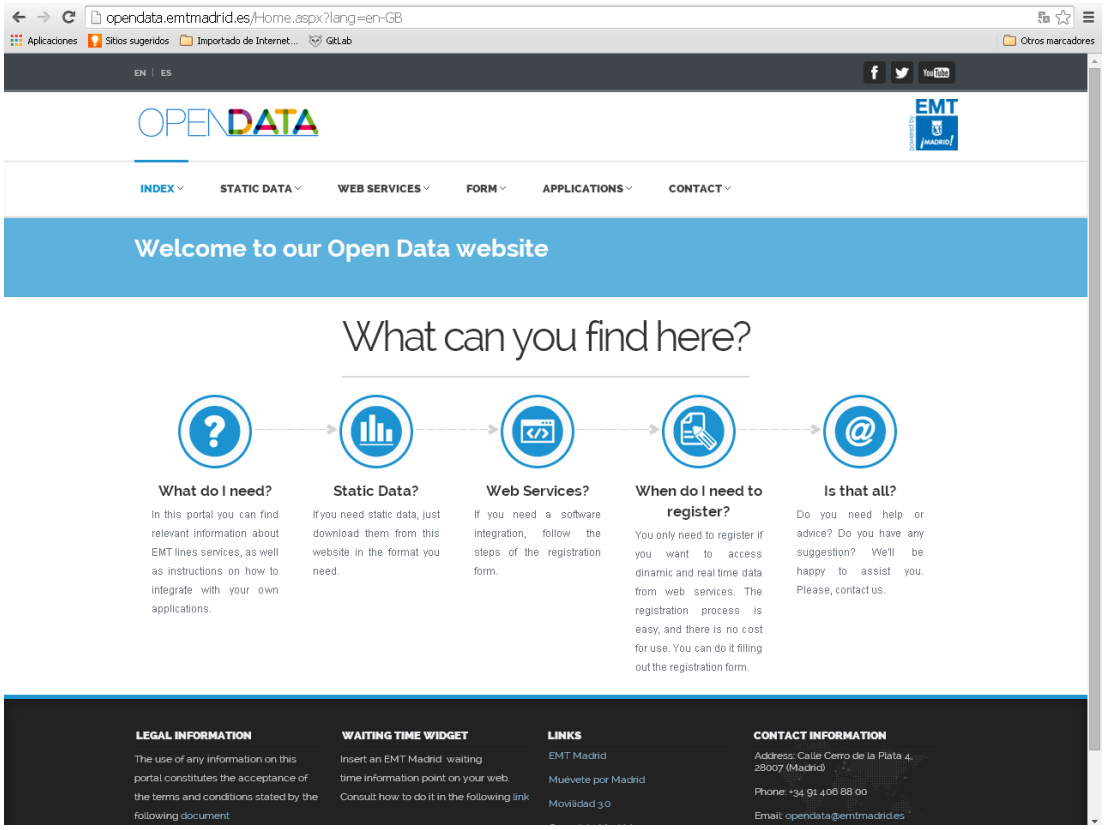


Figure 19. EMT Opendata portal access interface

3.3.2. Madrid City Council open data platform

To obtain information on the intensity of traffic the development of a connector with published traffic data within the RB platform is required, but for now, integration of the prototype has been performed using a direct connexion between the Message bus with the XML [15]traffic service. The location of these data is in http://datos.madrid.es/egob/catalogo/202087-0-trafico-intensidad.xml and the documents are published in http://datos.madrid.es/FWProjects/egob/contenidos/datasets/ficheros/Trafico_descripci%C3%B3n_campos.pdf

Figure 20. Madrid City Council Opendata portal (Datos abiertos)

3.3.3. Specifications of bus emulation server

The purpose of undertaking the construction of a bus emulator has been to be able to debug any software from Madrid Mobility Lab that needs the onboard Wifi connectivity. As described in previous deliverables, EMT buses have an Opendata web service in each vehicle, which provides information about the activity in real time about himself and about the bus line that serves. To get this content, a client must be connected to the wireless network of the bus (EMT-Madrid) and solve the following URL:

https://172.18.2.12 /rests/?srv=[nombre servicio] &[parámetros]

3.3.3.1 Emulation environment

The main challenge for any developer is the need to be "physically" present on the bus in order to establish connectivity to data provided by the vehicle, preventing a coherent and rapid debugging of an application. This fact raised the need for an emulated environment allowing the connexion with any running vehicle, or with a virtual vehicle that is permanently circulating in a bus line.

The bus emulator URL is:

https://mybus.emtmadrid.es:8073/rests

idCliente: EMT.SERVICIOS.OPENBUS

passkey: A2C983E5-5BA3-41F3-B47C-428F467041DC

By incorporating the standard parameters described above, the system provides information of a "virtual" vehicle with the number 1990 which is circulating in line 1 of EMT at a constant speed.

In addition, if added to either of the two methods a new parameter called "bus", the service returns the actual information of the requested bus, as long as this bus is in service. This ensures that any application is able to emulate in purification phase or even in production phase the fact of being in a vehicle physically and connected to the wireless access point of the vehicle.

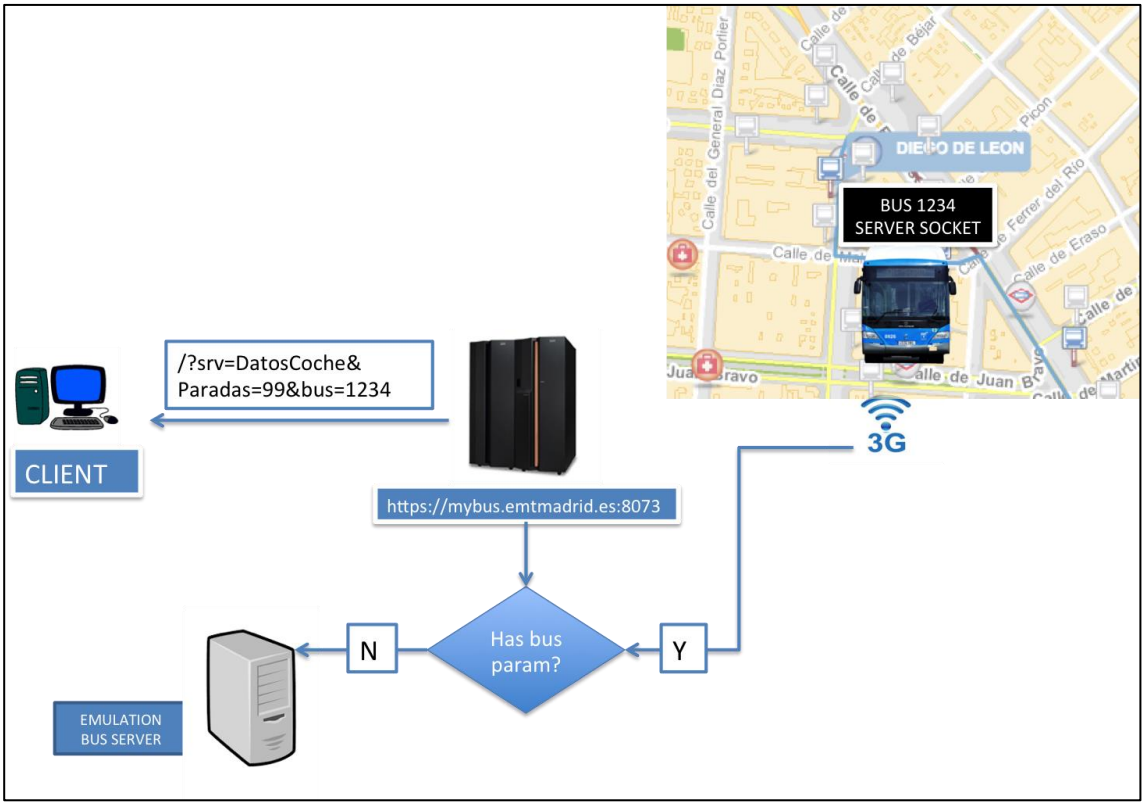


Figure 21. Functioning scheme of platform for bus emulation

The MyBus system is used in the prototype to emulate the position of the person with special needs during his/her trip, from the approach to a bus stop until the end of the bus trip. To do this, the client application locates the nearest vehicle to the position of the person with special needs in their "emulated" approach and incorporates this person into this vehicle, linking the position of the person with special needs to the real bus in service. This fact is described in the section 3.4.4 “SP user interface prototype and SP route simulator”.

Srv can contain the following values:

DatosCoche: It is used to obtaing the performance of the service as a method that offers the information about the vehicle activity on that precise momento. It accepts the parameter “paradas” (stops) with a whole number bigger than 0.

The process returns a XML document with the main node named “DatosCoche” and the following subnodes:

- vehiculo Numerical value with the identifier of the bus which is offering the content.
- linea Numerical indentifier of the bus line in which the vehicles is providing service.
- sublínea Numerical indentifier of the bus subline in service.
- viaje Number of trip within that certain bus line in which the bus ir providing service
- sentido Direction of the trip (1 for the onward trip and 2 for the return trip)
- estado State of service.
- horario Subnode without a value. When it appears means that the bus line is being regulated by time instead of frequency.
    - desfase Mismatch of the bus from the theoretical timetable (in seconds). Positive values mean ahead schedule.
    - desfase_ref Mismatch of the bus from the reference timetable (in seconds). Positive values mean ahead schedule.
    - posicion Bus position. If it is located, it is the position within the bus route associated to a certain direction. Any other state means that it is the distance covered since it changed to that specific status.
        - gps GPS position. It has the following attributes:
        - est UTM position towards the East in metres (X)
        - nor UTM position towards the North in metres (Y)
        - alt Altitude over sea level in metres
        - zone UTM zone of the position
        - band UTM band of the position
- paradas Table with the requested bus stops. It contains a list of subnodes with the name “parade”. The information of each bus stop is included into the node attributes.

The “stop” nodes that inform about each next bus stop have the following attributes:

- codigo Numerical code of each bus stop
- x X UTM coordinate of the bus stop, in metres
- y Y UTM coordinate of the bus stop, in metres
- distancia Distance to the bus stop from the current position, in metres
- hora Estimated arrival time to the bus stop. It is a numerical value with the format hhmmss (hour, minutes, seconds).

DatosParada: The second method available in the bus offers information about the complete route of the bus line in service and its definitions. When invoking, without any parameter, it gives back an XML document with the principal node named “DatosParada” and the following subnodes.

- linea Numerical identifier of bus line
- sublinea Numerical indentifier of bus subline
- label Alphanumeric label of bus line (up to 5 characters)
- sentido Direction in which the bus is located. It is sent only when the bus is located within a bus line.
- secciones List of sections that form the route of a bus line. It contains a subnode list named “sección”

The “seccion” nodes contain the following attributes:

- Atributo Description
- codigo Numeric code of the section
- distancia Direction in which the section is located
- longitud Lenght of the section in metres
- nombre Alphanumeric chain with the section name

In addition to this, each section contains a list of nodes with the name “paradas”, which content is the name of each of the bus stops. The name is coded in UTF-8[16].

Finally, each one of these nodes contains these additional attributes:

- codigo Numeric code of the bus stop
- cabecera If it appears has always the “true” value and means it is the starting point of a bus line
- posición Position of the bus stop within the section, in metres
- x X UTM [17]coordinate of bus stop in metres
- y Y UTM coordinate of bus stop in metres

3.3.4. Madrid Bus SDK API

The availability of an SDK for developing applications for Android and IOS [18]facilitates the work of a developer in integrating certain systems, besides encapsulating functionalities exposing only methods and public functions. The EMTIng SDK is designed to integrate multiple applications in an environment connected to the Madrid buses.

The prototype only considers the possibility of including it in the final product. However, we should describe the mode of operation of the SDK [19]in order to better understand what it will be the connection and data exchange between the application of the person with special needs and the platform connected to COSMOS.

The Madrid BUS SDK locates and links the bus SSID locates and bus links, authenticating itself through them in the EMTIng management platform. Once connected, and in an automatic way, the SDK provides all the services defined above in MyBus platform, in addition to many other functions related to user activity. Among others:

1. Ability to receive PUSH notifications from the platform: This can be useful when it comes to advice to the person with special needs that there’s been a deviation from the route or that this person with special needs may get off at the next stop, among other possibilities. For this, the SDK contains various methods for configuring the app to integrate PUSH Notification.

```
- (void)application:(UIApplication *)app
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)devToken
        {
[EMTing
didRegisterForRemoteNotificationsWithDeviceToken:devToken];
        }

- (void)application:(UIApplication *)app
didFailToRegisterForRemoteNotificationsWithError:(NSError
*)err
        {
```

```
        [EMTing
        didFailToRegisterForRemoteNotificationsWithError:err];
               }

        - (void)application:(UIApplication *)application
        didReceiveRemoteNotification:(NSDictionary *)userInfo
               {
        [EMTing didReceiveRemoteNotification:userInfo];
               }


               - (void)applicationWillResignActive:(UIApplication
        *)application
               {
        [EMTing setAppActive:NO];
               }

               - (void)applicationDidBecomeActive:(UIApplication
        *)application
               {
        [EMTing setAppActive:YES];
```

2. The SDK also provides a notification service with updated information from Open BUS Data services: This information will show in a own view of the SDK.

   The notifier also has the option of giving the information with voice synthetic.

   By default, the SDK will have both options disabled (the view and voice).

   To activate the view option or voice option, there is a method in the SDK that you must integrate in the didFinishLaunchingWithOptions in your AppDelegate.m:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
[EMTing configureNextStationNotifierShowingView:YES
andPlayingSound:YES];
}
```

3. Methods for getting the updated information about Open Bus Data service:

   This methods will provide the updated information from Open BUS Data service.

   For the correct integration, there is a method which let you know if the user is on BUS.

   Remember to use this method if you need to obtain information from the Open BUS Data service:

```
BOOL isOnBoard = [EMTing isOnBoard];

if (isOnBoard)
{
   …

}
```

If [EMTIng isOnBoard] returns YES you will use the next methods for obtain service information:

- User on board EMT line
  [EMTing getBUSLine];

- BUS Number:

[EMTing getBUSNumber];

- Next station name in route:

[EMTing getNextStationName];

- Next station distance in route:

[EMTing getNextStationDistance];

- Next station arrival time:

[EMTing getNextStationHour];

- List of n next stations in the route:

This method returns a NSMutable object with the n stations requested by parameter. If there would be less stations in the route, the method returns the number of available stations at that moment

To implement this method the developer could use the public object called EMT_stationBUS which it will be able to access to the station properties: nextStationCode, nextStationDistance, nextStationHour, nextStationName, nextStationLine.

