eVACUATE

FP7-313161

A holistic, scenario-independent, situation-awareness and guidance system for sustaining the Active Evacuation Route for large crowds

eVACUATE User Manual

Deliverable Identifier: D.12.4
Delivery Date: 31 May 2017
Classification: **Public**
Editor(s): Alessandra Rossetti - Vitrociset
Document version: 1.0 - 2017

Contract Start Date: April 1st, 2013
Duration: 50 months
Project coordinator: EXODUS S.A. (Greece)
Partners: EXO (GR), IT INNOVATION (UK), ICCS (GR), HKV (NL), TEL (GR), TEK (ES), AIA (GR), VITRO (IT), CDI (UK), INDRA (ES), KUL (BE), DXT (FR), POLITO (IT), STX-FR (FR), TUD (DE), TUC (DE), ASRS (ES), METB (ES), TIM (IT)

Project co-funded by the European Commission under the 7th Framework Programme

SEVENTH FRAMEWORK PROGRAMME

Document Control Page

Title	eVACUATE User Manual	
Editors	Name	Partner
	Alessandra Rossetti	VITRO
	Sandro Viola	VITRO
Contributors	Name	Partner
	All technical Partners	
Peer Reviewers	Name	Partner
	Marco Mancini	TIM
	Pedro Garibi	INDRA
	Rodríguez Vázquez, Jorge Juan	INDRA
Format	Text - Ms Word	
Language	en-UK	
Work-Package	WP12	
Deliverable number	D.12.4	
Due Date of Delivery	31/05/2017	
Actual Date of Delivery	19/06/2017	
Dissemination Level	Restricted	
Rights	eVACUATE Consortium	
Audience	☒ public ☐ restricted ☐ internal	
Date	19/06/2017	
Revision	1	
Version	1.0	
Edited by	Alessandra Rossetti	
Status	☐ draft ☒ Consortium reviewed ☒ WP leader accepted ☒ Project coordinator accepted	

REVISION HISTORY

Version	Date	Description and comments	Edited by
1.0	05/06/2017	TOC	Alessandra Rossetti
2.0	19/06/2017	First Draft	Alessandra Rossetti
2.1	19/06/2017	Reorganization of contents	Alessandra Rossetti

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 SCOPE

This document has been realized with the scope to collect from the sub-systems of eVACUATE platform all the information needed to install, configure and use each component of the system and, of course, to make them all working together.

Each component has been realized with a different technology so different rules must be applied for its deploying and integration.

The components are divided following their membership to the Work Packages of the Project:

WP	User Manual section
WP3	Crowd Behaviour Detection and Recognition in Crisis Situation
WP4	Evacuation Routes
WP5	3D Interactive Common Operational Picture
WP6	Communication Layer Apps for smartphone
WP7	Smart Spaces
WP8	Decision Making and Optimal “Situation-aware”
WP9	SAES (Situation Awareness and Evacuation System) Framework

In different chapters are described the guides for the end-users and the guides for configuration and administration of the components.

2 END USER MANUAL

This chapter describes, for components with GUI, all the steps needed to use them.

2.1 3D INTERACTIVE COMMON OPERATIONAL PICTURE

The COP is an intuitive system to navigate into 3D venue mock-ups, gather data from dynamic sources and display it in an legible way in real-time. This increases the situational awareness of decision makers which now have a complete picture of the ongoing venue situation. The decision maker can display and filter data, alter the simulation initial conditions to prepare for potential issues, plot his decisions into the 3D map to create his action plan and then share it with other connected users. This enables every involved organisations to share a common ground for crisis management. The COP also enables decision makers to start the evacuation process, triggering every other eVACUATE connected components, like the AER display in the dynamic exit signs.

2.1.1 General User Interface

The eVACUATE default general interface is presented in the figure below.

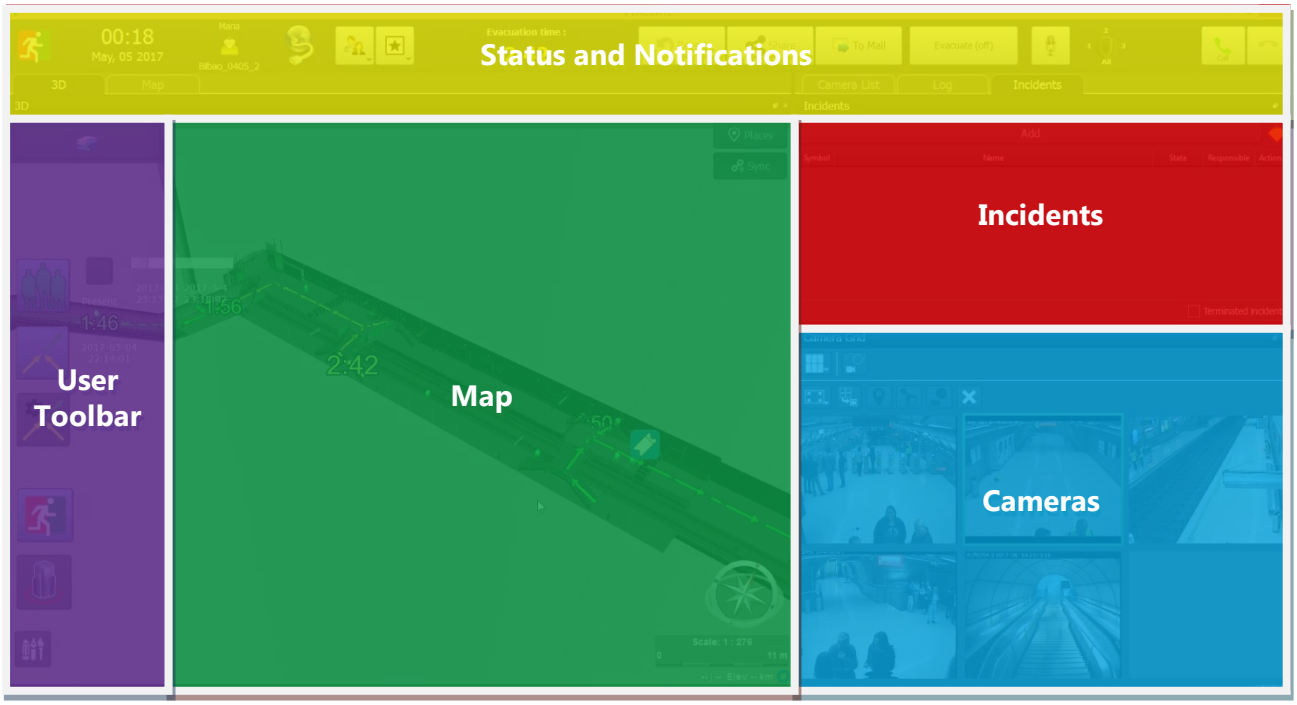


Figure 1: The COP default graphical user interface

Each part of the interface will be described hereunder.

2.1.2 Map

The eVACUATE 3D Map component enables interactive 2D/3D maps visualization and manipulation. Moreover, the eVACUATE Map component allows one to draw information on maps that are synchronized across multiple users. **Error! Reference source not found.** shows a 2D and 3D view of the eVACUATE Map component.

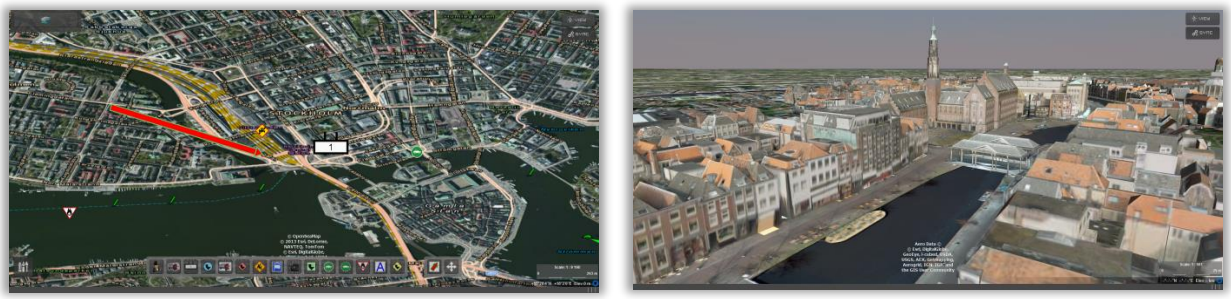


Figure 2: Left: 2D maps view of an European city. Right: its 3D representation

The eVACUATE Map component enables one to define the environment by importing most commonly used Geographical Information Systems datasets, such as roads, satellites imagery, plane imagery, terrains, buildings and vegetation.

2.1.2.1 Map manipulation

Mouse and touch maps manipulations are summarized in the figures below.

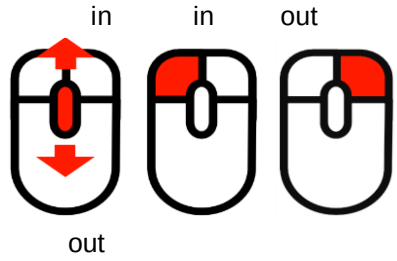
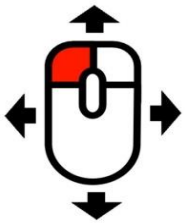
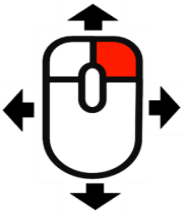
Zoom in/out	Translation	Rotation (3D Maps only)
		
		

Figure 3: Mouse usage

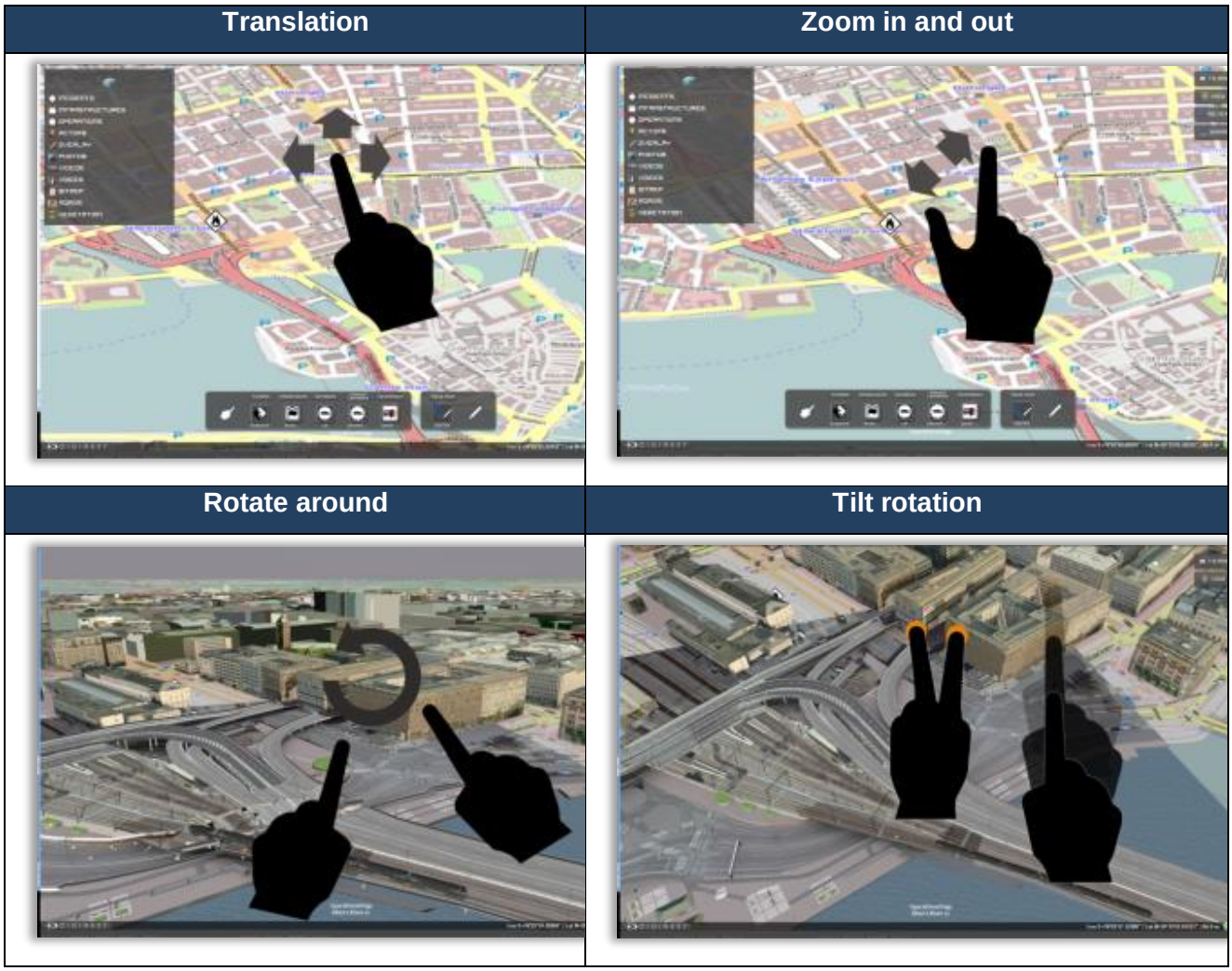


Figure 4: Maps tactile manipulation

2.1.2.2 Map edition

The Maps edition feature enables drawing symbols, lines, curves, texts, arrows and incidents on top of the map. The figure above shows the two user interfaces that are used to enable the Map edition feature. These features are described hereunder.

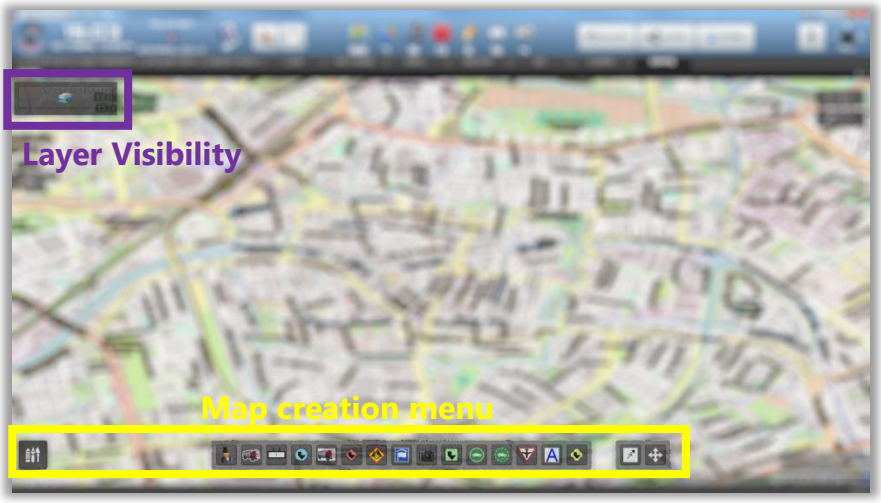


Figure 5: The user interfaces used to edit the eVACUATE Map component

2.1.2.2.1 *Map creation*

The Map creation menu is used to create information on the Map component. This menu is shown in the figure below. The Map creation menu is composed of two parts: Symbols and Annotations.

Figure 6: The Map creation menu, composed of the Symbols (left) and Annotations (right) menu

To show or hide the Map creation menu:

Press the Layer button to show or hide the Map Creation Menu.

2.1.2.2.2 *Symbols*

This user interface is composed of several stacks depending of your eVACUATE configuration, which display the last symbol used by user. These stacks correspond to the identified categories defined in the eVACUATE configuration settings. A common configuration includes :

1. Incidents;
2. Infrastructures;
3. Operations;
4. Planned Operations;

5. Risks;
6. Text annotations.

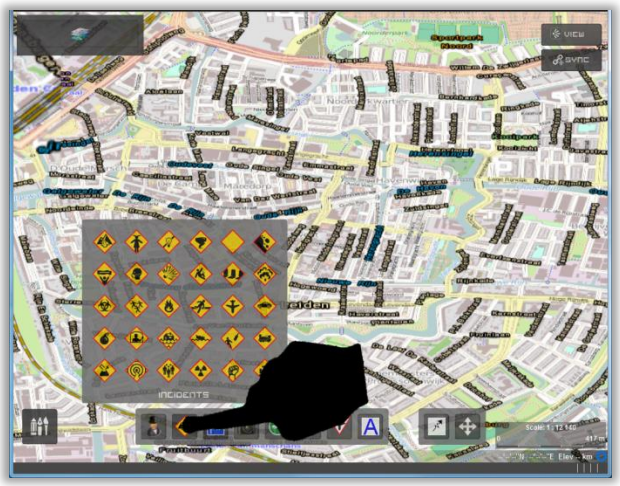
The symbol set can be adapted to the end users requirements. As an example, the integration of United Nations Office for the Coordination of Humanitarian Affairs icon set is demonstrated in the figure below

Figure 7: United Nations Office for the Coordination of Humanitarian Affairs symbols integration

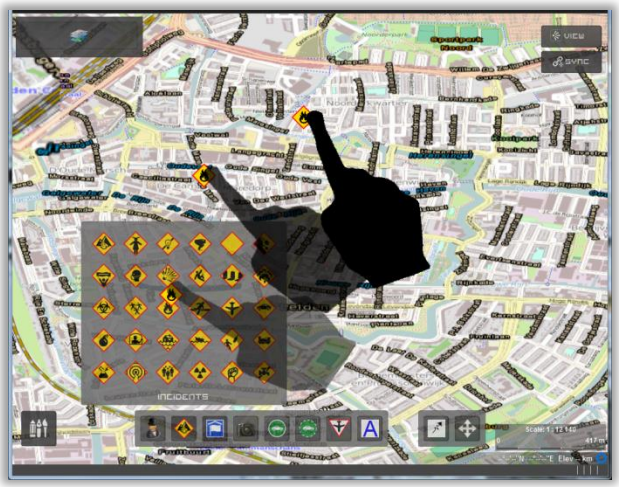
To create a symbol on the map:

1. The top of the stack symbol corresponds to your needs. Drag and drop the symbol from the menu to the desired location.

2. You are searching for a symbol. Select the category of your choice by tapping or clicking. A menu will open.



3. Drag and drop the symbol from the menu to the desired location.

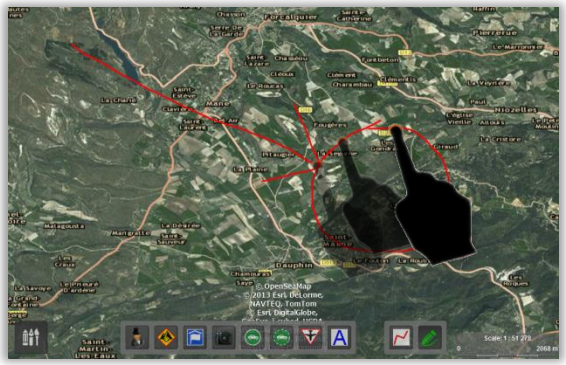


2.1.2.2.3 Annotations

Use this user interface to draw lines and polygons on the eVACUATE Map component. The stack displays the last item used.

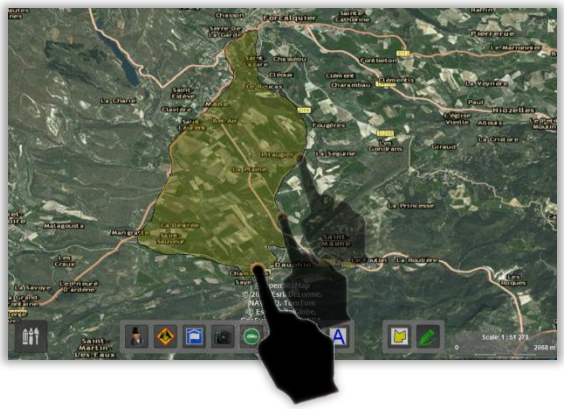
To draw annotations on the Map:

1. The top of the stack annotation corresponds to your needs. Draw on the map.



2. The top of the stack annotation does not correspond to your needs. Select the annotation category with a simple click or tap. A menu will open. Select the item of your choice.

3. Draw on the map.



2.1.2.3 Maps Modification

Use this User Interface to modify an existing symbol or annotation.

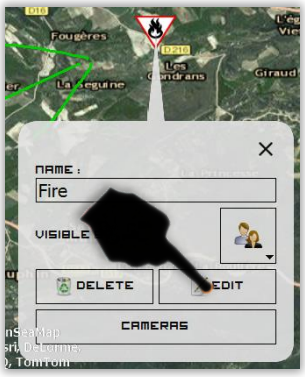
To delete an existing symbol or annotation on the Map:

Select the item to edit with a simple click or tap. A window opens. Press DELETE.

To translate an existing symbol on the Map:

1. Select the item to edit with a simple click or tap. A window opens. Press EDIT.

   Mouse shortcut: Right click on the symbol.

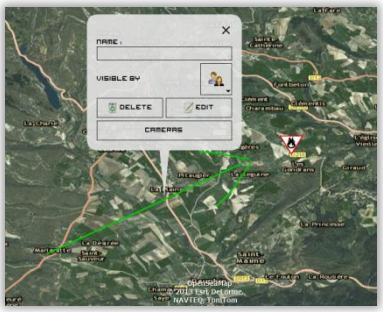


2. Drag the blue control circle to the desired location.

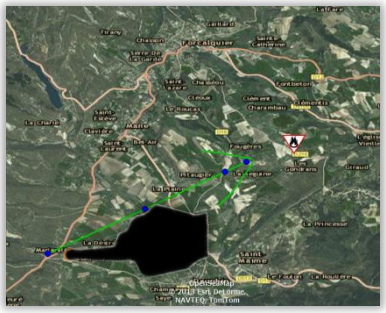
To edit an annotation on the Map:

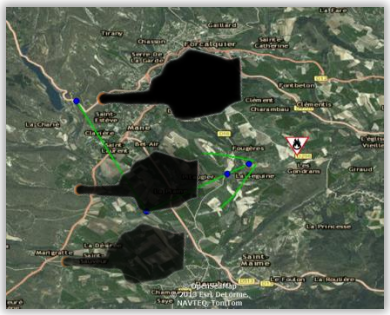
1. Select the item to edit with a simple click or tap. A window opens. Press EDIT.

   Mouse shortcut: Right click on the annotation.



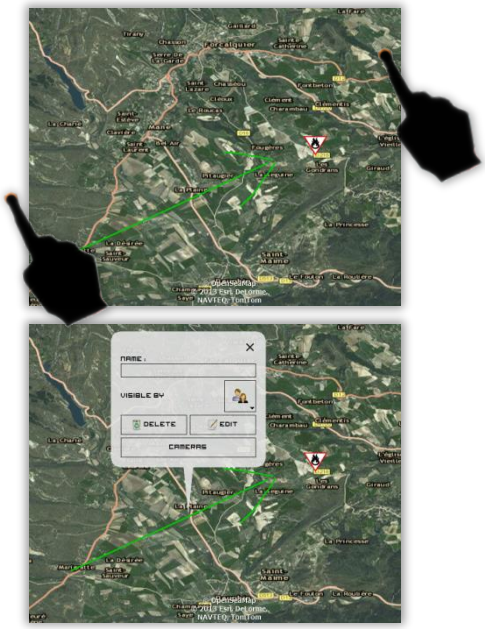
2. Drag the annotation control point you want to modify.





To edit a symbol or annotation text name:

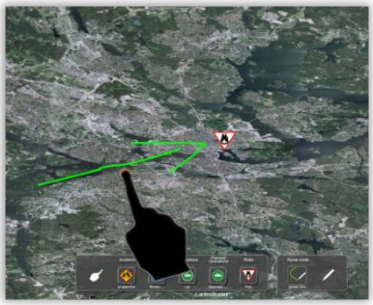
1. Select the item to edit with a simple click or tap. A window opens. Select the name text edit.



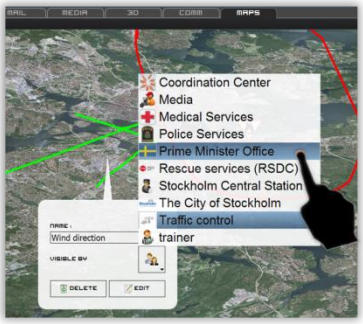
2. Type information with the physical or virtual keyboard.

To edit the group layer visibility of a symbol or annotation:

1. Select the item to edit with a simple click or tap. A window opens. Press on the group layer visibility icon.



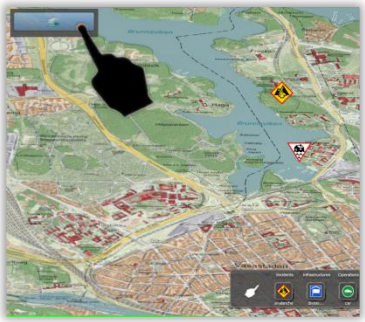
2. Select the new group layer that will see afterwards this information.



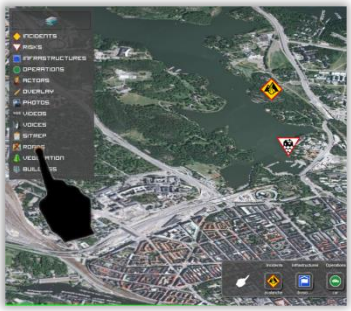
2.1.2.4 Layer Visibility

To display or hide a layer:

1. Press the Layer button to open the Layer Visibility Menu. A window opens.



2. Hide or display the layer by pressing on the desired layer name. A red cross over the layer icon indicates that the layer is hidden, otherwise the layer is displayed.

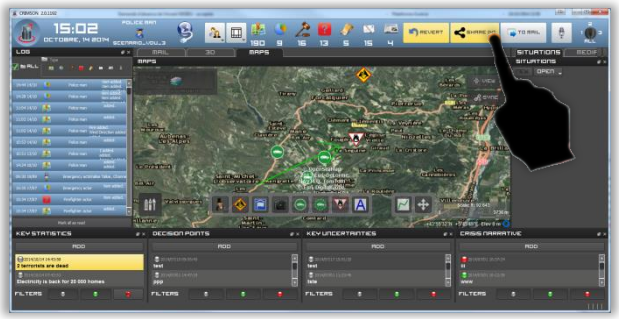


2.1.3 Share and Revert

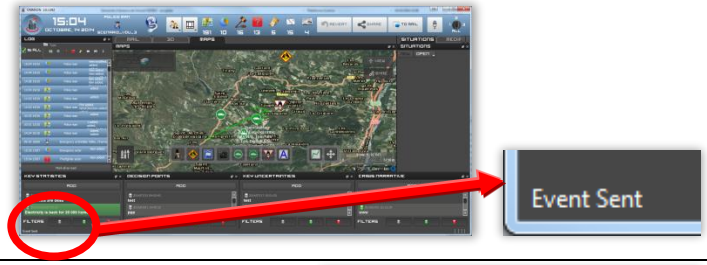
Every modification on the map can be reverted or shared by the Revert and Share buttons of the top bar. The count of modifications is between brackets.

To share the modification on the Map:

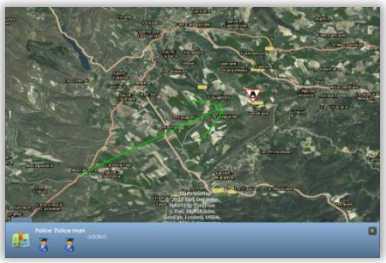
1. Press the Share button.



2. A network status notification is displayed in the bottom left of the screen.

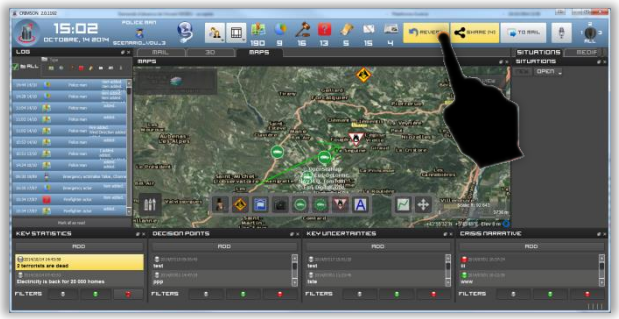


3. A notification indicates that the information has been successfully shared.

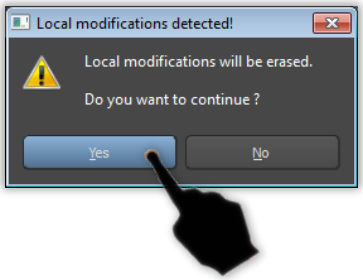


To revert all unshared modification on the Map:

1. Press the Revert button.



2. A confirmation is required. Press Yes to erase or No to keep your unshared modifications.



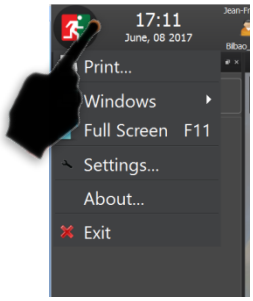
2.1.4 Status and Notifications

2.1.4.1 Status Bar

The Status Bar indicates global information related to the crisis and trainee/trainer connection. The status bar is located at the top and cannot be moved. A screenshot of the status bar with its components is displayed in **Error! Reference source not found.**.

Figure 8: The status bar, composed of 9 components, displays global information and controls

1. The eVACUATE button allows users to show a general menu.



a. Press the Print action to simply print the content of the screen by pressing a single button.

b. Press the Windows action to display the list of windows. Hide/open a window by pressing the desired window name.

c. Press the Settings action to configure the application.

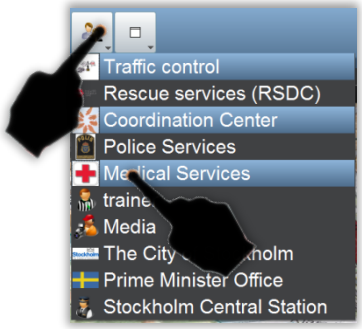
d. Press the About action to see license and plugins information.

e. Press the Exit action to exit the application.

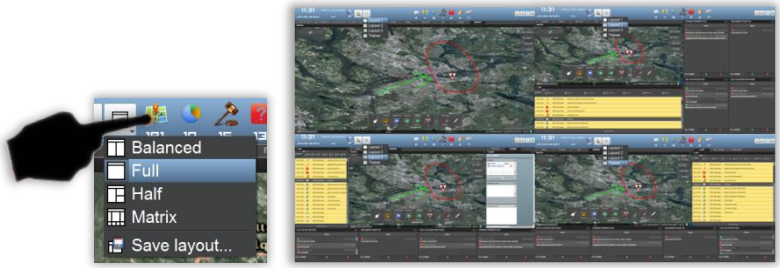
2. Current date and time information.

3. Database, login and network connectivity. In case of a network error, a red prohibition sign covers the network connectivity symbol.

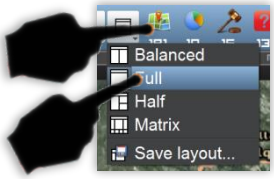
4. Press the Group Layer Filtering icon. Within the combo-box, select or deselect crisis group information layers to respectively display or hide information shared within this group. By default, information from all the groups is visible.