```
NSMutableArray *arrayStations = [EmTing getArrayStations:n];
```

3.4. Components developed in the Madrid UC prototype

At this point, the different modules and systems that have been developed for the prototype of Madrid UC will be described. Each part or component developed is oriented to provide a specific function to the system; in some cases supplying final functionalities still to be developed. For example, the burden of datagrams in the message queue has been developed with a WCF [20]web service that provides limited functionality regarding the number of concurrent messages. However, on the final model to be implemented, this functionality will be deployed on a Rabbit MQ Server[21]. In other cases, such as the person with special needs interface we have opted for a Windows application considering the intrinsic architecture model of EMT, although in the final app will be Android.

As for the necessary functionality, the prototype largely covers the functional flows proposed in the use cases.

3.4.1. Data loading service into the message queue

As mentioned in the previous point, the ultimate goal of the VEProt datagram charging process into the Reactive Box is to do it through message queues in Rabbit MQ. However, a datagram transport service has been developed whose purpose is to load the VEProt datagram storage MsgOut collection through conventional web services.

The service is posted on the following site:

https://servicios.emtmadrid.es:8443/ReactiveBox/VEProtocol/Service.asmx

And it contains the following methods:

putVEProtDatagram: Through this method it is possible to insert a datagram in the MsgOut message queue. The process returns the unique identifier of the inserted datagram. Its only parameter is a JSON object with the datagram, supplied through the VEProtDatagram parameter.

getVEProtDatagram: Using this method an entire datagram in JSON format can be recovered using as a parameter the identifier of a datagram. Its only parameter is idDatagram.

The infrastructure is based on a service model in two layers. The business logic for processing and insertion of the message datagram in MsgOut is in the inner layer, not accessible through Internet.

To accommodate the development, the EMT Opendata platform for public services has been used, in order to take advantage of resources and the existing SSL platform.

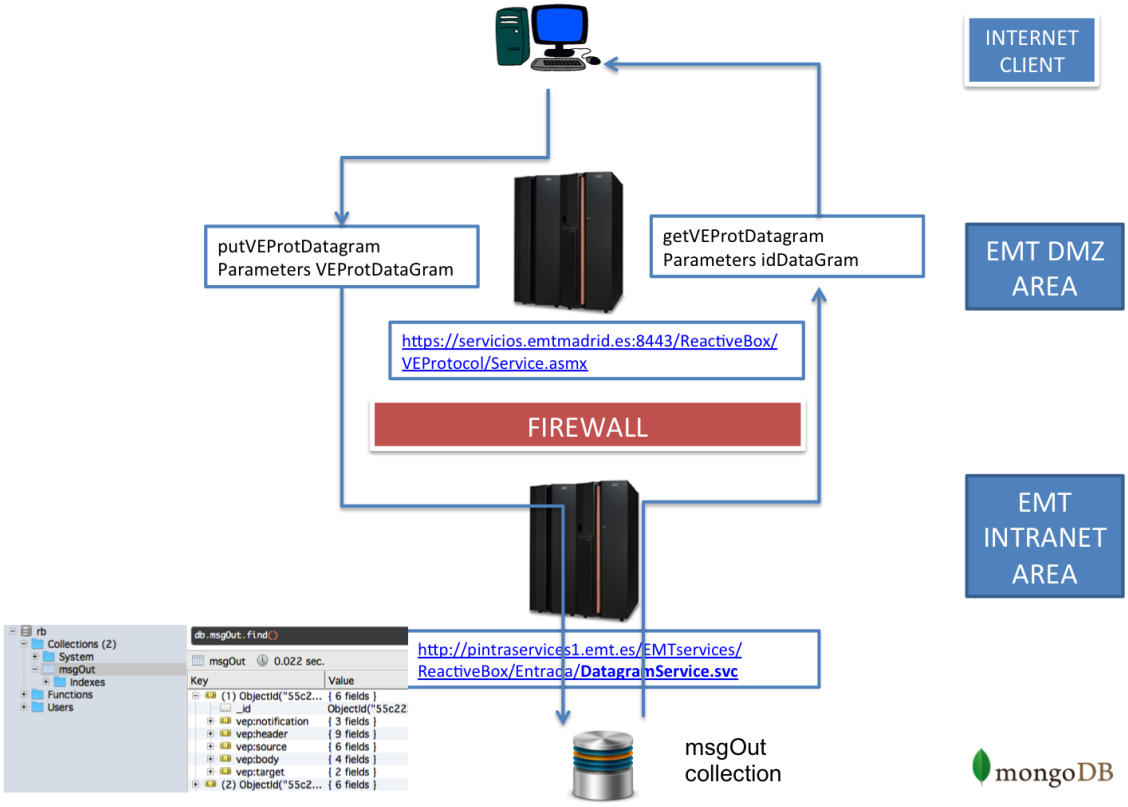


Figure 22. Service scheme of VEProt datagrams insertion into MsgOut

Besides security, the advantage of having a two layers service architectural model, is that it allows direct calls from EMT own Intranet and it integrates datagrams from other trusted providers through APIs from within the EMT server infrastructure without having to develop components in unprotected areas.

3.4.2. Special person Web portal for planning and monitoring

To manage the planning process and the monitoring of routes for a person with special needs, a ASPX [22] website with IIS engine has been developed. This option is conditioned by the availability of experts in this technology within the EMT systems department and by the EMT own server infrastructure, many of which are based on Windows Server architecture.

The design of the Web application is modular and contains sub-functions that allow to scale the solutions in two ways:

1. Scalable: adding new features, so that the prototype can become the final productive system.
2. Adaptive: admitting solutions based in multiple requirements or circumstances, as befits to the IoT COSMOS model.

3.4.2.1 Web portal general description

The overall project has been conducted under .NET platform using C# language primarily for connecting to MongoDB, making queries, saving collections and creating hidden controls so later on, in the client side, the information about the routes is stored on that controls. Finally, go through those controls that contain the information and store them under collections in MongoDB. In addition, LEAFLET [23]libraries are also used (open-source JavaScript libraries for interactive maps) to display and manage routes in the client side.

This website has several parts: the route management (subscribing, unsubscribing, route management) and the monitoring of a chosen route.

For the "route monitoring" part, it has been used METEOR, a JavaScript platform that allows reactive sites in real time, so that a web is synchronized with the server data without having to postback (reload page).

For the code on the client side JavaScript / jQuery [24]has been used.

MongoDB has been the system chosen as database; it is a document-oriented (NoSQL[25]) system, and these documents are stored in BSON [26](binary representation of JSON)

3.4.2.2 Login

The login portal requires a logon and password. User must be related to the person with special needs (caregiver, etc.), so they belong to the same user group but with a different user role (currently, for the prototype, the user "person with special needs" and "caregiver" are the same identity).

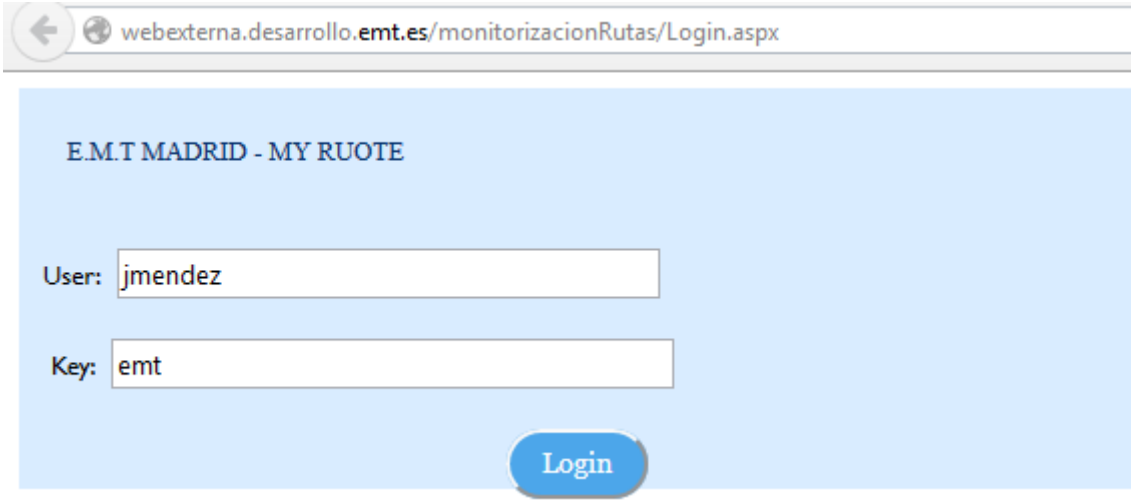


Figure 23. Login portal

3.4.2.3 General Screen of the monitoring portal

Once the access is granted, the monitoring portal prototype presents the available options: adding a planned route, delete it, view it and to monitor the person with special needs when making the route.

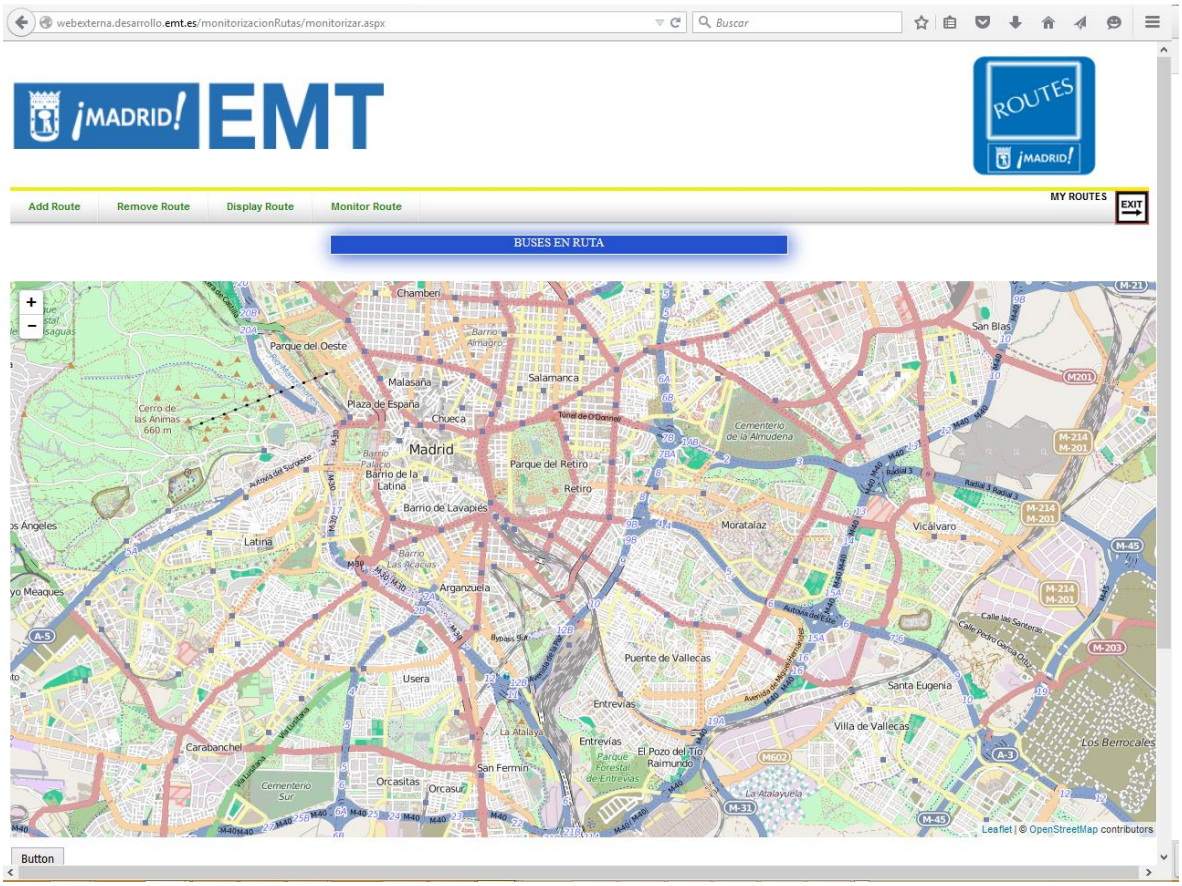


Figure 24. Overview of the speial persons route planning website

3.4.2.4 Management of the person with special needs planned routes

The "SUBSCRIBING" display consists of several .NET forms, and between them, a JavaScript script showing a LEAFLET map where the starting and final route point are set using the computer mouse.

The form asks for the following data:

1. A symbolic name for the route/itinerary (hospital, school, home, etc.), and the symbolic name for the starting and final destination point.
2. Dates (period) when each specific route must be active or activated (when it is going to be made).
3. Hours of the day (from-to) when the path is usually performed.
4. Weekdays: Within a date range, the user can travel only certain days (eg. only Monday or Mondays and Wednesdays, etc).
5. Type of day: Within the week, the user indicate to the system what kind of day a certain route is done (weekdays, Saturdays, holidays)

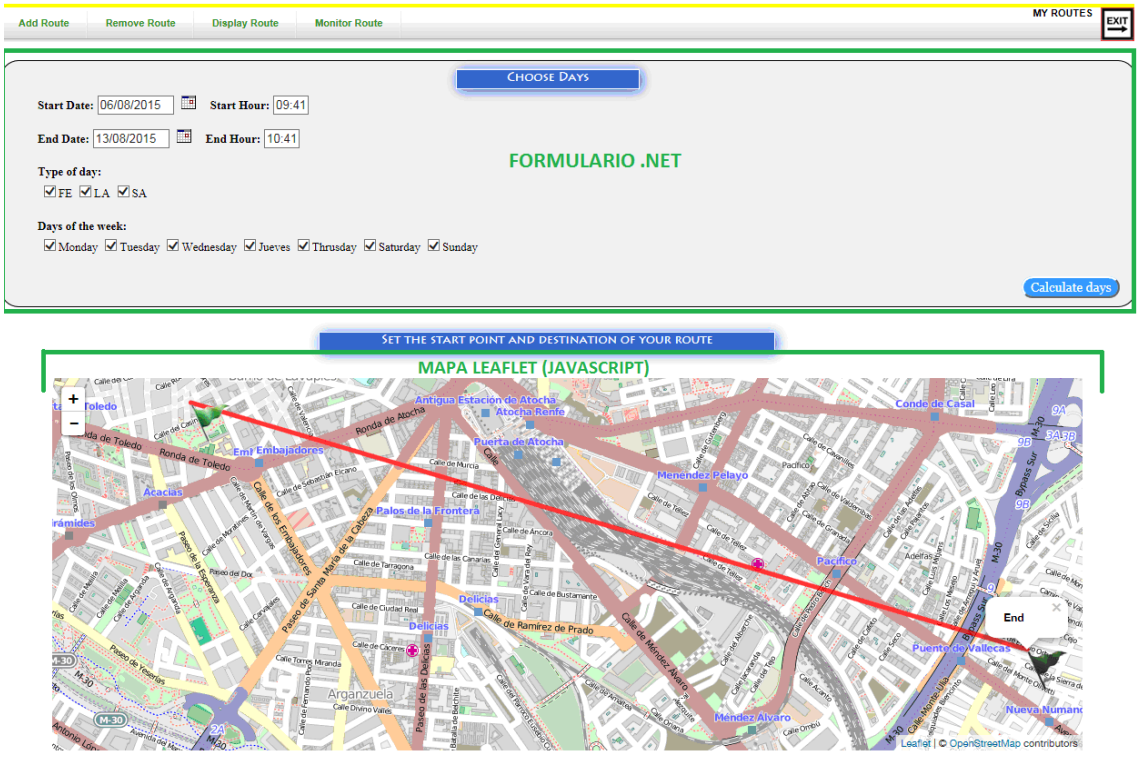


Figure 25. Route planning display

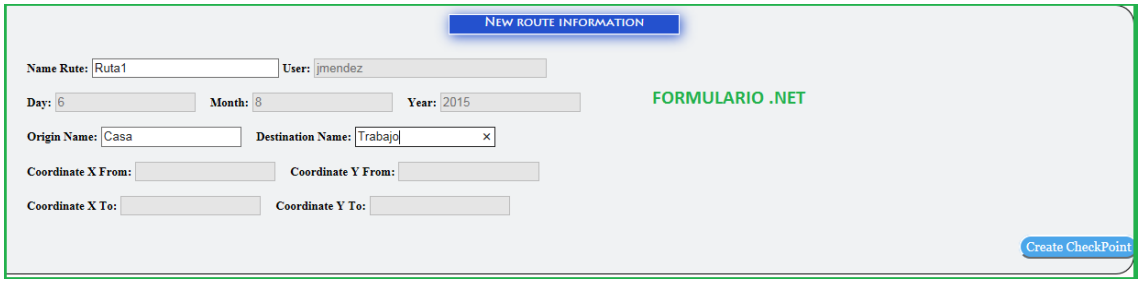


Figure 26. Indicating the route general data

Subsequently, once clicking on "`Create CheckPoint`" the system will call the EMT Opendata web service…

https://openbus.emtmadrid.es:9443/emt-proxy-server/last/geo/GetRouteLinesRoute.php

…providing the trip preferences and the starting and final points. This service returns a JSON object with route details (distance, distance to bus stops along the route, information about the various sections of route, etc.) and leads us to the display where to establish the different Checkpoints of the route.