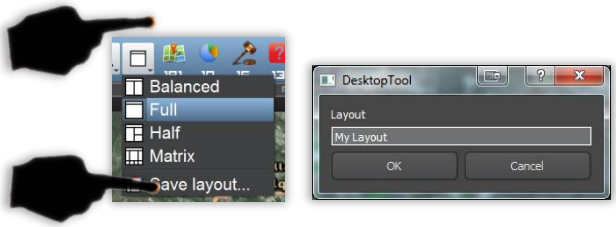
5. Layout can be changed and saved. From top to bottom, left to right: Layout Full, Half, Matrix and Balanced. The layout is not shared.



a. To change the layout, press the Layout icon, then select the preferred layout.



b. All windows can be moved/scaled at will. To save a custom layout, press the layout icon, then press the Save layout action. Enter the new layout name in the text prompt.



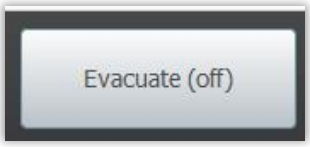
6. The current evacuation time provided by the system given the current building conditions and crowd prediction parameters.

7. Revert and Share buttons.

8. Press the "To Mail" button to prepare an e-mail with:
   - a screenshot of the 3D view as files attachment;
   - information from the Event Log.

9. The Evacuate button.

   Pressing this button will trigger the evacuation process.



2.1.4.2 Notifications

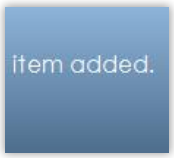
Notifications inform the user of new elements shared inside the network, as displayed in **Error! Reference source not found.**. The Notification "toasts" from the bottom of the screen, for 5 seconds. A short sound is played.

Figure 9: The Notification bar details

1. The type of shared information

2. Information on the Sender. The icons represent the crisis group and the user respectively.

3. Modification type.



4. Press the close button to close the notification. The notification fades out automatically after 5 seconds. Clicking anywhere on the notification will open the appropriate window and focus on the information.

2.1.5 Filtering the Crisis Groups Layers

The Filtering of Crisis Groups Layers enables one to focus on a specific layer of information, or to combine crisis information from other groups. Consider each crisis group as a layer of information, as represented in the figure below. The police layer of information (blue) is combined with the fire fighters layer of information (red). The filtering functionality enables one to display only the police or the fire-fighters layer, or both.

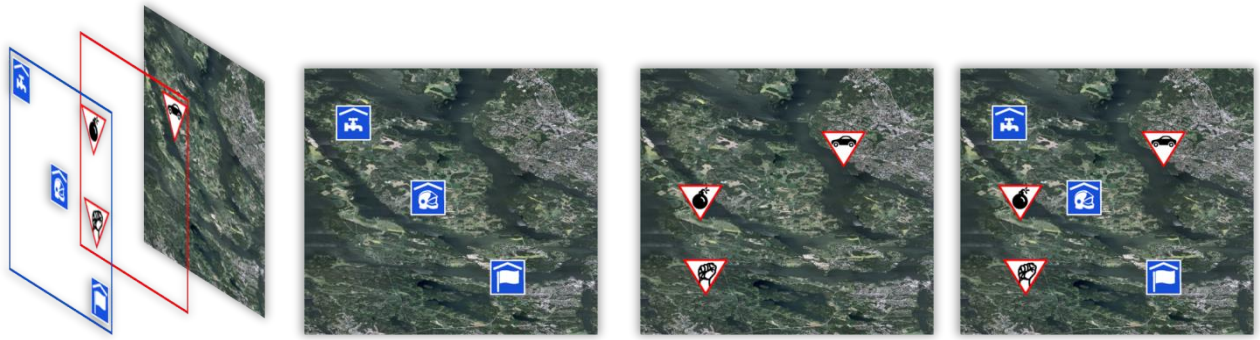
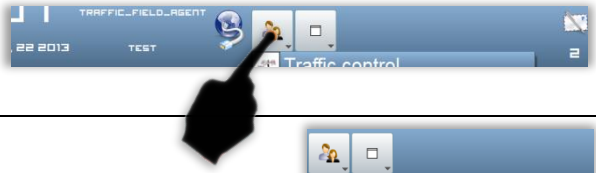


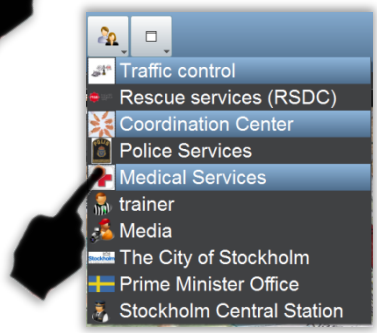
Figure 10: The police layer of information (blue) is combined with the fire fighters layer of information (red). The filtering functionality enables one to display only the police or the fire-fighters layer, or both

Filtering Crisis Groups Layers:

1. Press the Group Layer Filtering icon from the status bar. A combo-box opens.



2. Within the combo-box, select or deselect crisis group information layers to respectively display or hide information shared within this group. By default, information from all the groups is visible.



2.1.6 Event Log

The event log component's goal is to display and review all the events that have being logged within a session.

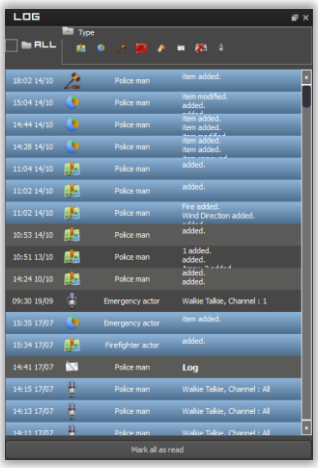
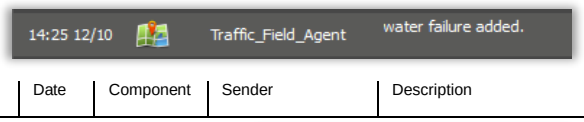


Figure 11: The Event Log displays all log events. The filter at the top enables one to display only events of a specific component

To select an event in the Event Log:

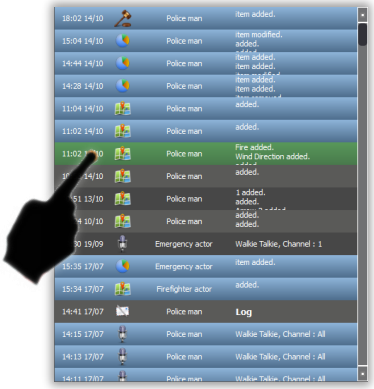
1. The log is composed of events. Each event contains date, type, sender and description information.



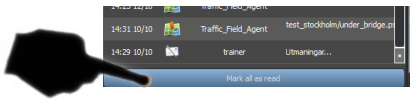
2. Look for an event in the list. The most recent items are at the top of the log. Scroll the list with two fingers.

You can also scroll with the mouse wheel.

3. Unread events are highlighted in blue. Press an event to read it. Pressing a Maps event will automatically show the maps items location.

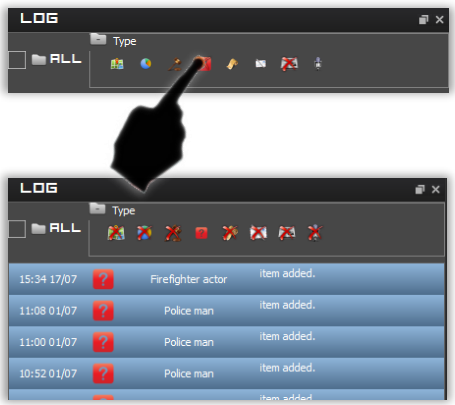


4. To read all events at once, press the "Mark all as read" button at the bottom of the Event Log Window.



To filter information in the Event Log:

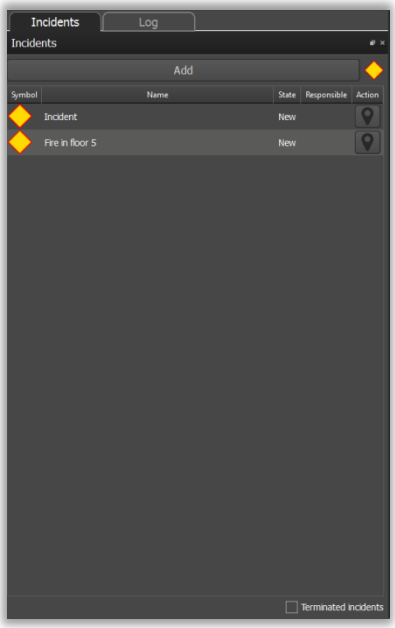
1. At the top of the Event Log window, check or uncheck the checkbox next to a component type to respectively display/hide the desired component type of information in the log. Press on the "All" checkbox shortcut to display all events at once.



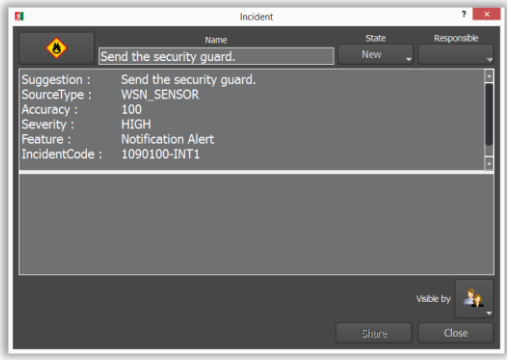
2.1.7 Incidents

The incident module enables the user to view at every incidents that currently happens inside the venue. Every incident can be either system or user generated. Whenever an incident is reported by the system or created by the user, it is both listed in the Incident list and on the Map (if the incident is geolocalised).

1. Unassigned incidents are blinking, meaning that the incident is still waiting for a user to take an action.



2. A user can create an incident by either dropping an incident icon directly on the map, or clicking on the Add button on top of the incident list. The user is then displayed a pop-up window enabling him to describe the incident.



3. A user can modify the state or the content of an incident by clicking on the incident on the list or on the Map. The same window as the creation pop-up is displayed for incident information editing.

2.1.8 Crowd management

The user of the COP is able to retrieve crowd information from the system and display it directly in the 3D representation of the building. The user of the COP is able to display live crowd densities, predicted crowd densities, active evacuation routes and can modify the blockage in the route to change the their computations.

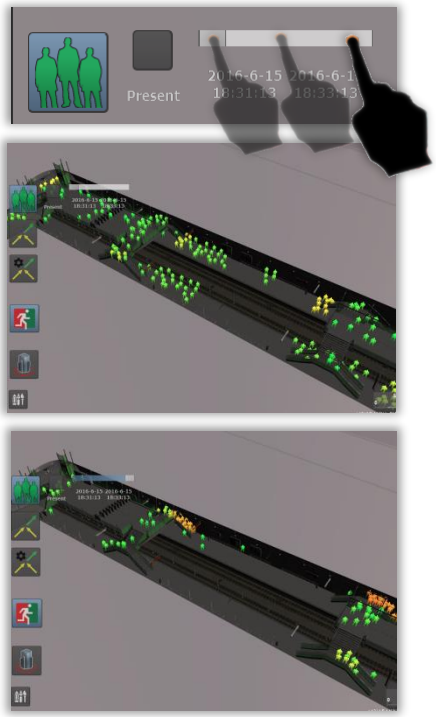
1. The crowd management menu. From top to bottom :
    a. Density display
    b. AER display
    c. AER modification

2. When pressed, the density display button displays a new menu.

    The first button switches the displayed density from the current situation to the one computed from the eVACUATE Crowd Prediction Module.

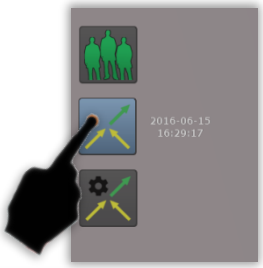
    The scroll bar enables the user to unfold the current evacuation in time, demonstrating how the crowd would evacuate given the current circumstances (densities, blockages, exit locations...)

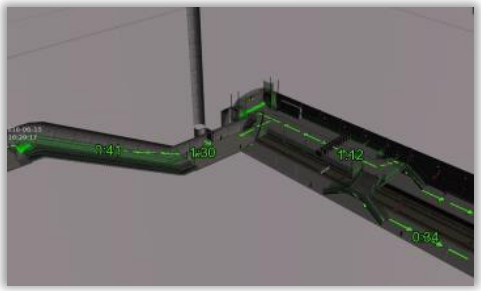
    Densities are displayed in a "traffic light" fashion from green meaning lightly crowded to red meaning highly crowded.



3. The AER display button triggers the Active Evacuation routes display within the 3D view.

   The routes are displayed with a "traffic light" representation, green meaning people will quickly evacuate from this point of the building, red meaning people at this specific place will evacuate the building in a time longer than the normally accepted time.

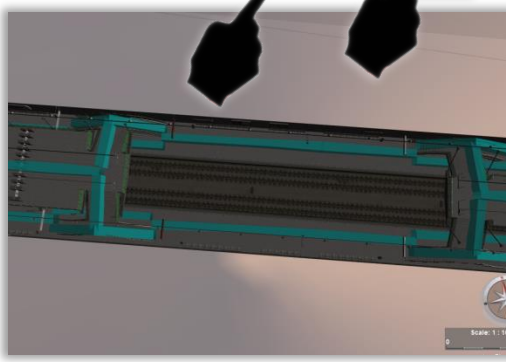


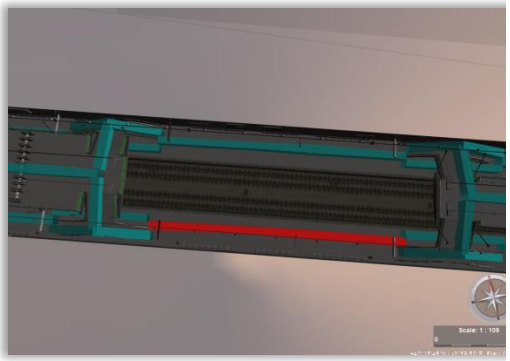


4. The AER modification button, when pressed, displays the passageways within the building. Each passage can be clicked to switch its state.

   When first clicked, the passage way turns red and the information is sent to the eVACUATE system. The next AER computation will take this information into account and will compute a new route with a blockage at this location.

   The user can click the passage again to clear the blockage.



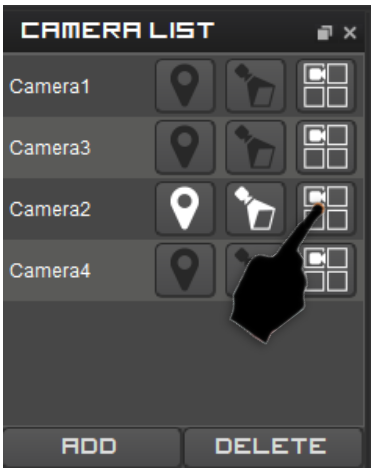


2.1.9 Camera list and grid

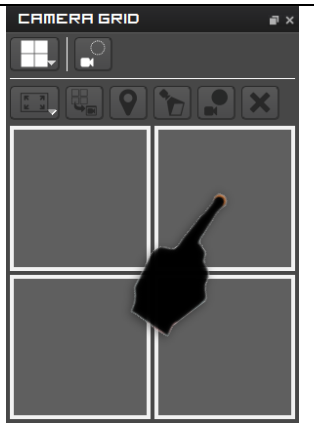
The Camera module enables to watch many video stream from a list of camera. The Camera list tab show the list of camera and the Camera grid show video stream.

2.1.9.1 Display the video stream of a camera

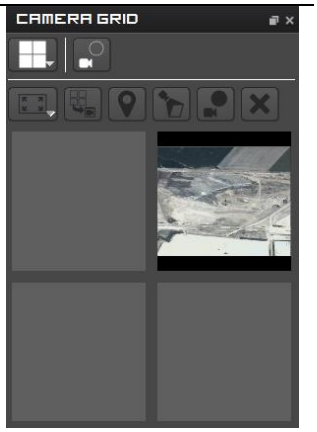
1. Press the assign button of a camera in the camera list.