Once in the aforementioned display, the different Checkpoints are established (checkpoint.aspx) and object elements are stored in the client, which contains data on the route and are drawn on the map as layers (JavaScript script).

NOTE: Everything drawn on the map can be consider a layer; there are on route Checkpoint layers, but also at bus stops, or in a walking intinerary, in a bus itinerary, etc. each one with its own information

On route checkpoint	
Stop checkpoint	
Bus stop checkpoint	
Walking checkpoint	
Bus checkpoint	

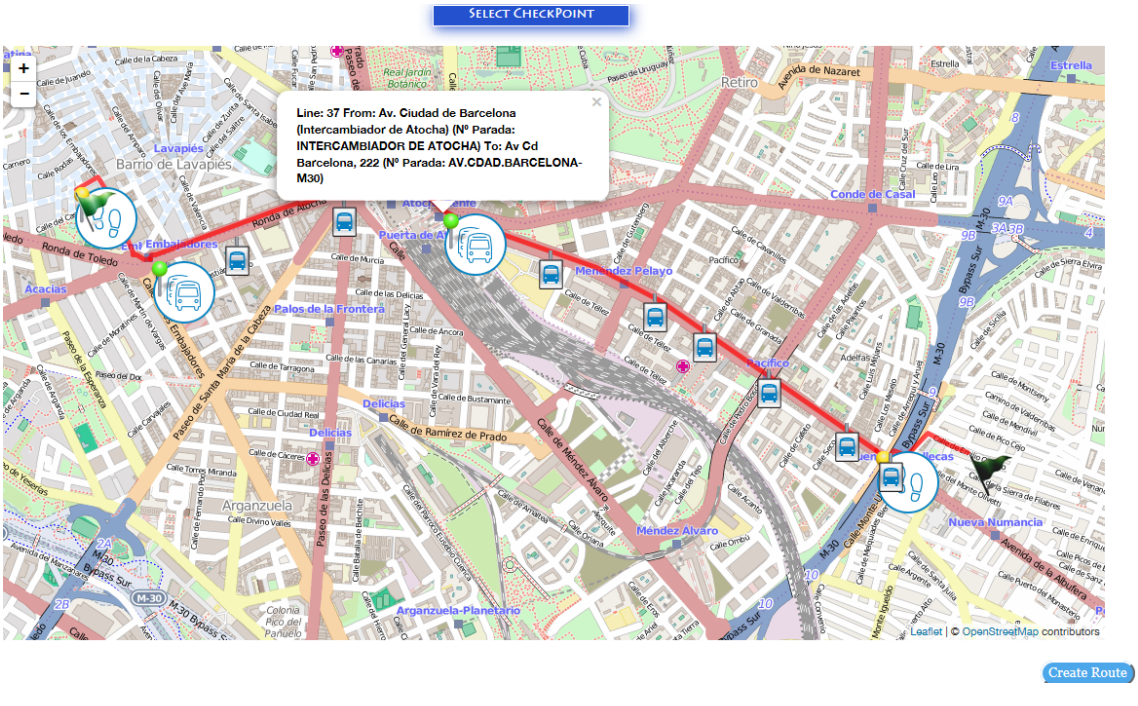


Figure 27. Creating the route checkpoints for the CEP

Once the details of the selected route are painted on the screen, the different checkpoints can be established by clicking on the map (in client mode); the checkpoint will be set as “on route” if clicking out of a layer, otherwise checkpoints can be selected as bus stops, etc. It is also possible to remove checkpoints by clicking on each Checkpoint layer.

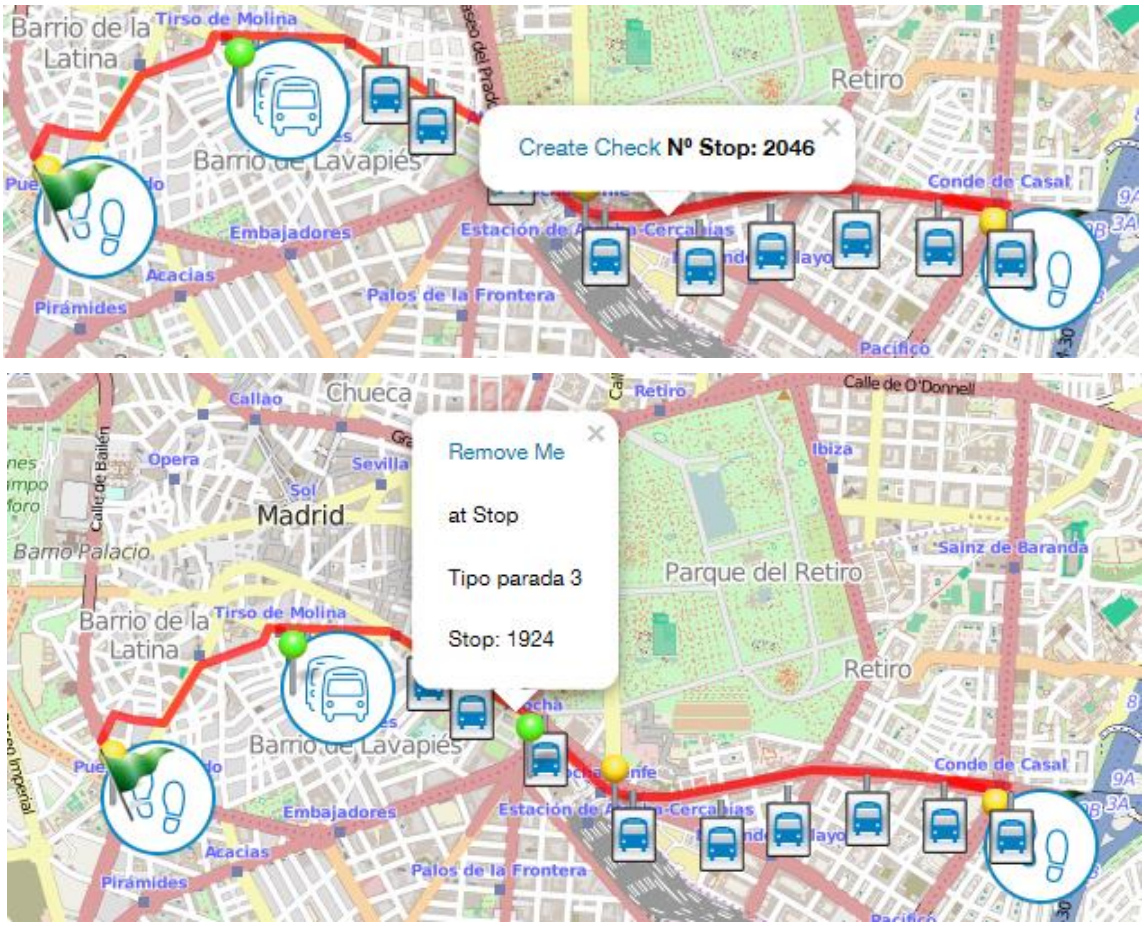
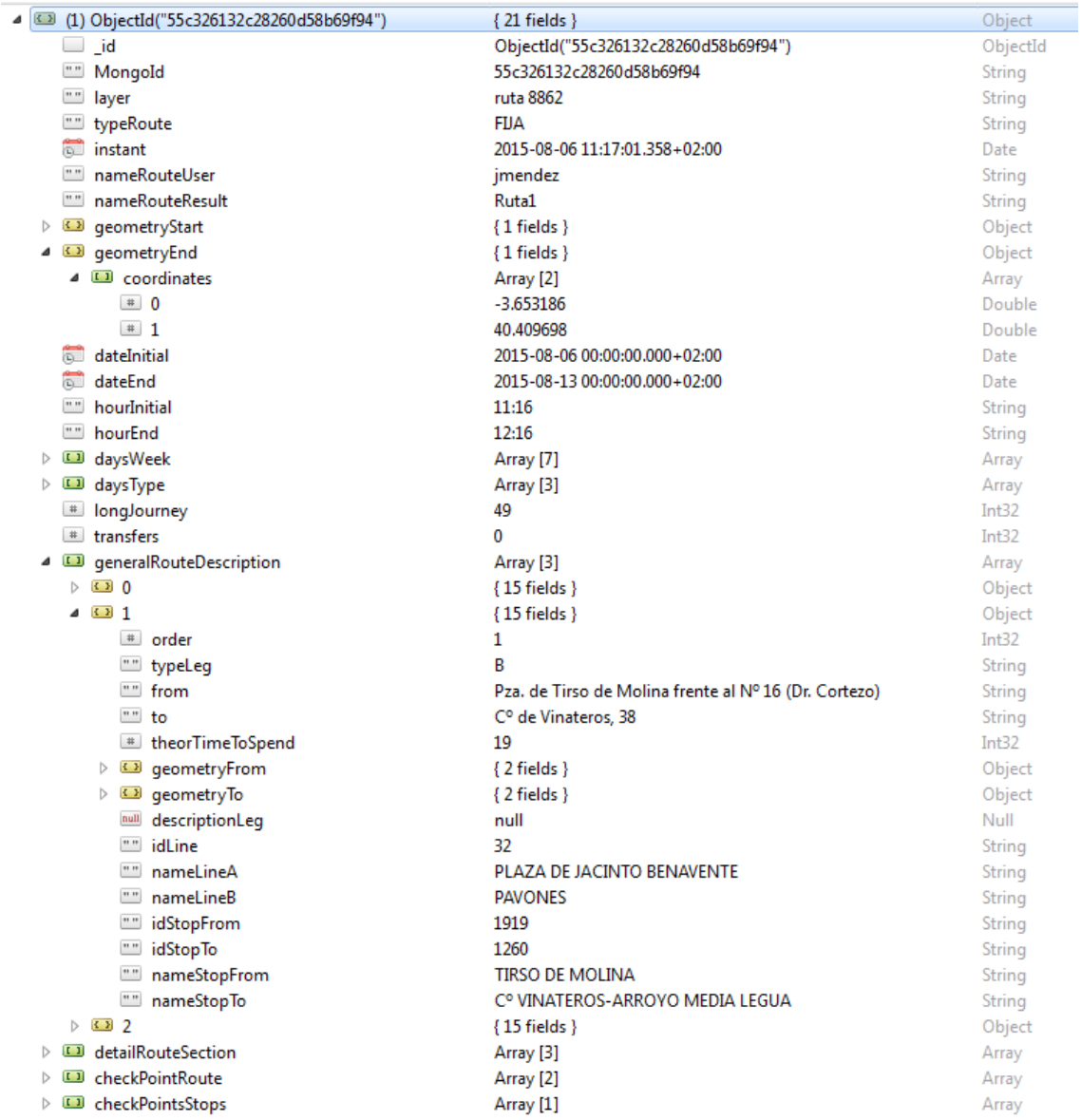


Figure 28. Creating and removing checkpoints

Checkpoints will be later used by the CEP to release data on average time of buses passing by at a specific location on a selected route in a certain day and time slot. Alarms may be set in case of “no appearance of the person with special needs" or "time deviation according to forecasted route", for instance.

By clicking on “`Create Route”,` once all the checkpoints are set up, all route data will be stored in a MongoDB collection with the following structure:



(1) ObjectId("55c326132c28260d58b69f94")	{ 21 fields }	Object
_id	ObjectId("55c326132c28260d58b69f94")	ObjectId
MongoId	55c326132c28260d58b69f94	String
layer	ruta 8862	String
typeRoute	FIJA	String
instant	2015-08-06 11:17:01.358+02:00	Date
nameRouteUser	jmendez	String
nameRouteResult	Ruta1	String
geometryStart	{ 1 fields }	Object
geometryEnd	{ 1 fields }	Object
coordinates	Array [2]	Array
0	-3.653186	Double
1	40.409698	Double
dateInitial	2015-08-06 00:00:00.000+02:00	Date
dateEnd	2015-08-13 00:00:00.000+02:00	Date
hourInitial	11:16	String
hourEnd	12:16	String
daysWeek	Array [7]	Array
daysType	Array [3]	Array
longJourney	49	Int32
transfers	0	Int32
generalRouteDescription	Array [3]	Array
0	{ 15 fields }	Object
1	{ 15 fields }	Object
order	1	Int32
typeLeg	B	String
from	Pza. de Tirso de Molina frente al Nº 16 (Dr. Cortezo)	String
to	Cº de Vinateros, 38	String
theorTimeToSpend	19	Int32
geometryFrom	{ 2 fields }	Object
geometryTo	{ 2 fields }	Object
descriptionLeg	null	Null
idLine	32	String
nameLineA	PLAZA DE JACINTO BENAVENTE	String
nameLineB	PAVONES	String
idStopFrom	1919	String
idStopTo	1260	String
nameStopFrom	TIRSO DE MOLINA	String
nameStopTo	Cº VINATEROS-ARROYO MEDIA LEGUA	String
2	{ 15 fields }	Object
detailRouteSection	Array [3]	Array
checkPointRoute	Array [2]	Array
checkPointsStops	Array [1]	Array

Figure 29. MongoDB structure for checkpoints list

Any stored route can be selected and removed by using the command **"Remove Route"**. This function simply deletes that specific MongoDB collection.

It is also possible to visualize the subscribed routes from **"Display Route".** This function recovers the collection for that specific route, stores it in hidden control son the server and from the client side, using Javascript, shows it on the map (LEAFLET).

The route monitoring function allows viewing, in real time, the user location and displaying the various alarms that are occurring in case of any deviation from the planned route. This functionality uses information supplied not only by the person with special needs Interface Prototype tracking but also by the alerts comming from the CEP.

3.4.3. Developed services for publishing Madrid UC into the RB

The different layers of the Mobility RB are designed to be loaded in real time, using control mechanisms based on observation of status changes. EMT own architecture needed to evolve itself to create subscribers of the collection subtype "things". More specifically, Windows servers have been migrated to Windows Server 2012 and the affected SQL database servers have been migrated to 2008 or 2012 versions, depending on the situation of each affected subsystem. The group of developed routines have been installed on the business servers as Windows services and have been directly connected to production systems.

Different services have been developed depending on the connected layers and feeders:

- Observers for “things” subtype layers
    - Observer of things from LINEBUSMAD, DRIVERSBUSMAD, BUSSTOPMAD: Loading is made directly from the EMT Madrid information systems, using real values and keeping changes through the SQL Server Change Tracking Service. It consists in a Windows service that detects changes in databases feeding the related layers:
        - Bus line routes and bus stops list: EMT ARCGIS Server [27]
        - Information of buses: EMT SAP [28]Maintenance system
        - General data of bus drivers: EMT HHRR SAP system.

    - Observer of things from APPLICATION, USERS: Loading is made by observing EMTIng database. This way, any subscribed app may allow registring, tracking and receiving PUSH notifications.

- Observers of event data layers related with the activity.
    - Observer of events from BUSSTOPMAD: Feeds in real time by a DDP observer connected to the RB SAE through the bus position layers.
    - Observer of events from PMVSTOPMAD: Feeds in real time by a DDP observer connected to the RB SAE through the activity and bus stop status layers.

- Specific observers for planning detection and use of routes by person with special needss.
    - To upload changes in route planning by person with special needs caregivers, a DDP observer subscribed to ROUTEPLAN.userplan has been developed.
    - To observe the bus movement within the segments of a certain bus route defined by a caregiver in the route planning functionality, there is a DDP connector observing BUSMADRID.eventpos. This allows the CEP to know the usual bus movements thorugh segments no matter if there are specials persons or not travelling on them.
    - Route observer for tracking of the person with special needs and monitoring through the monitoring website, using several DDP observers subscribed to ROUTEPLAN.userplan, BUSSTOPMAD.events and msgOut.

3.4.4. SP user interface prototype and SP route simulator

Although the final application is expected to be Android, for testing the deployed system at a prototype level, a Windows application has been built performing route simulations using a UI from the SP VE that manages, from the background, subprocesses communicating the different RB layers.

This application is used to set different contexts in which a trip can be run under different emulated conditions, building itineraries by both right and wrong ways, using different travel times for journeys on foot. It integrates into the Mobility RB directly through MongoDB, so its connection (currently) makes mandatory to hold a VPN tunnel with the EMT information systems, since the database is not accessible from Internet (only the Meteor server).