2. Press on a slot of the camera grid.



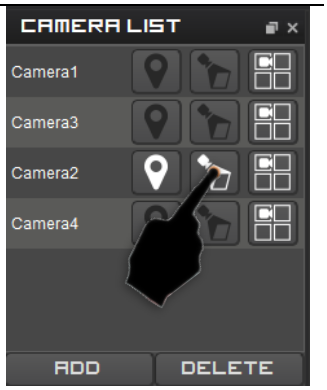
3. The view of the camera appear in the pressed slot.



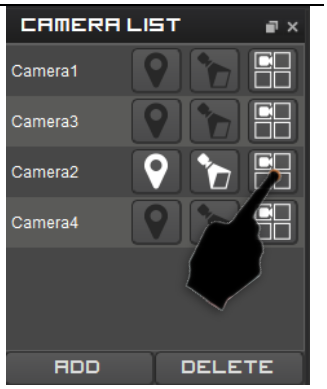
2.1.9.2 Camera list functionalities

1. Press the view point button of a camera to show this localized camera on the centre of the Map

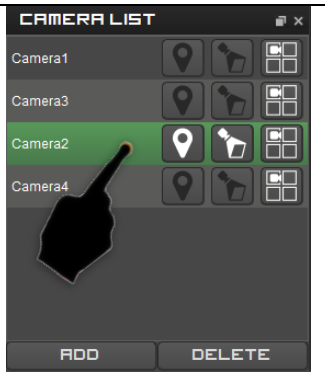
2. Press the project button of a camera to show the projection of the camera stream on the map.



3. Press the assign button of a camera to assign this camera to a slot of the Camera grid.

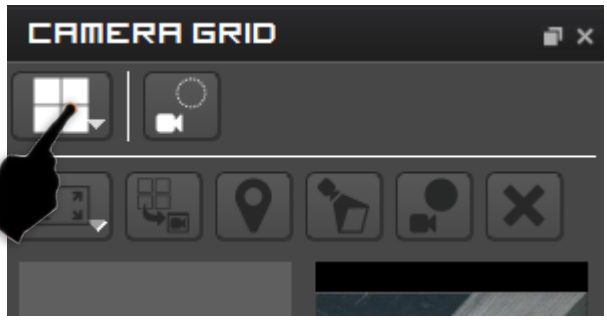


4. Press a localized camera to show the green pyramid of the camera view in the Map.



2.1.9.3 Camera grid functionalities

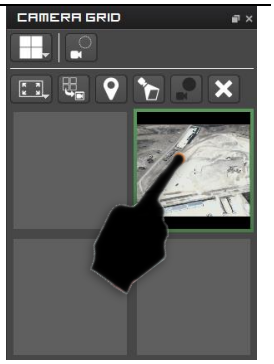
1. Press the layout button to select the layout of camera view.



2. Press the selection button to draw a circle on the Map.

   Cameras inside the circle will be shown in the grid view.

3. Select a slot with camera viewer



   a. Press the full screen button and select a screen to show the viewer on full screen.

   b. Press the deport button to deport viewer on an external window.

c. Press the view point button to show the localized camera on the centre of the Map.

d. Press the project button to show the projection of the camera stream of the map.

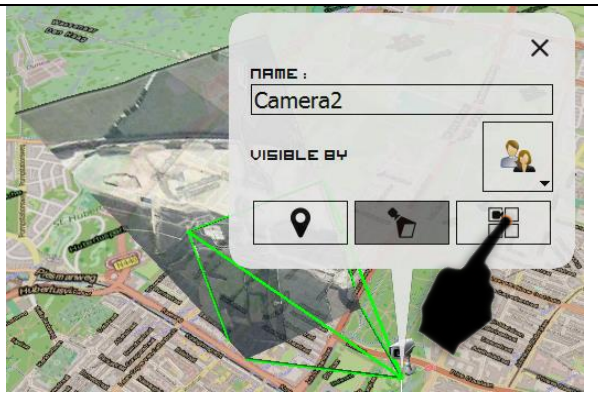
e. Press the close button to close the camera viewer of the selected slot.

2.1.9.4 Balloon view functionalities

1. Press the view point button of a camera to show this localized camera on the centre of the Map.

2. Press the project button of a camera to show the projection of the camera stream on the map.

3. Press the assign button of a camera to assign this camera to a slot of the Camera grid.



2.2 APPS FOR SMARTPHONE

The eVACUATE Mobile App (eVAMAPP) platform is composed of 3 separate applications.

The first is the main **eVAMAPP Application** intended for facility visitors and cruise ship passengers. The **eVAMAPP for First Responders** in case of emergencies, short name is eVAMAPP FR and finally the **Missing Persons Application**, again in the case of cruise ships, which will also be used by FRs in the Muster/Assembly stations to monitor the progress of the evacuation and identify individuals who may need assistance.

2.2.1 eVAMAPP

2.2.1.1 Application installation

First we need to download and install the application from the AppStore (iOS devices, iPhone/iPad).

We start the application and we go to the Settings menu, select a venue, so that the system may fetch the respective configuration and download its settings.

We restart the application.

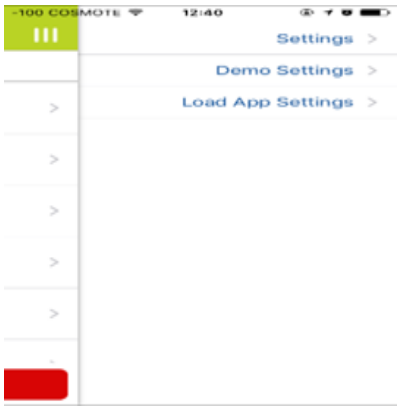


Figure 12: Settings menu

2.2.1.2 Application use

2.2.1.2.1 Home page

There is a "Call for Help" red button on the bottom part of every screen, that will display the emergency contacts, which are predefined, however may also be location-specific in the case of a cruise ship, they may be relevant to the specific country or port the ship visits.

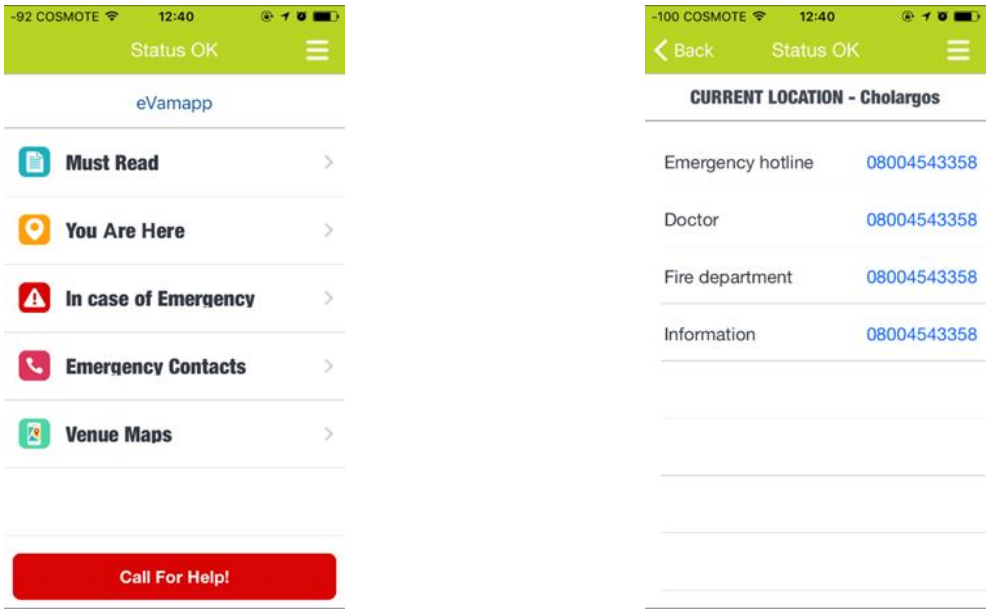


Figure 13: A call for help button appears on the bottom part of every screen (here shown the main screen), which displays venue-specific emergency contact numbers

On the main screen one may observe the main actions presented: "Must read", "You are here", "In case of emergency".

2.2.1.2.2 Status bar

On the upper side of the application, there is the status bar, that presents by color code, the situation, which might be Green (“the situation is OK”) or Red (“Evacuate”).

2.2.1.2.3 Messaging

When the evacuate status changes or something changes the active evacuations routes, a push notification alert will be sent to the smartphone.

2.2.1.2.4 Safety related information

In the menu selection "Must read", a user may find useful safety-related information about the selected venue.

All the information into the application (e.g. Must read section) may change dynamically from our server or when loading new configuration from our cloud server.

The following figure shows an example display of the safety instructions in the “Must Read” section.

Figure 14: Safety instructions (Metro example)

2.2.1.2.5 Location based information

In the “You are here" selection, one may see the map of the venue and the current position.

Figure 15: "You are here" presents the current location on a facility map

In case the user selects "In case of emergency", he is presented with the Active Evacuation Route (from the current location). He also has the opportunity to send an alert to the COP for an incident.

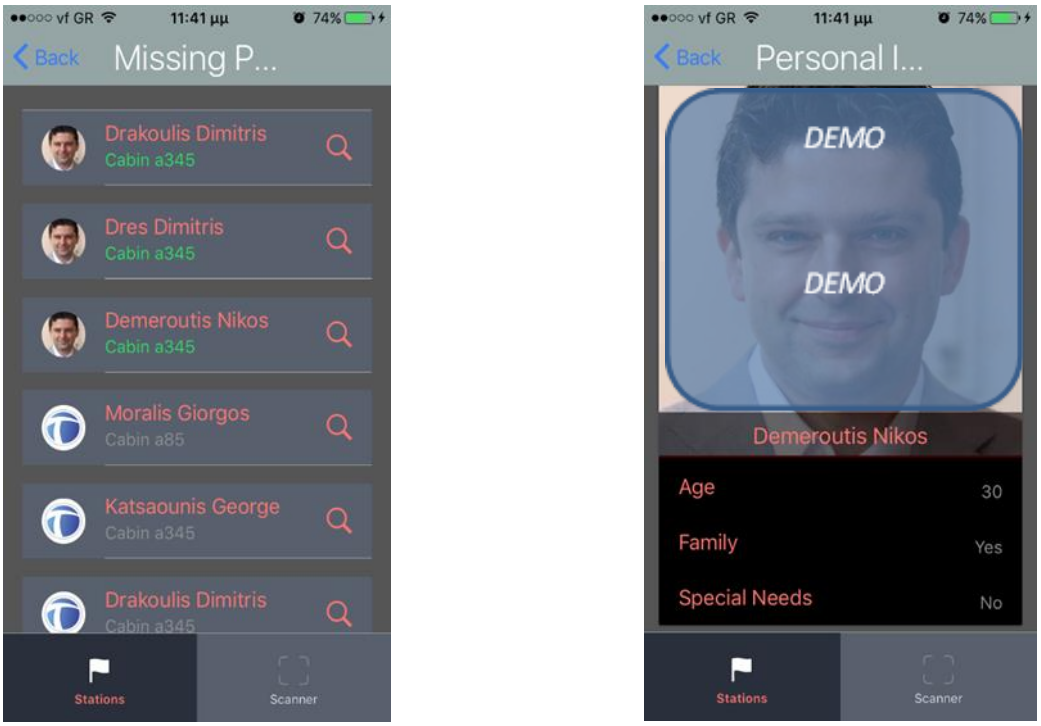


Figure 16: In case of an emergency the user may alert the authorities, using predefined messages for several types of events, or by composing a custom message (keyboard setup is local for the specific smartphone)

2.2.2 First Responder Application

2.2.2.1 Installation & configuration

The First Responder application is manually installed through the Apple TestFlight (available only for iOS devices, iPhone/iPad), it is not available in the App Store. The local network configurations need to be set up and the application restarted.

2.2.2.2 Functionality

Regardless of his/her actual position, a First Responder can "Check in" in a POI or a FR-location, to indicate that he is on duty on the selected post. The check-in location may be changed at anytime.

2.2.2.3 Alert sending

An FR may send an alert to the COP with the predefined message buttons or compose a custom message.

2.2.2.4 Receiving messages

When the EOC sends a message, it will be displayed on the top of the screen. The FR may change its status by pressing on it.

2.2.3 Missing Person Application

The application is available for Android and iOS smartphones.

2.2.3.1 Installation & configuration

Manually install it (with testfligh on iOS devices, with apk package on Android devices, on a Web Server). Open the application. One needs to set up the Mustering Locations (Names, image, capacity etc). While the passenger list (information for each passenger) must also be inserted.

2.2.3.2 Application functionality

The FR will first check in into a mustering station. A vertical slider offers an overview of the Muster Stations. For the selected Muster Station, one may observe vital statistics that show the overall evacuation / mustering progress.

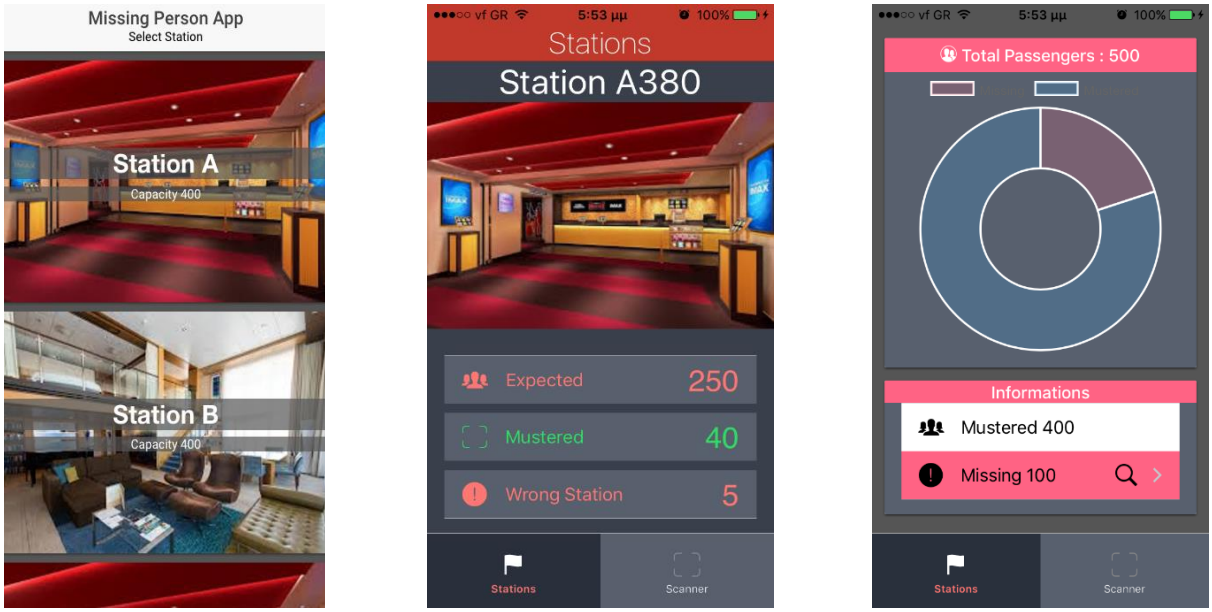


Figure 17 – Muster Station Information, from Left to Right: (a) Overview of Muster Stations, (b) Key Figures for a Muster Station (c) Statistics about mustering progress

The information offered about a specific Muster Station is (a) how many people are expected, (b) the number of people already mustered (c) number of people who should be in another station. A chart view of the station is also offered.

2.2.3.3 Scanning

To identify who is mustered and who is not, the QR code (could also be any type of code supported by the smartphone used by the staff) of a ticket needs to be scanned. The sequence is shown in the following figures.

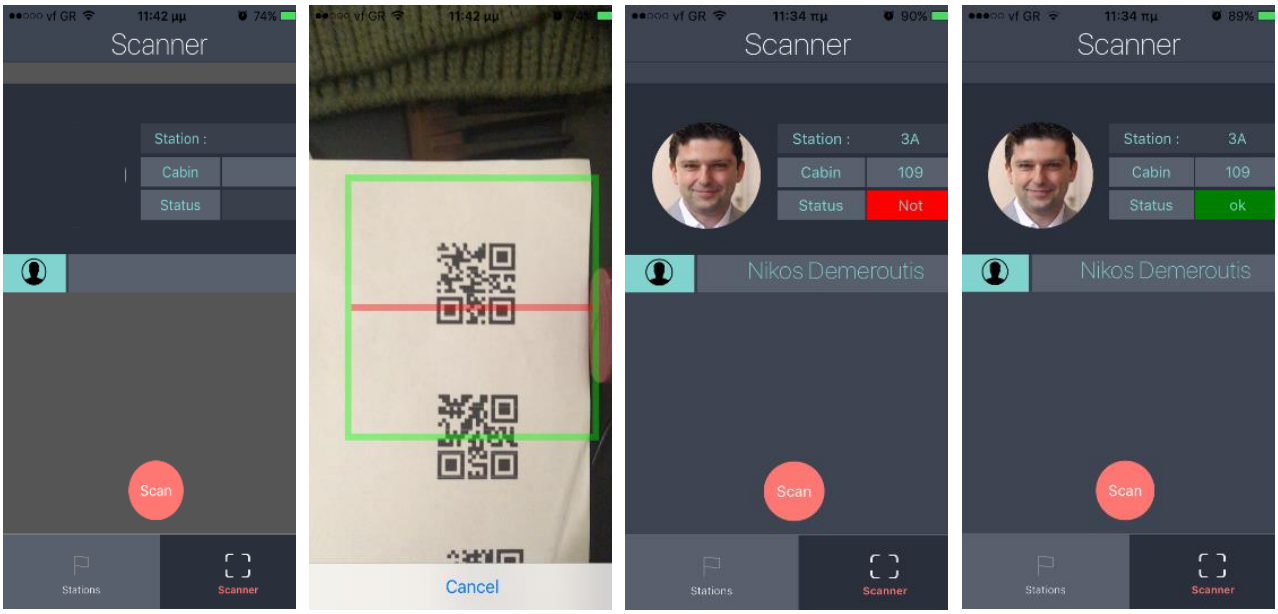
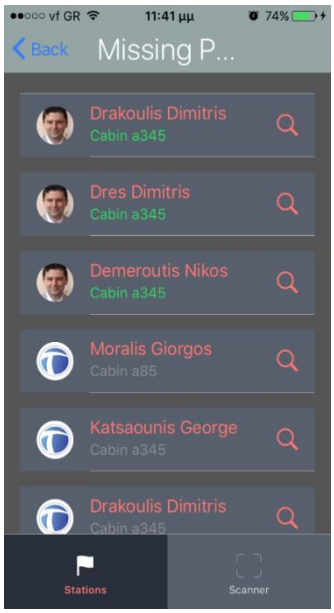
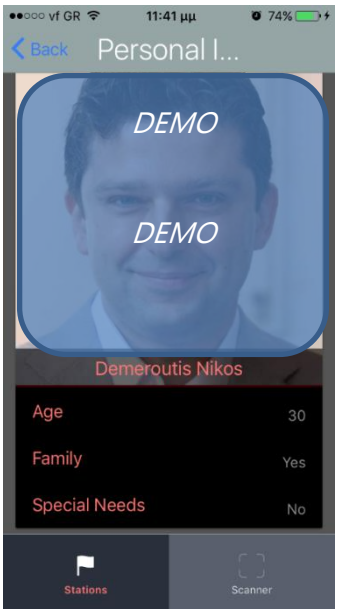


Figure 18: Scanning process, from Left to Right: (a) Scanner selected, (b) Ticket QR identified (c) Passenger is in the incorrect Muster Station (d) Passenger is in the correct Muster Station

2.2.3.4 Identifying missing persons

Once every person is in the Muster Station, the Crew Member may see the full list of missing persons, by selecting the “missing” tab. For each person the list contains only the cabin number and a thumbnail photo. However by selecting a specific missing person, one may also see the vital data for communicating with a search party who may seek to locate this passenger in case of an emergency.





2.3 EMERGENCY OPERATION CENTER

Based on the defined user requirements from D.2.1 and D.2.2 reports, the eVACUATE EOC provides the following functionalities, categorized as shown below:

- **Network infrastructure** for communication between Servers, Desktops, Sensors, Legacy and External devices and Mobiles phones. The communication can be LAN (Wired Ethernet) or wireless (Wi-Fi).
  - **Big Data** and **Cluster setup** offer best performance characteristics and using Ambari manager we have a Dashboard view of the **current** and **historical server performance**. View the historical log of the whole eVACUATE Platform.
- **Management of the Media File repository**. The Media File repository stores pictures, video and sound that are used from the Digital Signs. The operator should be able to add/edit/remove a media file from the file repository.
- **Capability to take over the control of the local Communication Gateway**. This is done with the use of Operator/PPDR Gateway and by providing a GUI to control all external and legacy devices. A list of all available over right functionalities is shown below:
  - Send an Alert/Text message to FR or User mobile application
  - Send an message to FRs TETRA device
  - Control the Exit Sign
  - Control the Digital Sign
    - STX PA System
    - STX IPTV System
    - METB Speaker System
  - Send a message to DECT Phone (in Cruise Ship)
- Provide the **status of all HW Components** of eVACUATE platform.
- Provide Interfaces for the following functionalities :
  - Send an Alert/Text message to FR or User mobile application
  - Send an message to FRs tetra device
  - Control the Exit Sign
  - Control the Digital Sign
  - Send a message to Dect Phone (In passenger suites)
  - View the historical log of the whole eVACUATE Platform
  - Perform Analytics on sensor data
  - Manage the Crisis details
- Provide the **infrastructure for eVAMAPP back office**.
- Provide the **infrastructure for COP**.
- API for **accessing the EOC Modules** using SOFIA and Restful API.
- **Ability to control the H/W modules** using a Restful API.
- Manage the **Users accounts and Users authorization**.

All functionalities supported by EOC and EOC's user manual are included and described in details in **D8.4 v2.0 report** and specifically in Section 2. For simplicity reasons and in order to avoid replicating the same information as such we prompt the reader to refer directly to the specific report.

3 ADMINISTRATION AND CONFIGURATION GUIDE

In this chapter is described how configure and maintain each component.

3.1 SAES (SITUATION AWARENESS AND EVACUATION SYSTEM) FRAMEWORK

3.1.1 The SOFIA2 Environment

3.1.1.1 Installation of SOFIA2's SDK

To operate properly with a K.P., users must follow a number of steps:

Download the Environment's last version from this URL:
http://sofia2.com/desarrollador_en.html#descargas . Then access the section Downloads.[1]

Uncompress the ZIP file that you have downloaded, SOFIA2_SDK.zip.

You will find all the components making up the SDK inside the SOFIA-SDK directory.

- The development environments for the C environments in Arduino and Java are in the Arduino and STS directories, respectively.
- The ROO directory includes the Architecture console with the SOFIA commands to create Plugins, KP's and Gateways.
- The CXF directory includes the productivity tools to work with WSDL and WADL.
- The Maven and M2_REPO directories include the Maven Tool and the dependencies it uses to develop on the SOFIA Platform.
- The API directory includes JavaScript and Java API's to develop KP's.
- The Workspace directory includes the IDE's' configuration directories.
- The Scripts directory provides the batch processes for the different components making up the SDK.

Once SOFIA2's SDK has been downloaded, access the SOFIA root directory and double-click the batch process Sofia2_SDK-START.bat to create the virtual drive S: and open a command console. This command console's session is configured to run with SOFIA's SDK and Runtime. Installing SOFIA2's SDK modifies the Sofia2_SDK-START.bat file, created by the Runtime. Due to this, installation directories must be observed.

If the command console is closed, we must re-run the batch process to re-open the console with the predefined configuration.

3.1.1.2 Opening of IDE and Consoles

Now, run the command S:\>Sofia2_IDE in the console. The iTR Architecture Eclipse IDE to develop SOFIA2 applications will open.

Run the command S:\>arduino in the console. The IDE to develop Arduino clients will open.

[1] Bear in mind that SOFIA2 was developed in Spain and some prompts and messages will be in Spanish language, particularly in command line environments.

The Architecture Console offers productivity tools to easily create SOFIA2 Projects. Run the command S:\>arqspring in the console. The iTR Architecture console will open. It provides us with the command ArqSpring>sofia create(Gateway, KP, Plugin) which will create a basic skeleton for a Java project where we can develop one of these components.

3.1.1.3 Application development

The first step is joining the SOFIA2 platform, unless of course you already have a username. Users can join the platform autonomously, or an administrator can add the users to the system. To join autonomously, access *http://sofia2.com/console/login;lang=en and select "Register for this site". This will re-direct you to a registration form. By joining autonomously, you get the role USER ("USUARIO") in the platform, lacking any special permissions. These users must ask an administrator to get special permissions.

Alternatively, the administrator can access a User Management area in the Platform. This area provides the administrator with a form to register new users in the platform, and to give them any needed role.

Once the user is registered in the Platform, the user can access the Platform according to her role. We will later see that the user can become the owner of several KP's, and those KP's can establish a logical session to send and receive information.

3.1.1.3.1 User roles

The platform has three main defined Roles, which specify the functionalities available to users at an administrative level:

- Administrador (Administrator)
- Colaborador (Collaborator): This role allows user to operate with the platform's information, spilling and consuming information, and to create new information structures. It also allows users to perform information-processing tasks (through rules, scripts, reports, etc.)
- Usuario (User): This role allows users to operate with the platform's information, spilling and consuming information from existing information structures where that user has been authorized for these operations.

3.1.1.4 Ontologies

The SOFIA Platform offers the following as the key of application interoperability: The ontological definition of the information that applications exchange with each other.

An ontology is a classification of information, standardizing the property of domain concepts with which the different KP's will interoperate. Thus, different KP's working with the same ontologies can exchange information through the platform in a completely uncoupled way, by exchanging instances of those ontologies.

SOFIA2 suggest the information exchange between KP's to use JSON format. As an ontology is, in SOFIA2, the unambiguous specification of a JSON-formatted information, those ontologies are defined following a JSON schema.[2]

[2] http://json-schema.org/

Its object is identifying the entities which will group the data exchanged by the KP's through the Platform.

One concept will cover one or several related data, which is or are interesting for the KP's. Thus, one information producer KP will collect all these data, group them following the specification that was defined in the ontology describing that concept, and send them that way, normalized to the Platform. On the other hand, an information consumer KP will receive and exploit all these normalized data.

The concept will also have attributes. Their goal is identifying the data grouped following the information concepts, and which are relevant for the KP's.

After identifying the data which will be exchange, the next step is standardizing that data so that they have an unambiguous definition as applied to the KP's in the Platform. This is the goal of the information ontologization, where every relevant concept is defined following a JSON schema.

3.1.1.4.1 Ontology Example

For instance, a SmartCity sensorization SmartSpace will have the following relevant concepts to be ontologized:

- TemperatureSensor
- HumiditySensor
- CO2Sensor
- MovementSensor
- TrafficLight
- Sprinkler

We may consider the following attributes:

- **SensorIdentifier**: string
- **Timestamp**: integer
- **Measurement**: double
- **Unit**: string
- **GPS Location**: Object
    - **Altitude**: double
    - **Latitude**: double
    - **Longitude**: double

The TemperatureSensor concept, with these attributes, can be defined in JSONSchema format like this:

TemperatureSensor.json

```
{
    "$schema": "http://json-schema.org/draft-03/schema#",
    "title": "TemperatureSensor Schema",
    "type": "object",
    "properties": {
        "_id": {
            "type": "object",
            "$ref": "#/identifier"
        },
        "TemperatureSensor": {
            "type": "string",
            "$ref": "#/data"
        }
    },
    "identifier": {
        "title": "id",
        "description": "inserted Id for TemperatureSensor",
        "type": "object",
        "properties": {
            "$oid": {
                "type": "string",
                "required": false
            }
        }
    },
    "data": {
        "title": "data",
        "description": "Info TemperatureSensor",
        "type": "object",
        "properties": {
            “identifier”: {
                "type": "string",
                "required": true
            },
            "timestamp": {
                "type": "integer",
                "minimum": 0,
                "required": true
            },
            "measurement": {
                "type": "number",
                "required": true
            },
            "unit": {
                "type": "string",
                "required": true
            },
            "GpsLocation": {
                "required": true,
                "$ref": "#/gps"
            }
        }
    },
    "gps": {
        "title": "gps",
        "description": "Gps TemperatureSensor",
        "type": "object",
        "properties": {
            "altitude": {
                "type": "number",
                "required": false
```

```
            },
            "latitude": {
                "type": "number",
                "required": true
            },
            "longitude": {
                "type": "number",
                "required": true
            }
        }
    },
    "additionalItems": false
}
```

Thus, the information produced and consumed by KP's and coming from this kind of sensor will be standardized following this format:

TemperatureSensor-instance.json

```
{
  "TemperatureSensor": {
    "identifier":"ST-TA3231-1",
    "timestamp":1357930309163,
    "measurement":25.1,
    "unit":"C",
     "GpsLocation":{
        "altitude":0.0,
        "latitude":40.512967,
        "longitude":-3.67495
    }
  }
}
```

A new ontology must be registered in the Platform before it is operative and available for the KP's to produce or consume information described in that ontology.

To do so, the Platform's Console has a section of **Ontology Management,** where ontologies can be administered, added and edited:

3.1.1.5 KP Development

A KP is any application that produces or consumes information to collaborate with others through the Platform, thus creating a ***SmartSpace*** with the KP's it works with.

To develop a KP you must program its business logic and also perform the following steps on the Platform:

For a user's KP's to produce or consume data from a given ontology, that user must have the right permissions on that ontology.

An ontology that is registered in the Platform may not be visible for a certain user, or it may be visible but that user is limited in operating it due to lack of permissions.

The platform provides administrators, through the section **Ontologies > Authorizations to my Ontologies,** a screen to manage a user's authorizations on the different registered ontologies.

Thus, depending on the KP's that the user is going to develop, the user must receive INSERT, **QUERY** or **ALL** permission on the ontology describing the data that the KP will use.

3.1.1.6 KP Registration

Users must register their KP's in the Platform. Otherwise, the Platform will reject the connection of that KP's.

To register a KP, the Platform provides an area of **KP Management**, where a user can create a new KP or manage those KP's that have been created previously.

A KP can make use of one or several ontologies. This is the Platform's information that the KP will produce or consume.

Once the KP is registered in the Platform, the KP can establish connections with the Platform.

3.1.1.7 KP Connection

The connection of a KP with the platform must be understood in two layers.

- **Physical connection**: Established by the transport protocol used by a KP to connect (TCP/IP, MQTT, JMS, Ajax-Push…). The specifics to achieve this kind of connection greatly depend on the used KP's API (Java, JavaScript, Arduino, C++...).
- **Logical connection**: Established by the SSAP (Smart Space Access Protocol), the messaging protocol defined in SOFIA. It is common to all the KP's API's.

We will now focus in the logical connection that a KP must keep with the Platform.

For a KP to connect to the Platform and produce or consume data and interoperate with other KPs, the KP must first open a session with a SIB in the Platform.

The SSAP provides two operations in that sense:

- **JOIN**: Where a KP reports its owner's user name, password and instance data to the Platform. If everything is correct, the Platform authenticates the KP and opens a session with it.
- **LEAVE**: Where a KP wants to close the session and reports that to the Platform.