3.4.4.1 Overview of the SP VE app prototype

The form consists of two sections:

First part (as shown in Figure 17):

- User selection: The person with special needs is authenticated within the application, which allows knowing his/her planned active routes.
- Proposed route to be done: the SP selects what route plans to do.
- Selection of the route model (wrong or right); refresh time of user tracking.
- Stop error rate: % of stops within the trip in simulated mode, in which for each wrong stop randomly obtained, the user will leave the bus by mistake and then will return to the stop to continue the journey.
- Start Route: Start of the route simulator.

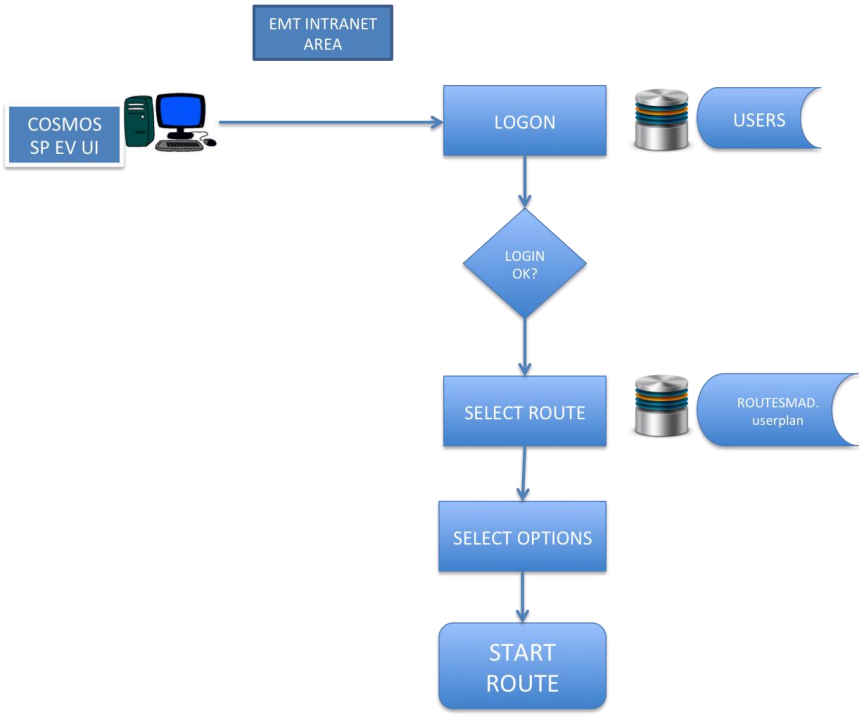


Figure 30. Logon sequence and route selection emulation tour

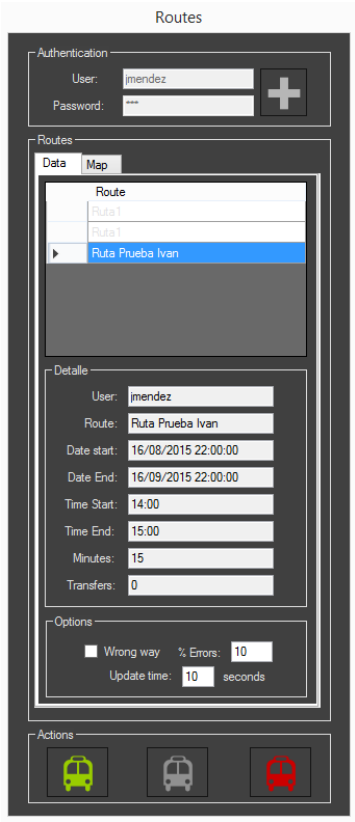


Figure 31. Main display of the person with special needs UI emulation

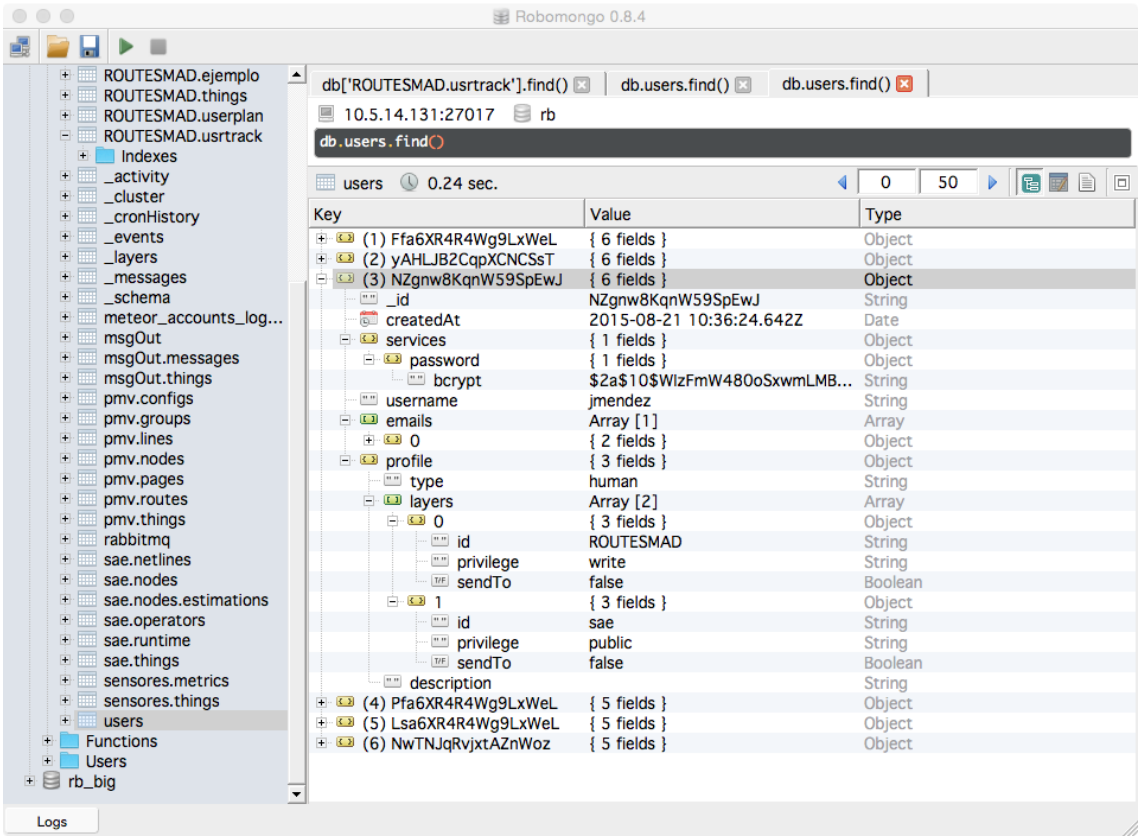


Figure 32. Scheme of user definition

Second part: User's route tracking.

In the second option of the screen, the route track emulator is activated. It consists of a map where the path of the user can be followed including the different points the user passes by.

The route can be ended at any momento and we can start a new tracking.

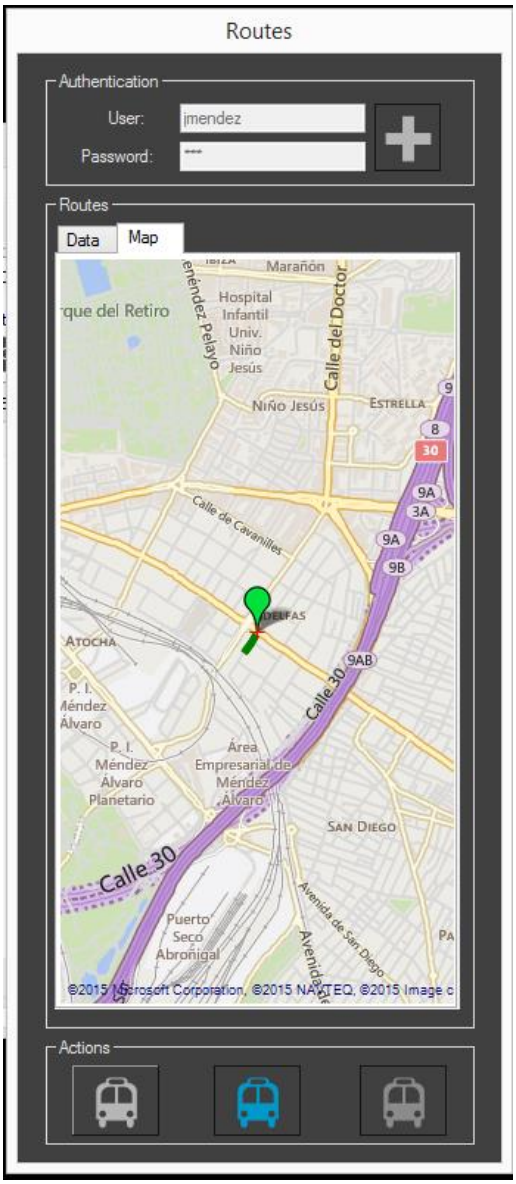


Figure 33. Appearance of the route emulator of the SP VE app prototype

3.4.4.2 Emulation process flow

The emulator has two models of operation, depending on the route segment that the user is performing

1. Walking tour: The system allows the variation in travel speed so that tracking can be established with varying speed. In the case of erroneous way, the system performs random stands off on route planning.
2. Bus tour: For a tour bus, the user approaches to the bus stop and the emulator asks to mybus.emtmadrid.es about the actual arrival time of the next bus on the specific bus line that the traveler must take and makes a real hold. Once the bus arrives at the bus stop, the SP gets on (gets linked to the bus) and begins its journey. If the wrong routes manager is activated, the user will randomly get off the bus at a certain bus stop and will get on again after waiting for the next bus. If the wrong routes manager is not activated, the user (SP) will make the planned route until the appropiate bus stop to begin the next path section. The bus tour is in real time as the user is virtual but really linked to a bus that is making a tour on the street at that time.

To get the arrival times of buses, the emulator uses a service of the EMT Opendata layer:

http://10.5.249.100/EMTServices/GEWSVGEO/Service.asmx

GetEstimacionStop
Checks the remaining time before the next bus arrives at the stop where the user is waiting (TimeLeftBus)

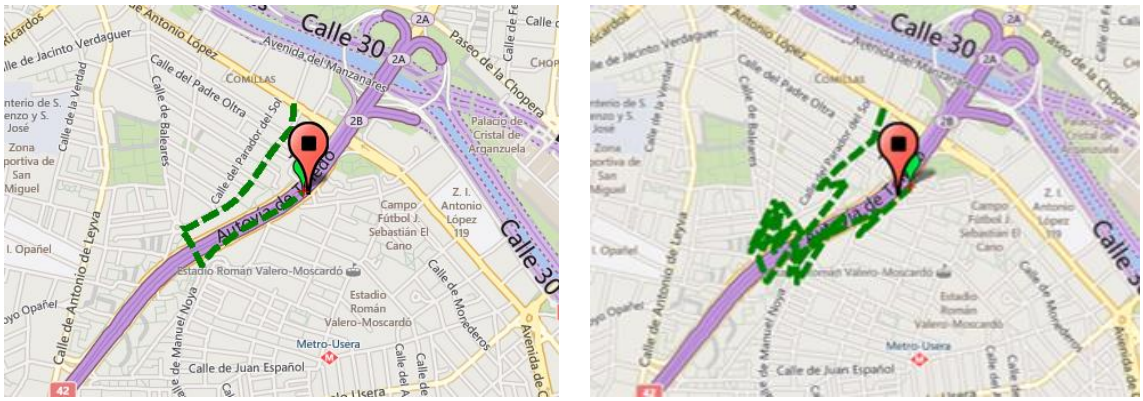


Figure 34. Tracking differences between a right route and a wrong route

3.4.4.3 Dial Symbols and information content

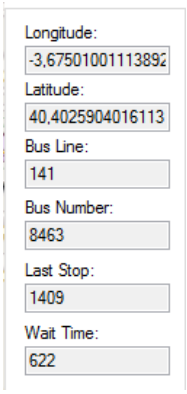
The app uses the following symbols when showing the tracking:

Walking user mark	
Bus user mark	
Bus stop mark	

The prototype shows the coordinates of the current position of the user. The time it takes to update can be configured for both walking and bus tours.

It also reports about the bus line, about the number of bus on which the SP is getting on or waiting for and about the last stop in which the SP has been.

In addition to that, if the SP is waiting for a bus, it shows the expected remaining time (in seconds) for the bus to arrive to the bus stop.



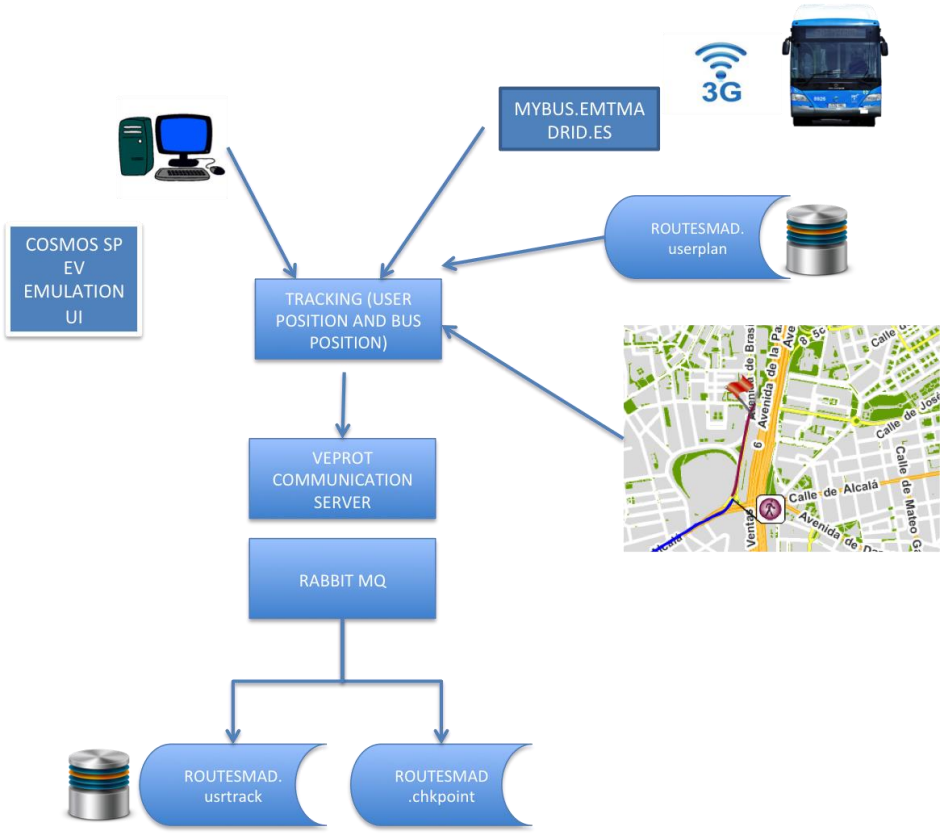


Figure 35. Update flow of route registring LAYERS from the SP UI

3.4.5. RB integration process into COSMOS

As discussed so far, all dependent processes from the Reactive Box have sufficient elements to observe changes in any system, taking advantage and the ability to create different adaptive entities (stored in collections) and grouped according to their nature with the purpose to relate them and to facilitate its subscription and observation.

Logically, as the RB architecture has been conceived as a IoT model under the requirements and specification of COSMOS architecture, the NodeRed+CEP subsystems can naturally use this technology, not only in the scope of Madrid use cases, but scaling it to multiple solutions .

What is described in the following points is the specific integration model for the prototype, although the logic NodeRed will allow subsequently establishing other logical subsystems just watching other RB LAYERS.

As indicated in previous documents, the suggested integration scheme within the COSMOS global model proposed for the UC is the following:

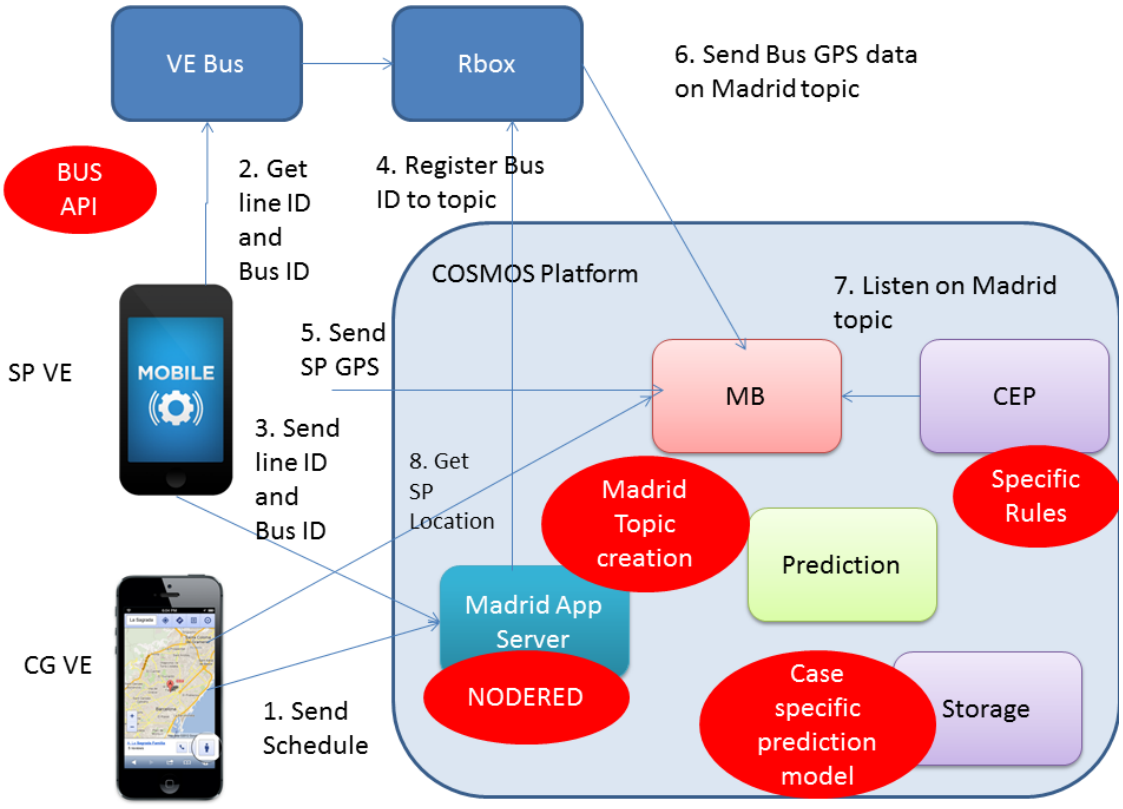


Figure 36. Madrid Assisted Mobility application runtime

3.4.5.1 Publication of data layers for supplying data from the UC to COSMOS

The process flow for the COSMOS platform will be defined as follows:

A NODERED process will be watching the route planning layer, detecting creations, changes or deletions of planning. To do this, it must control which plans are currently in place or will be in the future and ignore plannings that have already expired.

1. For each subscribed planning, it will receive an event (Caregiver Virtual Entity) that will serve to meet the scheme of the planned route. In this event, defined in the ROUTEMAD.userplan layer, the following information will be available:
   - Unique identifiers of each planning (user identifier, route identifier).
   - Date Range, type of day and days of the week in which plans to carry out monitoring of a SP VE.
   - Time interval in which the SP VE makes the journey.
   - Definition of walking paths (geographical start and end points and intermediate checkpoints of travel, if any).
   - Definition of bus rides (start and end bus stops and checkpoint bus stops).

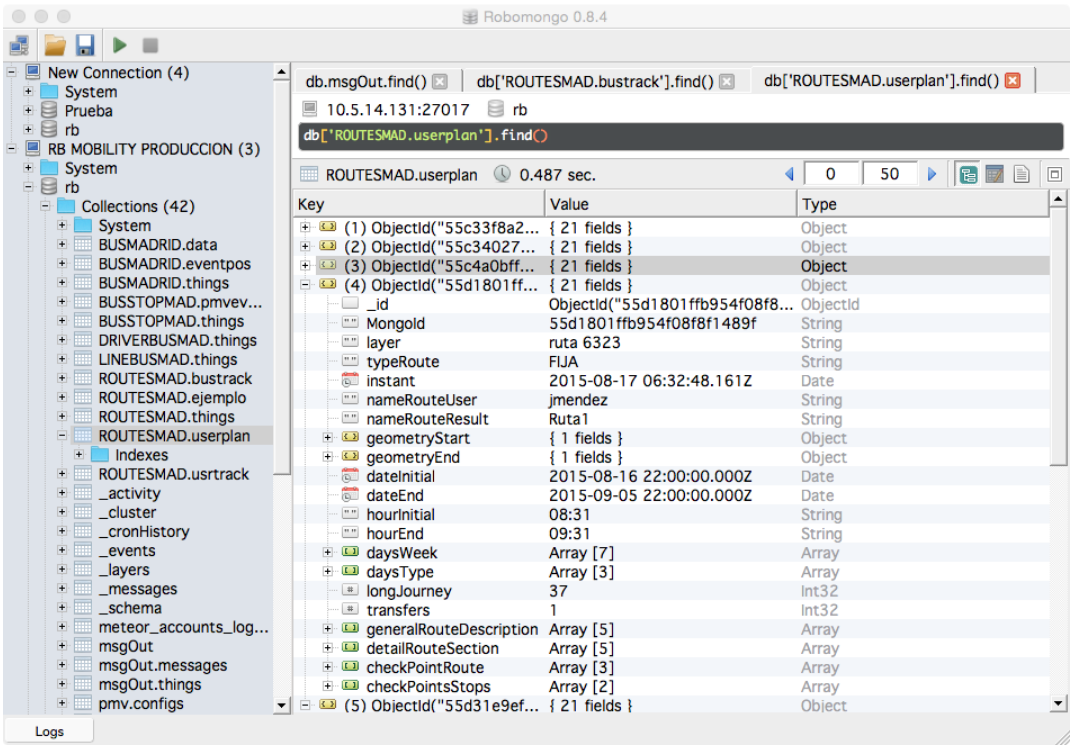


Figure 37. Route plan stored in MongoDB Layer

2. As indicated above, an internal observation process of EMT systems is in charge of providing constant information on the positions of the vehicles in the range of observation (dates, types of day, day of week, time interval) for the segments of each existing route planning. This information is constantly being generated, regardless of

whether or not there is a SP VE making the journey. This information is being generated in the ROUTEPLAN.bustrack layer and carries the unique identifiers of the associated planning route to which it belongs.

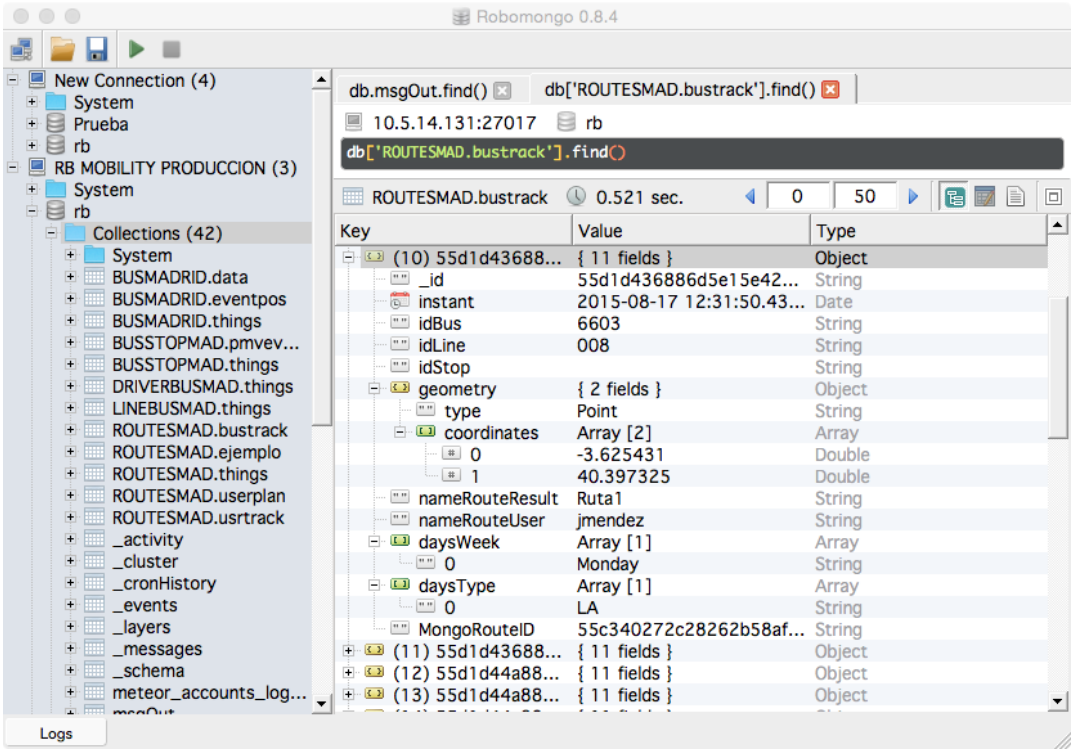


Figure 38. Data tracking of bus registered in MongoDB Layer

Once this information is consumed by NodeRed it is analyzed and stored by COSMOS performing computations through the CEP through which it obtains:

- Average times for each way trip on each observation group (type of day, day of week), creating patterns of behavior.