If there is a session between the KP and the Platform, the KP can use the other commands in the SSAP to produce and/or consume information.

3.1.1.8 Information exchange with a KP

Once you have built the JSON message with the data to be sent to the Platform, you must then build the corresponding INSERT message of SSAP, which will integrate the data.

The Platform will validate that the KP's owner user has the corresponding permission on the ontology representing such data, and it will also validate the data as following the ontology's JSONSchema.

If there is any problem, the KP will be notified. Otherwise, the data will be added to the SIB's real-time database, and will be available for all the KP's.

The INSERT command, along with all the SSAP commands, is contemplated in every KP's API's.

Similar to a KP sending information to the Platform according to an ontology, whenever a KP receives information from the Platform, the information is formatted in JSON following the corresponding ontology. Thus, once the KP has extracted the information from the corresponding SSAP message, the KP can deal with the information following the definition of the ontology, which is defined in the ***JSONSchema***

3.1.1.9 Other possibilities

The KPs and the ontologies have more detailed guidelines publically available online at **http://sofia2.com/desarrollador_en.html**

3.2 DECISION MAKING AND OPTIMAL "SITUATION-AWARE"

3.2.1 Introduction

Decision Making is a component of eVACUATE system, it gathers signals and status from external sensors/actuators in order to perform Data Fusion and apply Inference to obtain geo-referenced events, data, features and decisions. It is composed from two distinct layers, and from this point we can call it DFMS (Data Fusion Management System):

1. Mediation Layer
2. Complex Event Processing

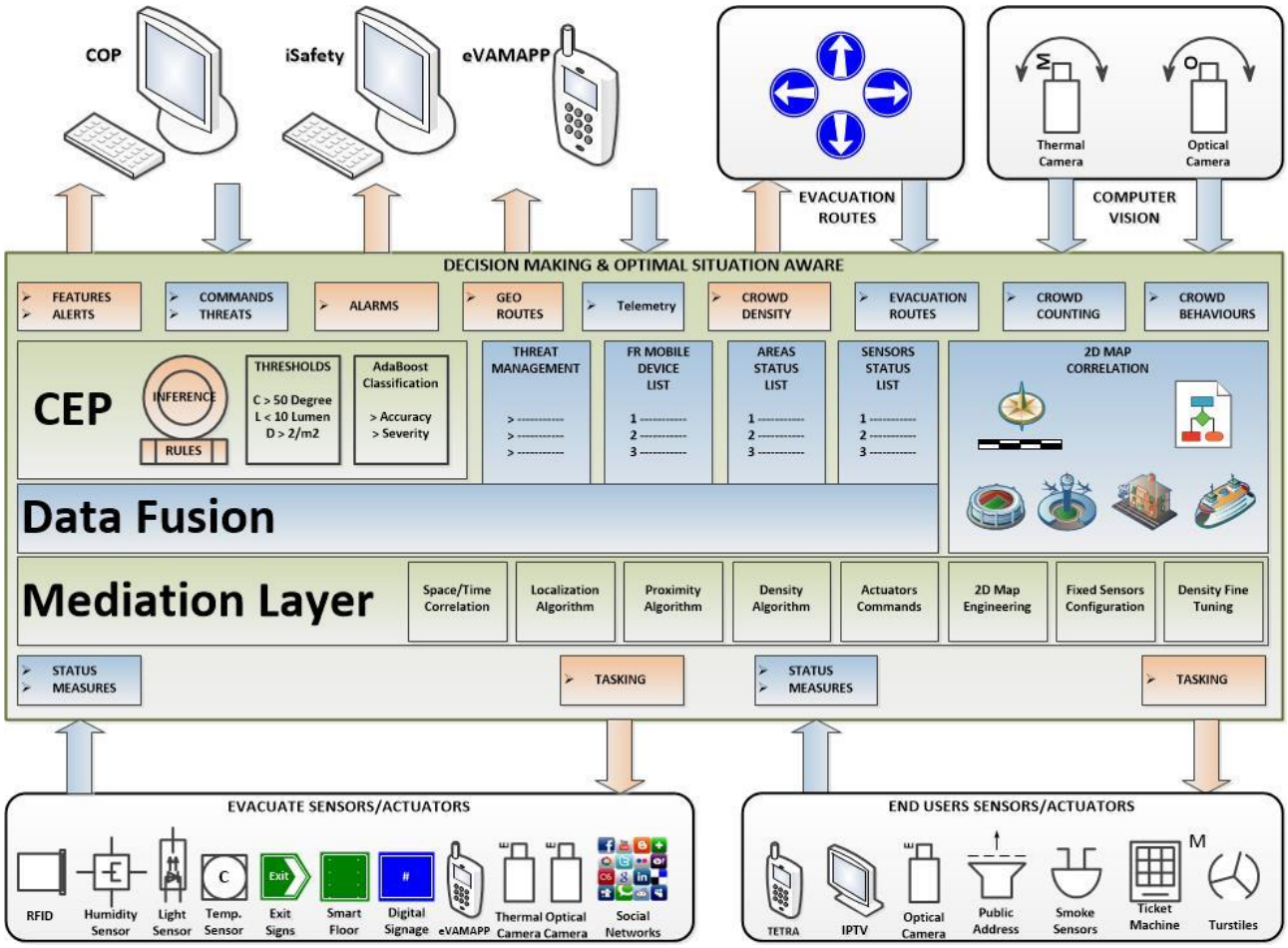


Figure 19: Data Fusion Management System Functional Block Schema

3.2.1.1 Mediation Layer

Mediation Layer is a sub-component of DFMS used as front-end layer for receive messages from all adjacent components. It is a software layer developed in java language and consist in a set of process (threads) implemented by SOFIA2 SDK. The main function is receive and elaborate messages from SIB/SSAP (Semantic Information Broker) using MQTT publish/subscribe protocol.

The architecture of the group of thread is very simple, it is composed by a Supervisor process that start and control a set of dedicated threads integrated with SOFIA2 Platform. There are two type of thread lunched from Supervisor, temporary and permanent (daemon) thread. The temporary thread

run for the necessary time to perform the job(s) accounted, when the jobs are accomplished the thread stop and exit from elaboration. The permanent run for all the time that the system run, and if permanent thread is subscribed for events we can define it a Knowledge Processors of SOFIA2 Platform (KP abbreviated), and it must be configured on the platform in order to allow connection and subscription to the events on SIB.

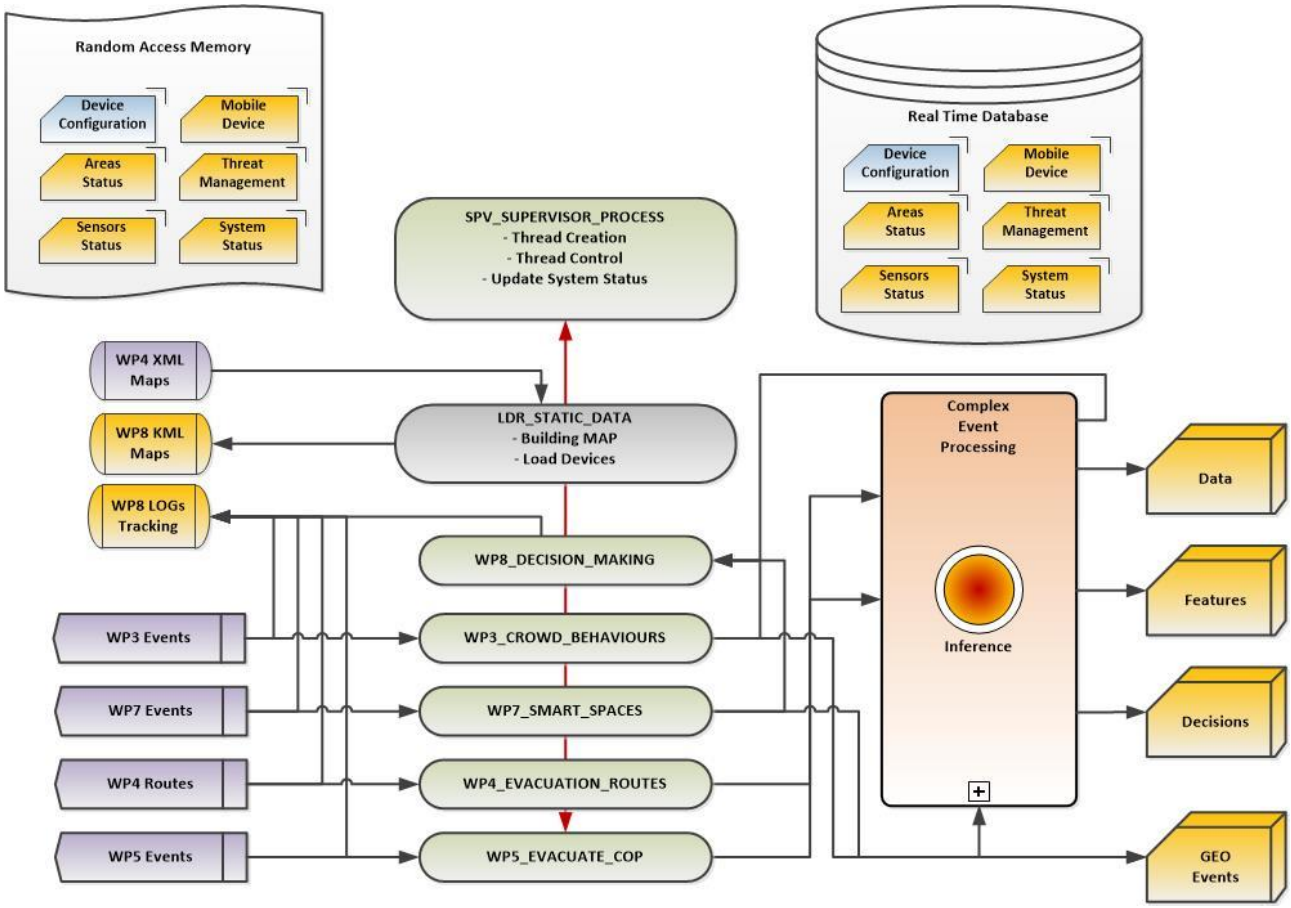


Figure 20: Architecture of Data Fusion Management System

Follow the list of DFMS's processes:

- SPV_SUPERVISOR_PROCESS

Permanent java process, create and control all threads and maintains up to date the state of the mediation layer and the state of the system. It is registered on SOFIA2 platform as Knowledge Processor of Mediation Layer.

- LDR_STATIC_DATA

Temporary java thread, it perform fast upload device configuration from file, download device configuration from database, building map from xml file, building data fusion context data in memory and on database (areas, sensors), also it creates kml static map files for every type of device and for any map layer configured.

- WP8_DECISION_MAKING

Permanent Java Thread, is a Knowledge Processor dedicated to Tracking all Events incoming or outgoing from DFMS. It subscribe for all known events and when receiving them it write a history log file. Also it is able to track all the CEP process scripts when the rules running and applying inference on data fusion. It is registered on SOFIA2 platform as Knowledge Processor of Mediation Layer.

- WP3_CROWD_BEHAVIOURS

Permanent Java Thread, is a Knowledge Processor dedicated to receive and perform space/time correlation and localization on all events coming from Crowd Behaviours component. Also it performs weighted average on configurable sliding windows containing counting people data coming from thermal camera. It is registered on SOFIA2 platform as Knowledge Processor of Mediation Layer.

- WP4_EVACUATION_ROUTES

Permanent Java Thread, is a Knowledge Processor dedicated to receive and perform space/time correlation and localization on all events coming from Evacuation Routes component. Also it perform geo routes feature for eVAMAPP and calculate the direction of the Intelligent Signage such as Exit Signs and Multimedia Devices. It is registered on SOFIA2 platform as Knowledge Processor of Mediation Layer.

- WP5_EVACUATE_COP

Permanent Java Thread, is a Knowledge Processor dedicated to receive and perform space/time correlation and localization on all events coming from Common Operating Picture component. Also it fuses and calculates counting people coming from optical camera and apply propagation of counting effect. It is registered on SOFIA2 platform as Knowledge Processor of Mediation Layer.

- WP7_SMART_SPACES

Permanent Java Thread, is a Knowledge Processor dedicated to receive and perform space/time correlation and localization on all events coming from Smart Spaces component. Also it receive Telemetries from all active First Responders Mobile Devices and maintains up to date the list for COP presentation. It is registered on SOFIA2 platform as Knowledge Processor of Mediation Layer.

3.2.1.2 Complex Event Processing

CEP is a sub component of DFMS, it map all geo events flows published from Mediation Layer and apply the Rules (Inference) using Data Fusion database tables :

- eVACUATE_WP8SystemStatus (Dynamic)
- eVACUATE_WP8AreasStatus (Dynamic)
- eVACUATE_WP8SensorsStatus (Dynamic)
- eVACUATE_WP8SpecificDensity (Dynamic)
- eVACUATE_WP8CountByCategory (Dynamic)
- eVACUATE_WP8ActuatorRoutes (Dynamic)
- eVACUATE_WP8FirstResponderList (Dynamic)

and Support database tables :

- eVACUATE_WP8Thresholds (Static Preloaded)
- eVACUATE_WP8AdaptiveBoostingWeight (Static Preloaded)
- eVACUATE_WP8Keywords (Static Preloaded)

The CEP (Siddhi) is provided by SOFIA2 Platform, it is a COTS choose from WSO2 platform. All process configured to elaborate the flows are implemented in Groove language scripting. SOFIA2 platform provide a web based application for configure and implement processes on Siddhi, also the platform provide a web page monitor in order to display processes and completion status. All steps for configuring, programming and monitoring CEP processes are detailed described on SOFIA2 manuals (“SOFIA2-Web Console Use Guide” and “SOFIA2-CEP Use Guide”).