- List of events observable by other sources, for example, traffic intensity and fluctuations in relation to calculating the route segment.
- Quantifiable abnormalities, for example, events that repeat patterns of behavior in which predictable deviations occur under analytical models.

3. Observations about route starts, route ends and route checkpoints. In this collection the basic route information of the user session is stored (INIT, PASS and END) as essential route checkpoints for controlling purposes. There are as many PASS as checkpoint are defined on the planned route and as long as the user moves through that points.

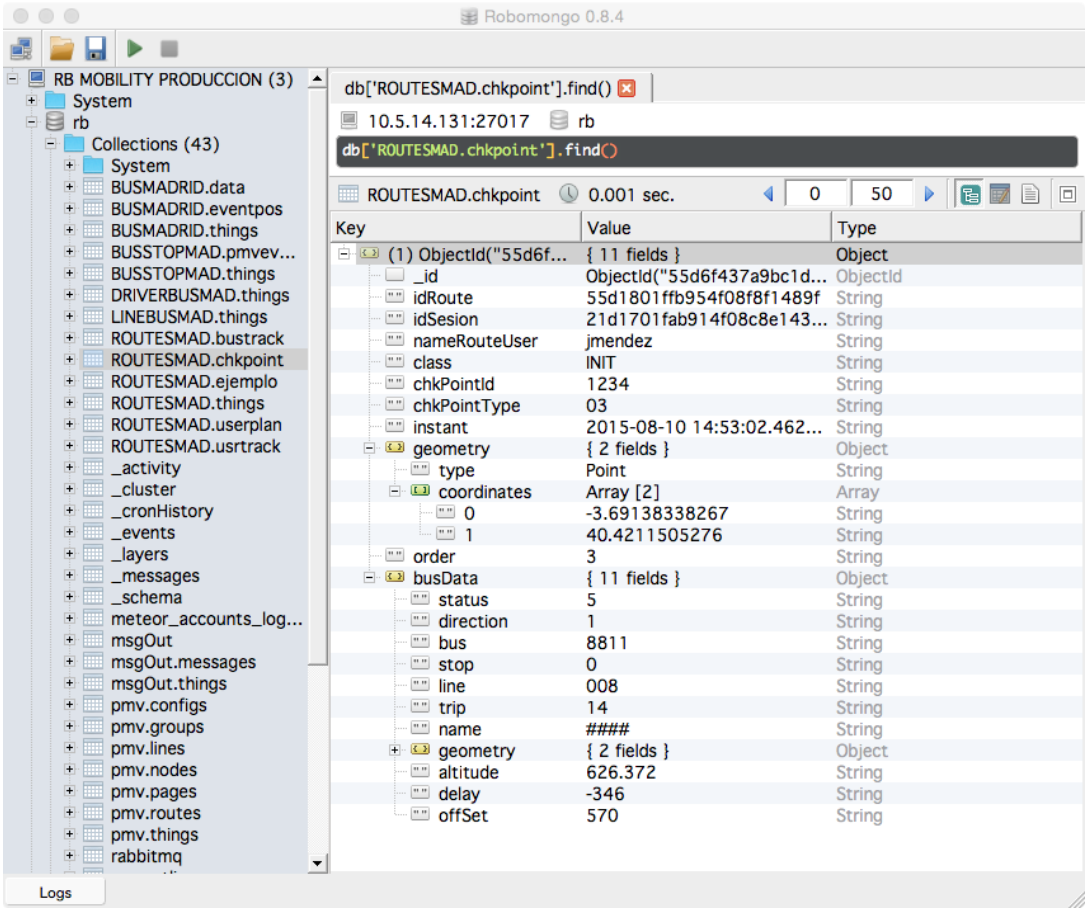


Figure 39. Pass thru data on a certain check point, in MongoDB Layer

When it gets an INIT event (beginning of a route by a SP VE) the CEP will make an initial calculation of the theoretical intermediate times passing through the checkpoints, in order to further establish the passing by deviations through them. To do this, it takes into account the theoretical time of each segment of the route and the location of the checkpoint along the route.

4. Observations of the SP VE route trips by subscribing to the ROUTEPLAN.usrtrack layer to detect the activity and movement of the SP VE.

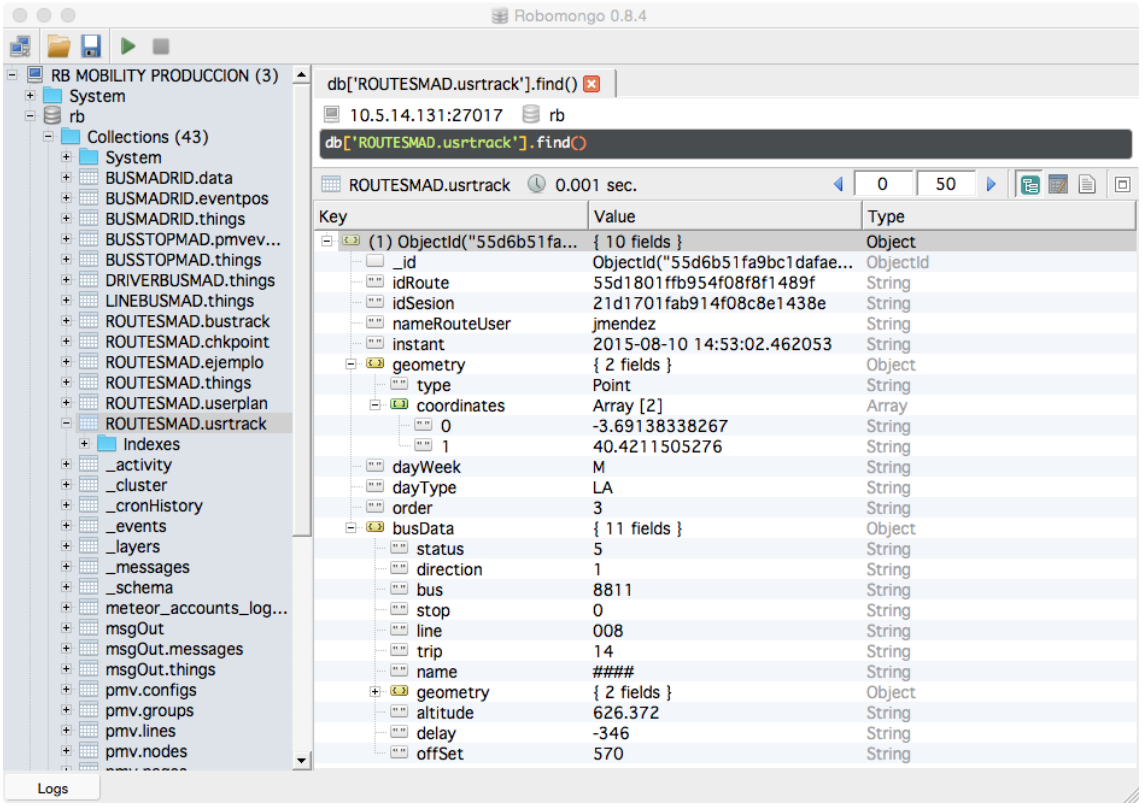


Figure 40. Data register of user track in MongoDB Layer

3.4.6. Charging COSMOS solutions and integration in Madrid Mobility

One of the most important tasks in all the functional flow, for both the prototype and the final solution, is the ability of COSMOS system to provide solutions to complex problems. These solutions are vital for the effective control of the exceptions that occur in the flow of normal activity when using any mean of transportation. Within Madrid scenario this is an essential element for the monitoring phase.

3.4.6.1 Integration working model

As discussed in the previous section, the ROUTESMAD layer with all its collections behaves like a VE able to provide routing paths, historical activity of buses through those routes, and tracks of the SP that run through them. All this information, through DDP observation models, are analyzed and recorded in the COSMOS system in order to provide knowledge related to "normality" and standardization of events. Thus, the system can know when an event ceases to be normal.

After obtaining the benchmarks and detected an "anomalous" event, the COSMOS system will incorporate an event within the MsgOut collection, reporting to the system. To do this, it will use VEProt as message scheme.

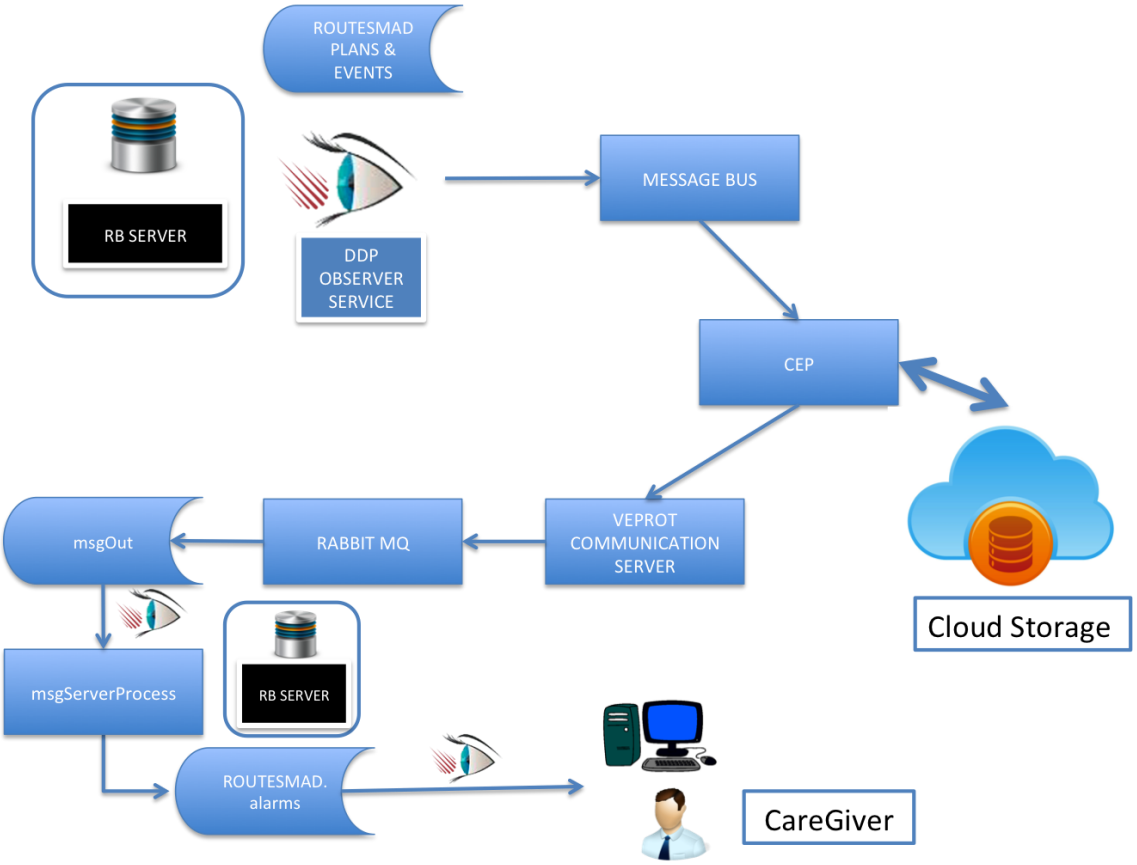


Figure 41. Bidirectional flow scheme between VEProt and the alarms management of COSMOS predictive model

3.4.6.2 Event list that COSMOS system will provide to Madrid UC and prototype cases

Although the final system model considers a wide variety of alarms and events that the CEP related processes can manage and shoot for the benefit and use of the Madrid mobility area, the prototype, for practical reasons, has been restricted to only a few events that largely allow to demonstrate the benefits and usefulness of the system. Among the global list of events, only those reported as "prototype" will be considered within the scope of this phase:

event_id	Description
cosm_mad_uc_ev_1	Abnormal traffic density that may affect the bus route (prototype).
cosm_mad_uc_ev_2	The SP gets off the bus before scheduled (prototype)
cosm_mad_uc_ev_3	The SP continues beyond the scheduled bus stop in which it should gett off the bus (prototype)
cosm_mad_uc_ev_4	The SP does not reach the checkpoint points during the walking tour (prototype)
cosm_mad_uc_ev_5	The SP alters the planned route on foot and gets diverted from the planned route (prototype)
cosm_mad_uc_ev_6	Demonstrations, public events and other disturbances in the normal city traffic flow.
cosm_mad_uc_ev_7	Deviations from the usual route.
cosm_mad_uc_ev_8	The vehicle in which the SP travels fails or gets stuck into a traffic jam.
cosm_mad_uc_ev_9	The arrival of the bus at the bus stop where the SP is waiting is significantly delayed

Table 5. List of events in Madrid UC

3.4.7. Functional scheme and summary of the technical interactions of Madrid UC

The process flow through which the prototype is modeling Madrid UC is summarized as follows. The individual components have already been explained in this document.

First phase: The SP responsable access to the management portal and manages the SP routes to monitor. This includes the date range, the time range and the type of day. The SP responsable also registers the control checkpoints along the route that the user must perform. The system generates the route. The system begins to publish records of all passing buses in real time for each road sections affecting the planning. COSMOS system observes and collects the plannings and buses passing by records to obtain data analysis.

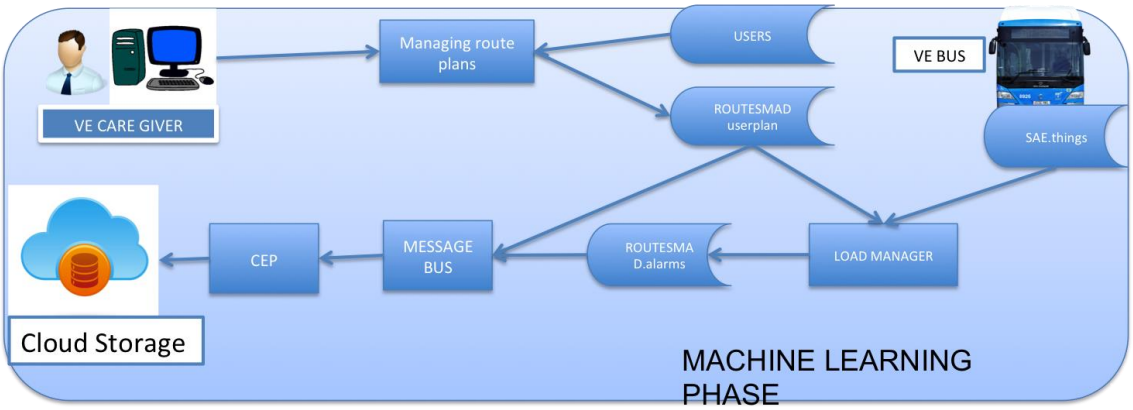


Figure 42. Machine Learning Phase

Second phase: The route tracking process begins with the selection of a route by the SP in order to start using it. From now on two type of entities are registered:

1. Steps on foot and by bus every 30 seconds from the user's position.
2. Steps for the control points (checkpoints)

SP activity and record data are used by the CEP to further knowledge and experience of real use of the route, plus for route tracking in real time.

In parallel, it is being done the route monitoring and SP tracking by the caregiver.

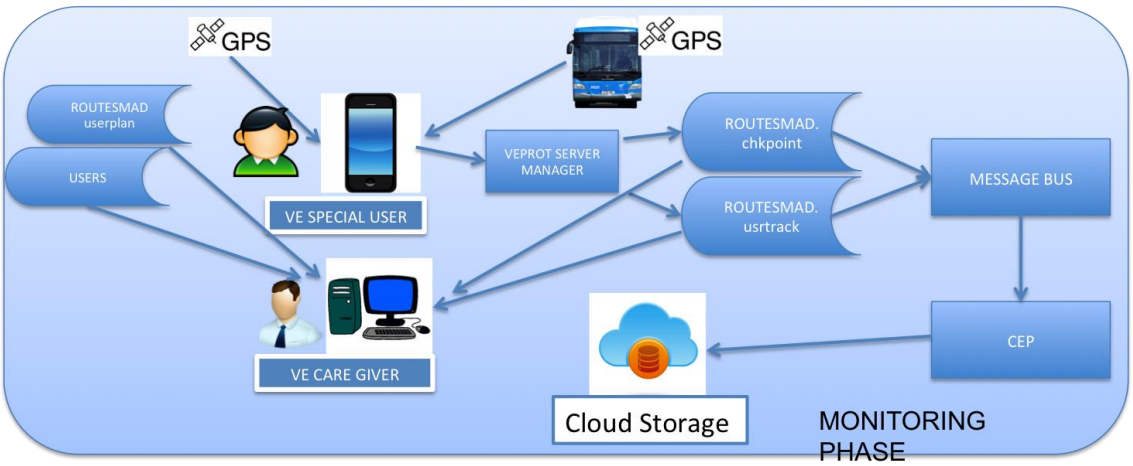


Figure 43. Monitoring phase of a SP

Third phase: From the knowledge and experience accumulated through the CEP, and the use of various analysis elements stored in the Cloud, COSMOS is able to anticipate and know what anomalies will occur or are occurring within the trip plan or in previous phases.

The anomalies are inserted into the system by transferring events through the RB message queues. The messages are transferred in real time, both to the SP application (if the SP is on route) and to the caregiver monitoring website.

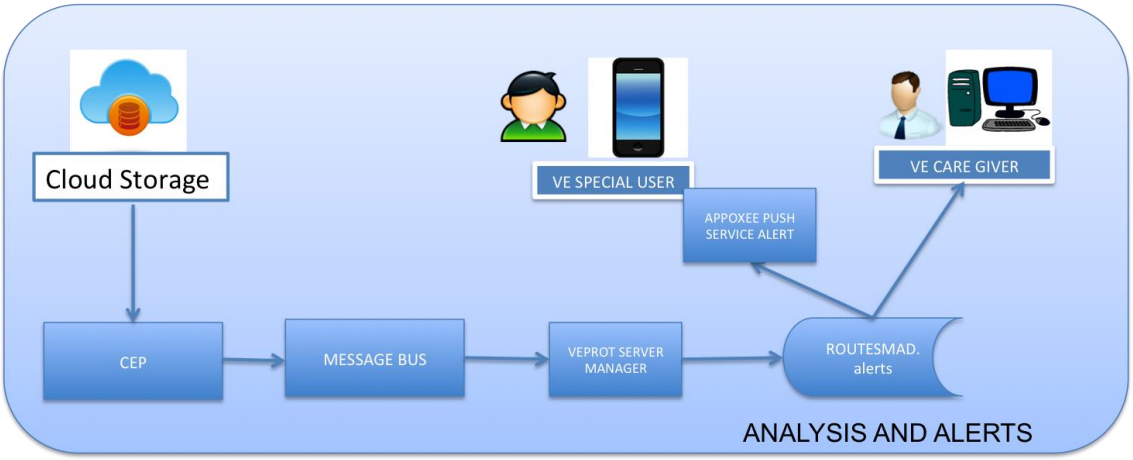


Figure 44. Analysis and Smart Events alerts

4. Final tasks and executive phase

4.1. Testing plan

Once the development of the prototype is completed, it requires a series of tests to verify not only the COSMOS solutions model but also each and every one of the developed functionalities. Given the complexity of these, as requires specific circumstances of mobility and tours are analyzed and recorded, having the support of MyBus platform has been essential, as it allows to be virtually on a bus making a real trip, while keeping working in the on-site development.