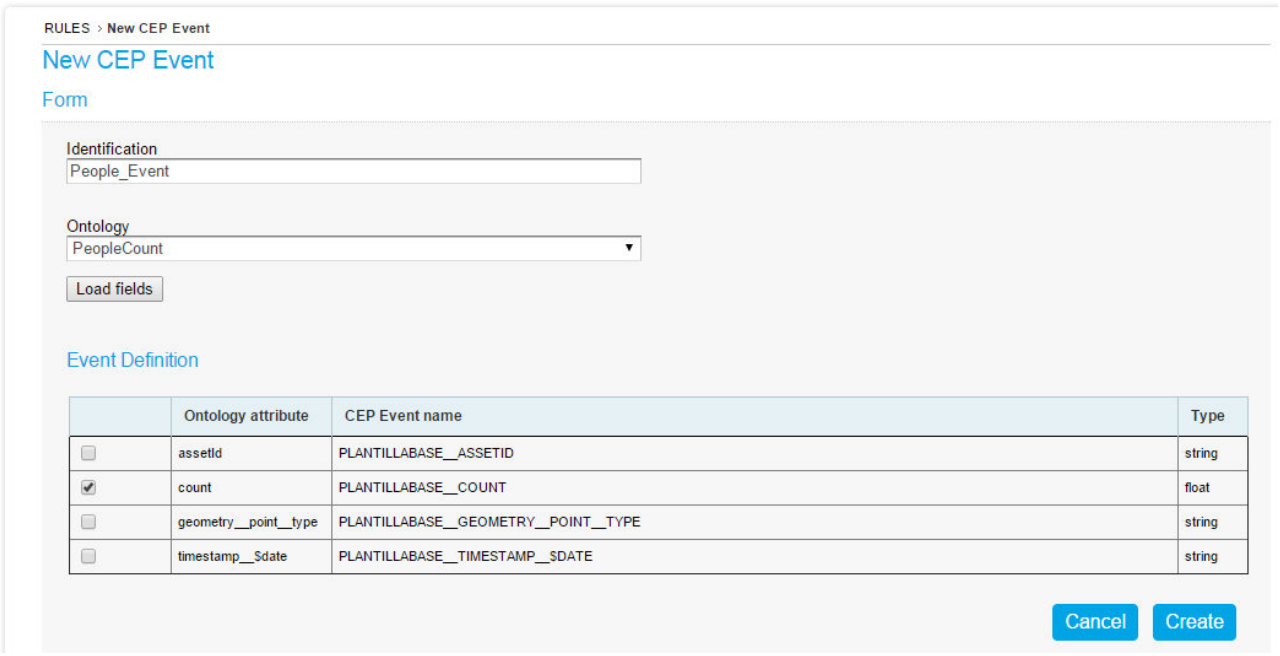


Figure 21: SOFIA2 Web Console CEP Configuration Event

3.2.2 Application

DFMS Java application is developed on SOFIA2 SDK environment, it provide a default IDE for implementation code (Eclipse) but leave free the user to choice the IDE. We have choose JetBrains IntelliJ Idea 14.0.1, and this is the environment structure tree used for develop and running DFMS application :

- WP8_DFMS_EvacuateSystem
  - jar
  - lib
  - log
  - map
  - ontologies
    - incoming
      - WP3
      - WP4
      - WP5
      - WP7
    - outgoing
      - datafusion
      - geocommstatus
      - geoevent
      - geofeature
      - geomeasure
      - geostatus
      - inference
      - system
      - tasking
    - specification
  - resources
  - src
    - com.vitro.sofia2.ssap.kp.implementation
      - ARE
      - DEV
      - DNS
      - EVR
      - FRL
      - ITF
      - KML
      - LDR
      - MAP
      - SEN
      - SIB
      - SPV
      - SYS
      - TMH
      - UTL
      - WP3
      - WP4
      - WP5
      - WP7
      - WP8

3.2.3 Configuration

DFMS configuration turn around a single java properties file. It is placed on resource directory and it is composed by one section for each thread process:

- SPV_SUPERVISOR_PROCESS Section
- LDR_STATIC_DATA Section
- WP3_CROWD_BEHAVIOURS Section
- WP4_EVACUATION_ROUTES Section
- WP5_EVACUATE_COP Section
- WP7_SMART_SPACES Section
- WP8_DECISION_MAKING Section

3.2.3.1 Processes

3.2.3.1.1 SPV_SUPERVISOR_PROCESS

- sofia.host : Host name of SOFIA2 Server Platform 185.119.248.15
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)
- kp.data : List of kp name for loading static data
- kp.static : List of kp class for loading static data
- kp.list : List of kp name for manage ontologies
- kp.class : List of kp class for manage ontologies
- kp.retry : Number of retry to launch kps when a kp have a fatal fail
- p.sample : Time to check state of kps, Value Range [10 - 1000] milliseconds
- kp.tracking : Time to display check state of kps, Value Range depend from sample, sample * tracking = milliseconds
- kp.delay : Time separation between every launch kp, Value Range [100 - 10000] milliseconds
- sys.status.ontology : Ontology name for System Status
- sys.status.instance : Ontology instance for System Status
- sys.status.data : Ontology data for System Status

- sys.status.clean : Flag to remove Sensors data before populate
- sys.status.create : Flag to insert Sensors data

3.2.3.2 LDR_STATIC_DATA

- sofia.host : Host name of SOFIA2 Server Platform
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)
- ontology.list : List of static ontologies to load
- eVACUATE_DEVSpecification.instance : Instance of specified ontology
- eVACUATE_DEVSpecification.data : Ontology data for specified ontology
- map.path : Path of maps container directory
- map.dump : Dump on log file all object found in the active map
- map.kml : Create kml map file as output of building
- map.adjacences : Distance parameter for proximity algorithm
- map.active : Active map for space/time correlation
- map.list : List of all maps readable
- map.X.file : File name of X map
- map.X.desc : Description of X map
- map.X.trans : Flag Translation of coordinate of X map
- map.X.coord : Reference coordinate to translate X map (trans=false => don't care)
- map.X.meter : Meter per Degree (trans=false => don't care)
- map.X.sign : Orientation Intelligent Signage (UP,DOWN,LEFT,RIGHT
- map.X.layer.height : Height of each physical map layer (Have a sense when it is constant for all layers)
- map.X.layer.list : List of layers present in the map

- map.X.Y.altitude : Altitude of Y layer
- map.X.head : Heading of Orientation in degree (0..360)
- map.X.propagate : List of areas affected to density propagation effects
- map.ontology : Areas Ontology name
- map.sensors.clean : Flag to remove Areas data before populate
- map.sensors.create : Flag to insert Areas data
- dev.clean : Flag to remove Device Specification Data
- dev.create : Flag to insert Device Specification Data
- dev.verbose : Flag to dump Device Data when Upload/Download
- dev.ontology : Sensors Ontology name
- dev.sensors.clean : Flag to remove Sensors data before populate
- dev.sensors.create : Flag to insert Sensors data
- dev.device : List of Device to Populate
- dev.venues : Venue list
- dev.VENUE : Device to assign at specified venue
- dev.pref.DEVICE : Prefix for Device Identifier

Devices Prefixes

- dev.CAMERA.prefix=cam
- dev.LOCATION_BEACON.prefix=tel-bea-
- dev.MOBIMESH_AP.prefix=tim-mobi-
- dev.WSN_SENSOR.prefix=tim-wsn-
- dev.ACTIVE_EXIT_SIGN.prefix=tek-aes-
- dev.MULTIMEDIA_DEVICE.prefix=exus-ds-
- dev.RFID_READER.prefix=tud-rfid-
- dev.LIGHT_SENSOR.prefix=aia-bms-
- dev.TEMPERATURE_SENSOR.prefix=aia-bms-
- dev.GREEN_BOX_DOOR_KEY.prefix=aia-bms-
- dev.GREEN_BOX_DOOR_BUTTON.prefix=aia-bms-

- dev.GREEN_BOX_DOOR_COMMAND.prefix=aia-bms-
- dev.ACCESS_CONTROL_POINT_MC.prefix=aia-bms-
- dev.ACCESS_CONTROL_POINT_COMMAND.prefix=aia-bms-
- dev.FIRE_ALARM_BUTTON.prefix=aia-bms-
- dev.MAGNETIC_DOOR.prefix=aia-bms-
- dev.TETRA_AIA.prefix=aia-tetra-
- dev.FIRE_DETECTOR.prefix=stx-fds-
- dev.DECT_PHONE.prefix=stx-dfp-
- dev.CAE.prefix=mb-cae-

Devices Suffixes

- dev.CAMERA.suffix=_Camera
- dev.LOCATION_BEACON.suffix=_Beacon
- dev.MOBIMESH_AP.suffix=_MobiMesh
- dev.WSN_SENSOR.suffix=_WSN
- dev.ACTIVE_EXIT_SIGN.suffix=_ExitSign
- dev.MULTIMEDIA_DEVICE.suffix=_MultiMedia
- dev.RFID_READER.suffix=_RFID
- dev.LIGHT_SENSOR.suffix=_BMSLight
- dev.TEMPERATURE_SENSOR.suffix=_BMSTemp
- dev.GREEN_BOX_DOOR_KEY.suffix=_BMSDoorKey
- dev.GREEN_BOX_DOOR_BUTTON.suffix=_BMSDoorButton
- dev.GREEN_BOX_DOOR_COMMAND.suffix=_BMSDoorComm
- dev.ACCESS_CONTROL_POINT_MC.suffix=_BMSPointMc
- dev.ACCESS_CONTROL_POINT_COMMAND.suffix=_BMSPointComm
- dev.FIRE_ALARM_BUTTON.suffix=_BMSFireButton
- dev.MAGNETIC_DOOR.suffix=_BMSMagDoor
- dev.TETRA_AIA.suffix=
- dev.FIRE_DETECTOR.suffix=_STXFireDetector

- dev.DECT_PHONE.suffix=_STXDectPhone
- dev.CAE.suffix=METBCAEs

3.2.3.2.1 WP3_CROWD_BEHAVIOUR

- sofia.host : Host name of SOFIA2 Server Platform
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)
- kp.sample : Time to check new message presence, Value Range [10 - 1000] milliseconds
- density.samples : Length of sliding windows (number of the samples to store)
- ontology.list : List of static ontologies to load
- XXX.instance : Instance of specified ontology
- XXX.class : Class for manage ontology
- XXX.data : Ontology data block for specified ontology
- XXX.geoname : Geo-referenced ontology name

3.2.3.2.2 WP4_EVACUATION_ROUTES

- sofia.host : Host name of SOFIA2 Server Platform
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)
- kp.sample : Time to check new message presence, Value Range [10 - 1000] milliseconds

- kp.tracking : Time to display new message presence, Value Range depend from sample, sample * tracking = milliseconds
- routes.kml : Boolean for produce kml map including dynamic route vectors and Intelligent Signage direction for tasking
- routes.dump : Boolean for dump route vector elaboration
- routes.verbose : Boolean for logging critical part of routes elaboration
- ontology.list : List of static ontologies to load
- XXX.instance : Instance of specified ontology
- XXX.class : Class for manage ontology
- XXX.data : Ontology data block for specified ontology
- XXX.geoname : Geo-referenced Ontology name produced by processing XXX
- XXX.action : Feature Ontology name produced by processing XXX

3.2.3.2.3 WP5_EVACUATE_COP

- sofia.host : Host name of SOFIA2 Server Platform
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)
- kp.sample : Time to check new message presence, Value Range [10 - 1000] milliseconds
- kp.tracking : Time to display new message presence, Value Range depend from sample, sample * tracking = milliseconds
- ontology.list : List of static ontologies to load
- XXX.instance : Instance of specified ontology
- XXX.class : Class for manage ontology
- XXX.data : Ontology data block for specified ontology
- XXX.action : Feature Ontology name produced by processing XXX

3.2.3.2.4 WP7_SMART_SPACES

- sofia.host : Host name of SOFIA2 Server Platform (192.168.1.251)
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)
- kp.sample : Time to check new message presence, Value Range [10 - 1000] milliseconds
- kp.tracking : Time to display new message presence, Value Range depend from sample, sample * tracking = milliseconds
- timer.list : Ontology name list on which executing a job after timeout
- telemetry.clean : Time in milliseconds to clean device “older than” from First Responder list
- timeout.telemetry : Time in milliseconds for timer to clean First Responder list
- ontology.list : List of static ontologies to load
- XXX.instance : Instance of specified ontology
- XXX.class : Class for manage ontology
- XXX.data : Ontology data block for specified ontology
- XXX.geoname : Geo-referenced ontology name
- XXX.action : Feature Ontology name produced by processing XXX

3.2.3.2.5 WP8_DECISION_MAKING

- sofia.host : Host name of SOFIA2 Server Platform
- sofia.port : Port Number of SOFIA2 Server Platform
- sib.timeout : SIB Timeout transactions (milliseconds)
- sib.url : URL to APIs services access SOFIA2 Server Platform
- kp.protocol : Protocol used by APIs to access SOFIA2 Server Platform
- kp.token : Registration token of KP (obtained from platform by KP creation configuration)
- kp.instance : Instance name of KP (obtained from platform by KP instance configuration)
- kp.keepalive : Keepalive time between SOFIA2 Platform and KP (seconds)

- kp.sample : Time to check new message presence, Value Range [10 - 1000] milliseconds
- kp.tracking : Time to display new message presence, Value Range depend from sample, sample * tracking = milliseconds
- ontology.list : List of static ontologies to load
- XXX.instance : Instance of specified ontology
- XXX.data : Ontology data block for specified ontology

3.2.4 Fundamental Steps

These are the step to configure DFMS:

- Set Venue Map : XML venue 2D Map file

Place the XML map file inside the directory map and set the property of LDR_STATIC_DATA process. This step auto-configure all system parameters, excluding Sensors/Actuators Placement and Fine Tuning parameters.

- Place Fixed Sensors/Actuators : Open the map and decide where place the devices

After a dry run of DFMS, open the kml map file (check the property about kml creation, it must be set to true) and then choice the area where place Fixed Sensors/Actuators. In the kml map the represented areas contains the geo coordinate in WGS84. Insert data inside the file for upload configuration.

- Fine Tuning : Proceed to set parameters about Density elaboration, and map layers

About density elaboration we needs to knows the number of slots of sliding windows on apply weighted average, and the list of area on which apply effect propagation of the counting people. Also the list of the layers in the map and the physical height (these last are used for draw kml maps for check device positioning). Fine tuning include the adjacencies distance for proximity algorithm and the minimum distance of First Responder for engagement when an incident occurred.

3.2.4.1 Venue Map

Set the DFMS parameters for a venue is very simple, there are 2 thread needing to know the venue for auto-configure the system :

- SPV_SUPERVISOR_PROCESS
- LDR_STATIC_DATA

Both have in the proper properties section the entries:

SPV_SUPERVISOR_PROCESS.map.active=METB

LDR_STATIC_DATA.map.active=METB

The venue list is :

- ANOETA
- AIA
- STX
- METB

After set the active map parameter we have to verify thet kml map creation is set to true.

Some properties like area.clean area.create are used to rebuild the data context on database.

If the number of areas in the venue is very high, the boot of the system may have a long duration.

3.2.4.2 Fixed Devices

All fixed devices must be configured in terms on type and positioning in order to be used for localise subarea and producing a geo-referenced event to be consumed by other SW components and CEP. Is possible to preload data on database using file eVACUATE_DeviceSpecificationData.json inside the directory ontologies\specification. The loader temporary thread check the property “dev.upload”, if it is set to true, the loader read the file and upload data on database. This is an example of different device type instance inside the file :

[
{"eVACUATE_DeviceSpecification_onto":{"identifier":"tek-aes-3","type":"ACTIVE_EXIT_SIGN","name":"Active Exit Sign
003","description":"FLOOR:UP","state":"DISABLED","location":{"name":"EDGE2","venue":"AIA","latitude":37.931372,"longitude":23.942075,"altitude":4.5,"orientation":{"rotationX":0,"rotationY":0,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"tek-aes-21","type":"ACTIVE_EXIT_SIGN","name":"Active Exit Sign
021","description":"WALL:UP","state":"DISABLED","location":{"name":"EDGE102","venue":"METB","latitude":43.26248345,"longitude":-2.9474031,"altitude":-36.0,"orientation":{"rotationX":0,"rotationY":0,"rotationZ":0}}}},
{"eVACUATE_DeviceSpecification_onto":{"identifier":"thermal_01","type":"CAMERA","name":"Thermal Camera 1","description":"Thermal
Camera","state":"DISABLED","location":{"name":"NODE34","venue":"AIA","latitude":37.9315969,"longitude":23.9425281,"altitude":4.5,"orientation":{"rotationX":2,"rotationY":1,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"visible_01","type":"CAMERA","name":"Optical Camera 1","description":"Optical
Camera","state":"DISABLED","location":{"name":"NODE10","venue":"AIA","latitude":37.931578,"longitude":23.9422979,"altitude":4.5,"orientation":{"rotationX":2,"rotationY":1,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"exus-ds-1621053","type":"MULTIMEDIA_DEVICE","name":"Multi Media Device
1","description":"Orientation=UP","state":"DISABLED","location":{"name":"EDGE666","venue":"ANOETA","latitude":43.300858149999996,"longitude":-1.9745002,"altitude":8.325,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"tim-wsn-0002","type":"WSN_SENSOR","name":"Wireless Sensor Network 2","description":"Wireless Sensor Network
Device","state":"DISABLED","location":{"name":"EDGE5","venue":"AIA","latitude":37.931443599999994,"longitude":23.94188535,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"tim-mobi-1","type":"MOBIMESH_AP","name":"Mobimesh Access Point 1","description":"Mobimesh Access Point
Device","state":"DISABLED","location":{"name":"EDGE21","venue":"AIA","latitude":37.931529975000004,"longitude":23.942555100000003,"altitude":4.5,"orientation":{"rotationX":2,"rotationY":1,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"tud-rfid-1","type":"RFID_READER","name":"Radio Frequency Identifier Device 1","description":"Radio Frequency Identifier
Device","state":"DISABLED","location":{"name":"EDGE15","venue":"AIA","latitude":37.931379475,"longitude":23.9424946,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-001-AVR-LIGHTAIA","type":"LIGHT_SENSOR","name":"AIA Light Sensor 1","description":"[Lighting Indicator of airport]","state":"ENABLED","location":{"name":"NODE46","venue":"AIA","latitude":37.931718,"longitude":23.94228,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002VAAHU0306RTVAL1","type":"TEMPERATURE_SENSOR","name":"AIA Temperature Sensor 1","description":"[Sensor Return Air
Value]","state":"ENABLED","location":{"name":"EDGE123","venue":"AIA","latitude":37.93151805,"longitude":23.9422543,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002IA208052000GBK","type":"GREEN_BOX_DOOR_KEY","name":"AIA Green Box Door Key