Finally, it is essential during the testing phase, to assess the viability of each and every one of the solutions and alerts that the Madrid UC proposes, besides avoiding simultaneously, any restriction of the model lowering it to a laboratory solution without any scope or real systems environments. Furthermore, the purpose of COSMOS is to test adaptive systems which intelligent design allows learning in this area. On this regard, it is meaningless to define a too simple scheme of the solution, as this could never prove the fairness of the system. In this sense, it can be highly interesting to know how the whole solution behaves under other areas (eg traffic light priority, or even communicating Camden infrastructure with Madrid Reactive Box and verify that the solution is adapted without major modifications).

Furthermore, the evolutionary plan being carried out encompasses further developments in the prototype, such as the ability to launch specific requirements of data under control of events expiration, trigger delay, locking and waiting mechanisms and iteration or recurrence of events. All this is currently being analyzed while performing the testing phase, which will not end with COSMOS but will be incorporated into the Madrid Mobility Laboratory, together with all the components necessary to enable reuse in other project areas or as evolutions of COSMOS solution.

With respect to the prototype phase, the test area will cover, originally, the proposed landmarks:

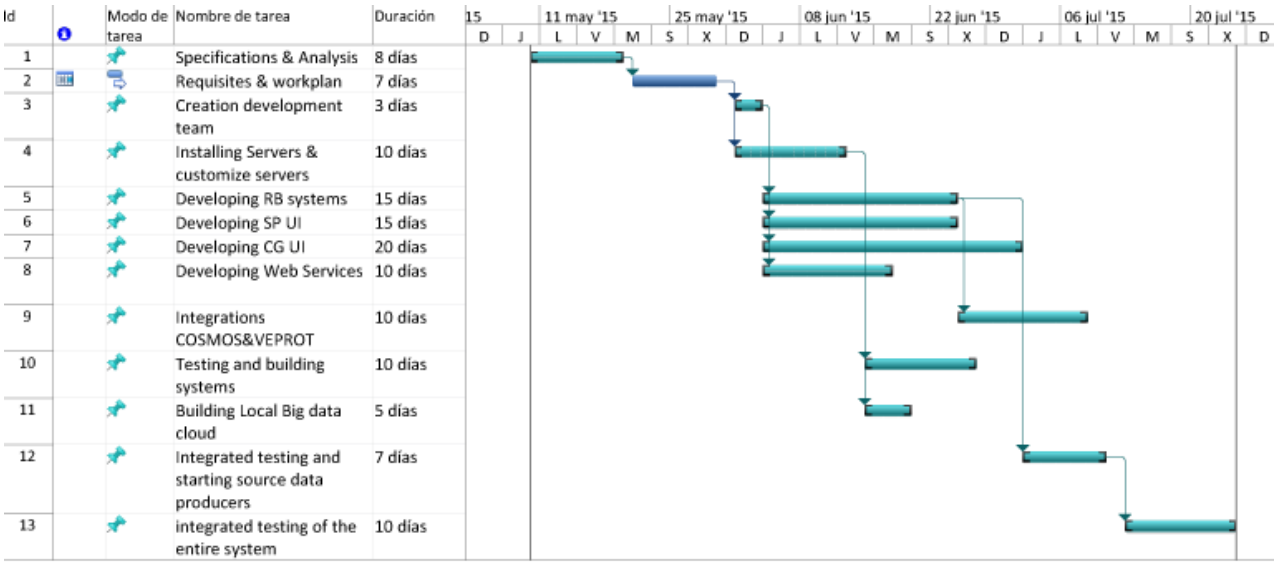
1. Abnormal traffic density that may affect the bus route
2. The SP gets off the bus before scheduled
3. The SP continues beyond the scheduled bus stop in which it should gett off the bus
4. The SP does not reach the checkpoint points during the walking tour
5. The SP alters the planned route on foot and gets diverted from the planned route

Once the test is over, it will be examined whether there is sufficient margin of time or opportunity in the field of COSMOS to integrate datasets and feeds from external systems to correlate, by CEP, the rest of the expected situations, in order to make a more extensive evaluation of the system:

- Demonstrations, public events and other disturbances in the normal city traffic flow.
- Deviations from the usual route.
- The vehicle in which the SP travels fails or gets stuck into a traffic jam.
- The arrival of the bus at the bus stop where the SP is waiting is significantly delayed.

4.2. Timetable

The development of the prototype has demanded the implementation of a multidisciplinary project team, affecting a significant number of members of the EMT Systems technical department that have had to make, either specific collaborations or dedicate themselves partially to achieve the milestones. This fact was reflected in the initial schedule of tasks which is shown below (Figure 30):



Id	Modo de tarea	Nombre de tarea	Duración
1		Specifications & Analysis	8 días
2		Requisites & workplan	7 días
3		Creation development team	3 días
4		Installing Servers & customize servers	10 días
5		Developing RB systems	15 días
6		Developing SP UI	15 días
7		Developing CG UI	20 días
8		Developing Web Services	10 días
9		Integrations COSMOS&VEPROT	10 días
10		Testing and building systems	10 días
11		Building Local Big data cloud	5 días
12		Integrated testing and starting source data producers	7 días
13		integrated testing of the entire system	10 días

Figure 45. Timetable for the development of the Madrid UC prototype

4.3. Conclusions

4.3.1. Disciplines and learning

IoT technological environments require new models of knowledge whose field jumps over the analysis and individual research processes requiring multi-disciplinary collaboration processes in which technicians must accept and take over challenges in areas of information systems whose edges and borders are even very vague. The making of a prototype is an excellent way to apply theoretical knowledge and skip the conventional rules. We know where we want to go, but we only see the road when we started to explore it.

The making of the prototype has demanded major changes within the Madrid mobility technology infrastructure, creating a new technological ecosystem based on Meteor-MongoDB, RabbitMQ and also based in management and definition of semantic data models, all that to guide the public transport and mobility architecture towards IoT. While much remains to go, the road is already mapped. Now it has a dynamic, abstract, adaptive and intelligent model. The prototype will demonstrate whether COSMOS is a key part of the proposed model for its capacity for analysis and prediction of events, something of high value to the VEProt system.

4.3.2. Adaptability of the prototype to the final project

Although the prototype design has endeavored to take into account aspects of reusing components for the final system, however there are always shortcuts in development that require more sophisticated designs and implementations, all in accordance with the COSMOS final system. Among the various components that will require a redesign later on, we can find the following:

- Caregiver Web Platform: it is currently developed within the EMT Intranet and connects to the RB MongoDB database in native mode, which means that access to it for all kinds of tests may need a VPN connection. In the final system, it shall be published on the Internet, which will force developing a set of connectors and services that provide access to the RB to login and for obtaining and processing route planning layers.
- SP application: The prototype is designed with Framework 4.5 [29](Visual Studio 2013) and requires a Windows computer to operate. Moreover, for route selection to be performed requires a direct connection to MongoDB (so needs a VPN connection). The final system must have an Android development and DDP access to subscription mechanisms of RB.

The remaining components, with a relative evolution, could be considered as final products, being most of them server functionalities in different technologies.

4.3.3. Smartphone user interface for a person with special needs

4.3.3.1 Synergies and end user engagement

As already specified in previous deliverables, during the specification phase it arose the opportunity of having a close collaboration with the Madrid Regional Transport Authority (CRTM) in the development of a Mobility solution oriented to groups of users with special needs, as the CRTM is currently involved at an H2020 project called INLIFE (http://www.inlife-project.eu/) about accessible mobility for people with mild cognitive impairment, which requires of a motor-assisted navigation system.

In addition to that, as a result of Y2 review, EMT and Madrid City Council started to think how to involve end user and how to collect their feedback in order to design from scratch the functionalities to be used by end users.

Therefore, conversations started between CRTM and EMT with the overall idea to foster synergies among both projects, having the first meeting on April 25th, 2016, with external stakeholders, including:

- EMT (technology, consultancy and CSR departments)
- CRTM (accesibility department)
- ATOS
- Federación Autismo Madrid (Madrid Autism Federation)
- CERMI (Spanish Committee of Representatives of People with Disabilities)
- UPM (Madrid Politechnic University)

In order to get the aforementioned feedback, a survey was circulated in April 26th to all associations, in order to spread it among their associates and members. The survey is avaibale at the following link:

https://docs.google.com/forms/d/1PhkejCY3C7IB0Pnml1HMOfpVzOBEkkFNa9jqy73Lyxk/edit?usp=sharing

Feedback was collected until June 16th, 2016, getting 65 responses, and it has been taken into account in the following aspects:

- In the design of the app user interface:
  - Color contrast
  - Font size
  - Icon shapes
- Fine tunning of app/web portal functionalities:
  - App functionalities:
    - Simplificating message sequencies
    - Locking the possibility of ending the session by the user
  - Caregiver web portal, including new functionalities:
    - To check the previous routes being done
    - To monitor several users at the same time

4.3.3.2 Smartphone user interface development

Therefore, the engine developed within the COSMOS project for the Madrid SmartMobility use cases became an opportunity to provide a continuity and consolidate its use. the COSMOS team involved at the development of the Madrid SmartMobility use cases offered to INLIFE project managers and stakeholders the possibility to collaborate in the development of the user application under the global requirements of monitoring them. That is, not just thinking of the group described within INLIFE project, but also in the other user groups that were already the COSMOS target groups (children, elderly, people with autism or even tourists).

As a result of this collaboration, INLIFE has currently an Android user interface which navigation and routes engine is provided by COSMOS. In an analogous way, COSMOS has a user interface to consolidate and market the solution, provided by INLIFE.

The INLIFE application for route monitoring is a development that allows a user with special needs to know a route in advance. This route is previously defined in a Caregiver web portal developed as one of the COSMOS Madrid use cases. The application, in a native way and once started, gets connected into the Reactive Box, performing a startup session. Once established, the system returns the planned routes, represented and stored by the VE ROUTESMAD.userplan.

When the user starts a route, the INLIFE application notifies to COSMOS system, thorugh the ReactiveBox, an Start route event using the information represented by VE ROUTESPLAN.chkpoint. This implies that the Caregiver is notified and the system begins to monitor the user's position through the system.

Every 30 seconds, the application notifies the activity of the route and the device status through the RB, including the position of the device, on an automatic way.

In addition, along the entire route the app will indicate the milestones/checkpoints that should be passing through and travel directions. When the next milestone is to get on a bus it will indicate which bus stop to go and how long it will take the bus to arrive.

This information keeps being sent to COSMOS by the RB, which uses it to monitor and foresee circumstances that may abnormally affect the route.

The app automatically connects to the on-board Wifi of the bus and gets the information about the bus stops on the way. The device itself will notify to the user, in advance, when to get off. Any anomaly, such as traffic problems, deviations from the route, descent or early departure from the planned route, will be automatically notified to the Caregiver. The Caregiver can also send messages to the user using the webportal, at any time.

The MyRoutes app is completely accessible, and both contrasting colors, buttons and icons size, and even the model of messages that appear, has been designed to be simple, as it is aimed at people with special needs (and of course according with their feedback).

Figure 46. Smartph

4.3.4. Scalability possibilities to other projects

Considering the Madrid UC conceived architecture and its adaptability to the needs of COSMOS, it is possible to make new additions to justify the strong development and bring added value. In this regard, as described above, the design of the prototype solution has always sought to take into account multiple and polymorphic solutions when creating and developing it. Among other features, these are some of the feasable alternative solutions:

- Traffic light priority system. Conceptually, the solution is valid in many of its components.
- Navigation and guidance of people with wider cognitive disabilities. Virtually the entire system is valid because its philosophy has a great similarity with the scheme proposed in COSMOS Madrid UC.

4.3.5. Dissemination of experience

4.3.5.1 Publication of documentation and source code in public repositories. Opensource

There is a clear commitment by the agencies involved in Madrid, for publishing an important part of the solution in opensource mode. Among other content, it will be published in GIT:

- PYTHON source code of the VEProt datagram manager and connectivity model with RB.
- C# source code for the DDP connection to the layers of the RB.
- Specification of the semantic schemes and data elements that support all the Madrid Mobility model of Madrid in the context of this project.

4.3.5.2 Publication of connector http://rbmobility.emtmadrid.es in public and open mode

The RB system for public use is already published. However, it requires an API KEY for each client system to ensure an adequate access. This is due because the information published in the various layers may contain personal or confidential data, or may be only accessible to certain processes.

This means that, although access to the platform is free and certain layers are permanently accessible, it requires a personal identifier for each connector that is granted depending on the characteristics and needs of each connected subsystem, as it occurs with COSMOS, whose identity allows it full access to all ROUTESMAD layers with information on the activity and monitoring of people with special needs.

4.3.5.3 Dissemination through http://openlabmobility.emtmadrid.es

Currently, the part of the project involved in the dissemination of ideas and initiatives around IoT for intelligent transport (which is covered by Madrid Mobility Lab), is under specification. For such dissemination a web portal that aims to serve as a hub of ideas and initiatives arising within or outside the COSMOS field is being designed, no matter if these ideas and initiatives belong to business or educational areas, and regardless of their relationship with the public sector.

This fact seeks to create knowledge, experience and new ideas that create innovation and helps transforming the city, at least in the field of transport and mobility. To do this, the broadcast center of knowledge and experience that will publicly display the contents and projects is based on standardized and easy adaptation support tools. It is installed and under construction in http://openmobilitylab.emtmadrid.es

4.3.5.4 Wordpress

CMS Wordpress has been installed as the manager of content and publications related to the laboratory tasks. The aim of the site will be to provide help and knowledge for as many IoT based ideas as possible in the field of transport and mobility.

In conclusion, the COSMOS project will be widely disseminated through this site, providing detailed contents about the implementation and development of the project. The Madrid UC developed prototype will be announced and published in the web site with links to the corresponding Wiki.

4.3.5.5 Wiki

To support the extensive information and documentation about terms, references and definitions that can provide useful and evolutionary development to the site and experience sharing, the website will contain a Wiki to provide references and data. In this Wiki there will be functional and technical specific documentation of Madrid UC prototype.

4.3.5.6 GIT.

Although, initially, the portal has been endowed with a repository of control and management of documents and source code, it has finally opted for the use in open mode of github.com due to the fact of the degree of dissemination, universality and accessibility of this popular site. Currently, part of the definition of the data model associated with the Madrid UC and other RB related subsystems can be found in https://github.com/madridopenlabmobility

4.3.5.7 Development of the MobilityLabs portal.

One of the challenging issues during the development of COSMOS ecosystem has been the ability to present, from a technical point of view, the different elements that houses the infrastructure, and especially how to make possible to other developers the use of the elements that have been designed.

Once this objective was achieved during Y2, the next step was to provide servers and services infrastructure to let MobilityLabs have its own identity. This development has been carried out, exposed and self-defined by the following portal:

https://mobilitylabs.emtmadrid.es

Unfortunately, at this moment it is only in spanish, having the following aspect:

Figure 47. MobilityLabs website interface

Figure 48. MobilityLabs screenshot including COSMOS info

The portal contains definitions of all the subsystems that exist on it, with operational details. For example, in the case of COSMOS, we can see how the COSMOS-CEP is presented in detail:

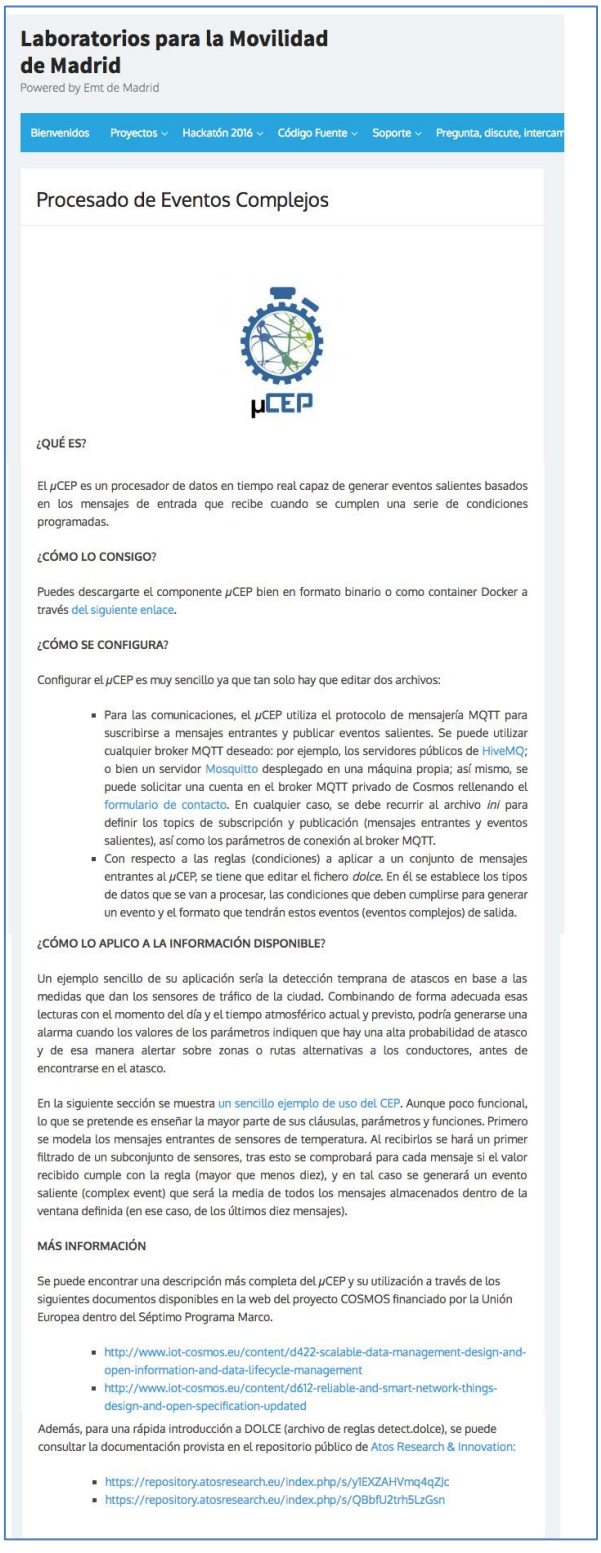
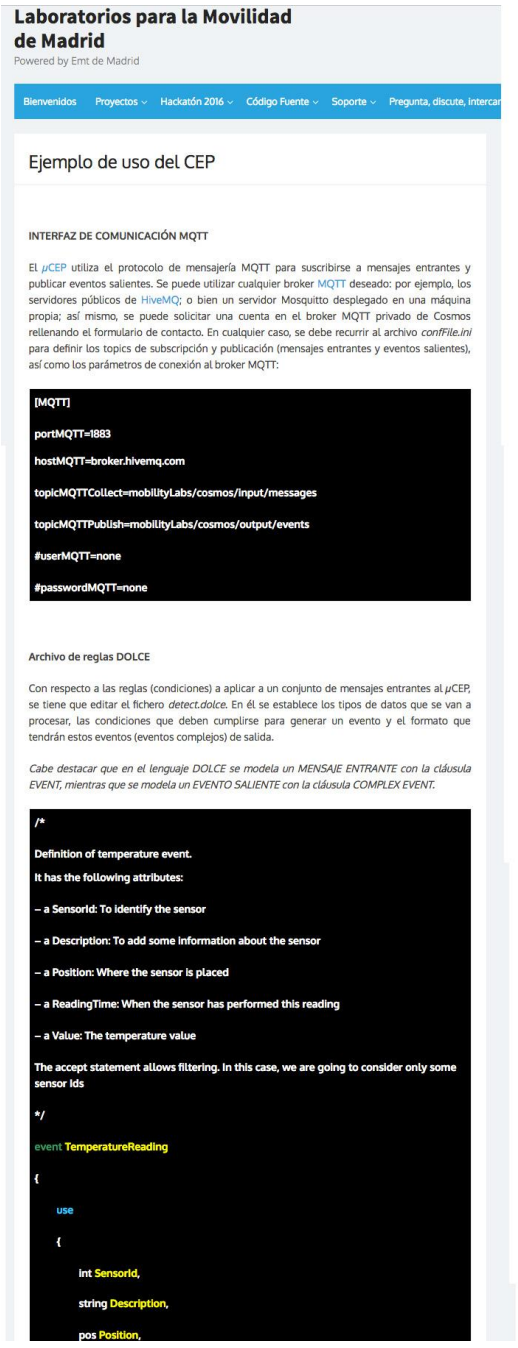


Figure 49. MobilityLabs screenshot including CEP info (1)



There are also sections devoted to source code, available to be downloaded and used with real working examples which are capable of delivering information to some Opendata VE and operating models to interoperate and observe Madrid traffic and public transport information. In the following example, we can see the page published with source code examples using the COSMOS-CEP

Figure 50. MobilityLabs screenshot including CEP info (2)

Annex A. Components Involved

➢ RB platform

○ Meteor Environment

Name	Version	OS
Node.js[30]	0.10.4	Linux
Meteor	1.1.03	Linux
Java Runtime Environment	1.7	Linux/Windows

Table 6. Software requirements for the RB platform

○ No-SQL data server

Name	Version	OS
MongoDB	3.0	Linux
Ubuntu[31]	13.10	Linux

Table 7. Software requirements for Data Mapping and Storage

➢ Windows System Servers

Name	Version	OS
Sql Server	2008/2012	Windows
Internet Information Server	7	Windows

Table 8. Software requirements for the EMT http servers

➢ VEProt Interfaces

Name	Version	OS
WCF Services	Framework 4.5	Windows
RabbitMQ server	3.5.4	Linux
Python	2.7	Linux/Windows

Table 9. Software requirements for VEProt Integration Services

➢ Services for integration RB with Internal EMT Systems

Name	Version	OS
Windows Services	Framework 4.5	Windows
DDP Protocol	1.0	Windows
MongoDB Connectors	2.0	Windows

Table 10. Software requirements for Integration and Interchange of Smart Events

➢ SP prototype

○ Person User Interface

Name	Version	OS
Windows forms	Framework 4.5	Windows
Windows client	7 / 8 / 10	Windows
MongoDB Connectors	2.0	Windows

Table 11. Software requirements for SP UI

➢ Care Giver user interface

Name	Version	OS
Windows ASPX	4.0.4	Linux/Windows
Javascript	7.0.54	Linux/Windows
Leaflet	0.7.3	Linux/Windows
OpenStreetMap[32]		

Table 12. Software requirements for the prototype of Caregiver UI and Emulation

➢ User Interface for People with special needs

Name	Version	OS
Java	5.0 or higuer	Android

Table 13. Software requirements for the User Interface of Person with Special Needs

References

[i] Distributed Data Protocol and Meteor specifications reference are in https://www.meteor.com

[ii] Data definitiones is published in https://github.com/madridopenlabmobility/MOBILITY-MADRID-virtual-entities web site.

[3] JSON-LD specification is published in http://json-ld.org/. Also some references and supports are from https://en.wikipedia.org/wiki/Semantic_Web, http://schema.org/, http://geojson.org/ and "Nosotros los Constructores de la Web Semántica", written by Luis Criado (http://criado.info/luis/NosotrosLcdlWS.htm)

[4] MongoDB specifications are in https://www.mongodb.org/

[5] More information about SAE system and other features of EMT systems are in http://showroom.emtmadrid.es/ing/index.html

[6] https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[7] https://www.python.org/, https://pypi.python.org/pypi/py and github.com

[8] https://technet.microsoft.com/en-us/library/ms189826(v=sql.90).aspx

[9] https://www.javascript.com/resources, https://en.wikipedia.org/wiki/JavaScript

[10] https://en.wikipedia.org/wiki/Microsoft_SQL_Server , https://www.microsoft.com/es-es/server-cloud/products/windows-server-2012-r2/Resources.aspx

[11] Internet Information Services official web page https://www.iis.net/

[12] http://httpd.apache.org/, http://www.apache.org/, https://en.wikipedia.org/wiki/Apache_HTTP_Server

[13] http://opendata.emtmadrid.es

[14] https://www.microsoft.com/net , https://en.wikipedia.org/wiki/.NET_Framework

[15] http://www.w3.org/TR/REC-xml/

[16] https://en.wikipedia.org/wiki/UTF-8

[17] https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system

[18] https://www.android.com/, http://www.apple.com/ios/

[19] https://en.wikipedia.org/wiki/Software_development_kit
[20] https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx

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

[22] https://en.wikipedia.org/wiki/ASP.NET

[23] http://leafletjs.com/
[24] https://jquery.com/, https://en.wikipedia.org/wiki/JQuery

[25] http://nosql-database.org/, https://en.wikipedia.org/wiki/NoSQL

[26] http://bsonspec.org/

[27] http://www.esri.com/software/arcgis/arcgisserver

[28] http://news.sap.com/business-suite-erp/

[29] https://en.wikipedia.org/wiki/.NET_Framework_version_history

[30] https://nodejs.org/

[31] https://en.wikipedia.org/wiki/Ubuntu_(operating_system)

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