1","description":"[SAT
2/7.06(04)]","state":"ENABLED","location":{"name":"EDGE300","venue":"AIA","latitude":37.93113835,"longitude":23.942765225,"altitude":4.5,"orientation":{"rotationX":2,"rotationY":1,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002IA208053000GBA","type":"GREEN_BOX_DOOR_BUTTON","name":"AIA Green Box Door Button 1","description":"[SAT
2/7.06(04)]","state":"ENABLED","location":{"name":"EDGE300","venue":"AIA","latitude":37.93113835,"longitude":23.942765225,"altitude":4.5,"orientation":{"rotationX":2,"rotationY":1,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002AC208001000DO","type":"GREEN_BOX_DOOR_COMMAND","name":"AIA Green Box Door Command 1","description":"[SAT
2/6.26(01)]","state":"ENABLED","location":{"name":"NODE13","venue":"AIA","latitude":37.9314433,"longitude":23.94282255,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002AC204001000DO","type":"ACCESS_CONTROL_POINT_COMMAND","name":"AIA Access Control Point Command 1","description":"SAT 2/9-15/01 /
1.12","state":"ENABLED","location":{"name":"EDGE15","venue":"AIA","latitude":37.9313591,"longitude":23.9425846,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002AC204001000MC","type":"ACCESS_CONTROL_POINT_MC","name":"AIA Access Control Point MC 1","description":"SAT 2/9-15/01 /
1.12","state":"ENABLED","location":{"name":"EDGE15","venue":"AIA","latitude":37.9313591,"longitude":23.9425846,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},

{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002F_201089003PB","type":"FIRE_ALARM_BUTTON","name":"AIA Fire Alarm Button 1","description":"[SAT Room 2/9-01
/]","state":"ENABLED","location":{"name":"NODE22","venue":"AIA","latitude":37.931634,"longitude":23.942213,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}},
{"eVACUATE_DeviceSpecification_onto":{"identifier":"aia-bms-002IA201022042MC","type":"MAGNETIC_DOOR","name":"AIA Magnetic Door 1","description":"[SAT 4/0 R 2/8-05-02
/]","state":"ENABLED","location":{"name":"NODE17","venue":"AIA","latitude":37.931472,"longitude":23.941913,"altitude":4.5,"orientation":{"rotationX":1,"rotationY":2,"rotationZ":0}}}}
]

Some properties like sensor.clean sensor.create are used to rebuild the data context on database

If the number of sensors in the venue is very high, the boot of the system may have a long duration.

After file preparation we have to set the orientation for the system, assign to the exit signs the orientation type and angle:

LDR_STATIC_DATA.map.AIA.sign=UP

LDR_STATIC_DATA.map.AIA.head= 307.4977155275622

DFMS assign UP direction using as reference angle the head property, at the same time the device setting "description":"FLOOR:UP" complete the parameters for the tasking operations when a Start Evacuation is decided from COP operator.

3.2.4.3 Fine Tuning

In order to complete the configuration of system parameters of DFMS we needs to evaluate some conditions and physical characteristics of the venue, for example the physical decks in the ship, or the field of view of thermal camera used for counting people and the effect propagation of counting.

3.2.4.4 Map Layers

In the Loader section we can insert the list of the layers name and the altitudes. This setting is fundamental for localization algorithm in order to establish which layer is affected. Also it is necessary to create kml files for verify sensors/actuators placement.

This is an example of layers settings:

LDR_STATIC_DATA.map.STX.layer.height=2.8

LDR_STATIC_DATA.map.STX.layer.list=**DECK6,ESC67,DECK7,ESC78,DECK8**

LDR_STATIC_DATA.map.STX.DECK6.altitude=10.7

LDR_STATIC_DATA.map.STX.ESC67.altitude=11.1

LDR_STATIC_DATA.map.STX.DECK7.altitude=12.5

LDR_STATIC_DATA.map.STX.ESC78.altitude=13.9

LDR_STATIC_DATA.map.STX.DECK8.altitude=18.299999

This settings was used for cruiser ship, how we can see, the decks affected from the exercise are three, also we had included the escalator layers between the decks.

3.2.4.4.1 Density Elaboration

In order to apply the density elaboration DFMS needs to some properties settings. Before density elaboration DFMS apply data fusion on counting people coming from thermal camera and Simulation counting people. After data fusion DFMS algorithm dispose data inside a sliding windows slots in order to apply weighted average, the number of the slot determine the response in terms on time of the density at the same time it determine the noise reduction. The number of the slots of sliding windows is configurable

WP3_CROWD_BEHAVIOURS.density.samples=10

The effect of the value can affect time of response about density modification and noise reduction of result density value respectively :

1. High number of Sliding Windows slots give a Better Noise Reduction but an Long Time Response when Density Change
2. Low number of Sliding Windows slots give a Worst Noise Reduction but a Short Time Response when Density Change

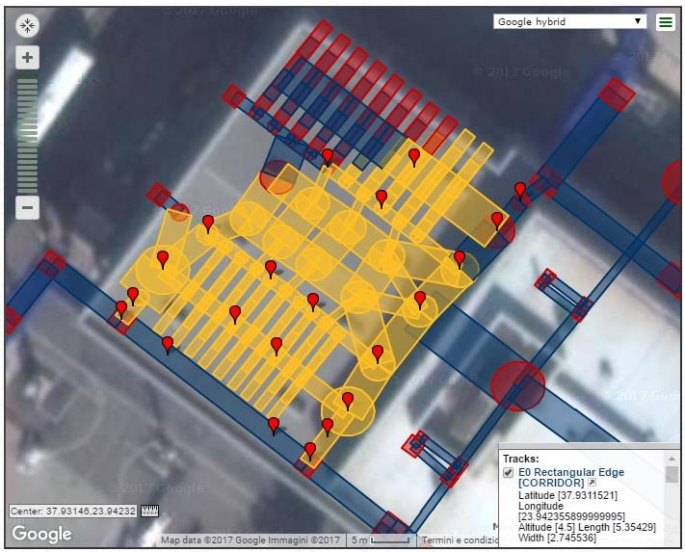
This parameter has to be fixed after some tests on the system, in order to obtain the right error filter, and at the same time, an acceptable response time to the density variations.

3.2.4.4.2 *Density Propagation*

In order to obtain a realistic exercise we have decided to extend the effect of density elaboration on a large portion of the venue, such as we disposing of a high number of thermal cameras.

Configuring a list of areas on which apply the same effect of density value of the area where placed the thermal camera, we simulate the same density of the area affected from the camera field of view.

LDR_STATIC_DATA.map.AIA.propagate=EDGE10,EDGE11,EDGE13,EDGE14,EDGE16,EDGE17,EDGE22,EDGE23,EDGE32,EDGE33,EDGE37,EDGE20,EDGE27,EDGE35,EDGE38,EDGE39,EDGE40,EDGE41,EDGE42,EDGE44,EDGE63,EDGE64,EDGE65,EDGE66,EDGE67,EDGE68,EDGE69,EDGE70,EDGE71,EDGE72,EDGE73,EDGE74,EDGE75,EDGE76,EDGE77,EDGE78,EDGE79,EDGE80,EDGE81,EDGE82,EDGE83,EDGE84,EDGE85,EDGE86,EDGE87,EDGE88,EDGE89,EDGE90,EDGE91,EDGE92,EDGE93,EDGE94,EDGE95,EDGE96,EDGE97,EDGE98,EDGE99,EDGE100,EDGE101,EDGE102,EDGE103,EDGE104,EDGE105,EDGE106,EDGE107,EDGE108,EDGE109,EDGE110,EDGE111,EDGE112,EDGE113,EDGE114,EDGE115,EDGE116,EDGE117,EDGE118,EDGE119,EDGE121,EDGE122,EDGE123,EDGE124,EDGE125,EDGE126,EDGE127,EDGE128,EDGE129,EDGE130,EDGE131,EDGE132,EDGE133,EDGE134,EDGE135,EDGE136,EDGE137,EDGE138,EDGE139,EDGE140,EDGE141,NODE3,NODE8,NODE19,NODE20,NODE21,NODE23,NODE24,NODE26,NODE27,NODE28,NODE29,NODE31,NODE35,NODE36,NODE37,NODE41,NODE47,NODE48,NODE49,NODE50,NODE51,NODE63,NODE64,NODE65,NODE66,NODE67,NODE68,NODE69,NODE70,NODE71,NODE72,NODE73,NODE74,NODE75,NODE76,NODE77,NODE78,NODE79,NODE80,NODE81,NODE82,NODE83,NODE84,NODE85,NODE86,NODE87,NODE88,NODE89,NODE90,NODE91,NODE92,NODE93,NODE94,NODE95,NODE96,NODE97,NODE98,NODE99,NODE101,NODE102,NODE103,NODE104,NODE105,NODE106,NODE107
{"eVACUATE_DeviceSpecification_onto":{"identifier":"thermal_01","type":"CAMERA","name":"Thermal Camera 1","description":"Thermal Camera","state":"DISABLED","location":{"name":"NODE34","venue":"AIA","latitude":37.9315969,"longitude":23.9425281,"altitude":4.5,"orientation":{"rotationX":2,"rotationY":1,"rotationZ":0}}}},



Density propagation give us two advantages, manage camera field of view include an high number of areas (like AIA venue) or simply propagate the density effect on more areas not covered by cameras (like ANOETA venue). To verify the physical effect propagation is possible to open a kml file produced by the loader process at the end of its jobs. The kml file could be opened by a kml visualizer. The map displays as yellow color all the areas affected from field of view included the extended area affected from propagation effect.

Figure 22: AIA map for thermal camera field of view

3.2.4.4.3 Hazardous Space and First Responder Engagement

In order to advice the COP operator populating prediction field of Notification Alert, DFMS adopt the proximity algorithm for calculate hazardous space near an incident. The same algorithm is used for populate suggestion on Notification Alert. It include the list of First Responders available near an incident that COP operator can engage for a first verification/evaluation.

LDR_STATIC_DATA.map.adjacences=20.0

The value of this properties is expressed in meters.

3.2.5 Display KML Maps

DFMS create a set of kml map files that could be displayed from a kml visualizer. This set of files is important to verify the device positioning after fill eVACUATE_DeviceSpecificationData.json data file. The number of files depend from the number of the device types and from how many layers are configured on the map:

LDR_STATIC_DATA.dev.ANOETA=

LDR_STATIC_DATA.dev.AIA=LIGHT_SENSOR,TEMPERATURE_SENSOR,GREEN_BOX_DOOR_KEY,GREEN_BOX_DOOR_BUTTON,GREEN_BOX_DOOR_COMMAND,ACCESS_CONTROL_POINT_MC,ACCESS_CONTROL_POINT_COMMAND,FIRE_ALARM_BUTTON,MAGNETIC_DOOR

LDR_STATIC_DATA.dev.STX=DECT_PHONE,FIRE_DETECTOR

LDR_STATIC_DATA.dev.METB=CAE

LDR_STATIC_DATA.dev.EVACUATE=CAMERA,LOCATION_BEACON,MOBIMESH_AP,WSN_SENSOR,ACTIVE_EXIT_SIGN,MULTIMEDIA_DEVICE,RFID_READER

LDR_STATIC_DATA.map.ANOETA.layer.list=RING1,RING2,RING3

LDR_STATIC_DATA.map.AIA.layer.list=FLOOR0,FLOOR1

LDR_STATIC_DATA.map.STX.layer.list=DECK6,ESC67,DECK7,ESC78,DECK8

LDR_STATIC_DATA.map.METB.layer.list=

After choose the venue, DFMS perform a union of evacuate device, that are present in all pilot, and specific device provided from and-user. At start-up DFMS with loader process read the map and build areas in the data context, the upload device from file to database (fast configuration) and then download ll devices from database for build the sensors data context. At the end of loader operation DFLM create one file for each device for every layer.

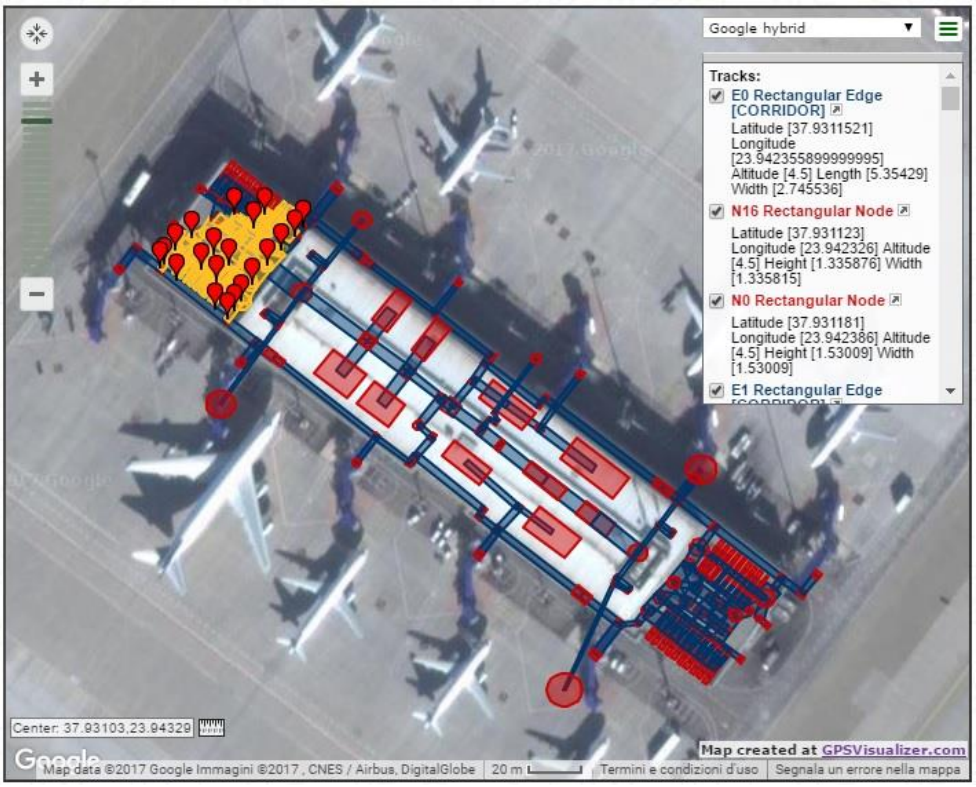


Figure 23: AIA map venue including Exit Signs and Propagation Effects

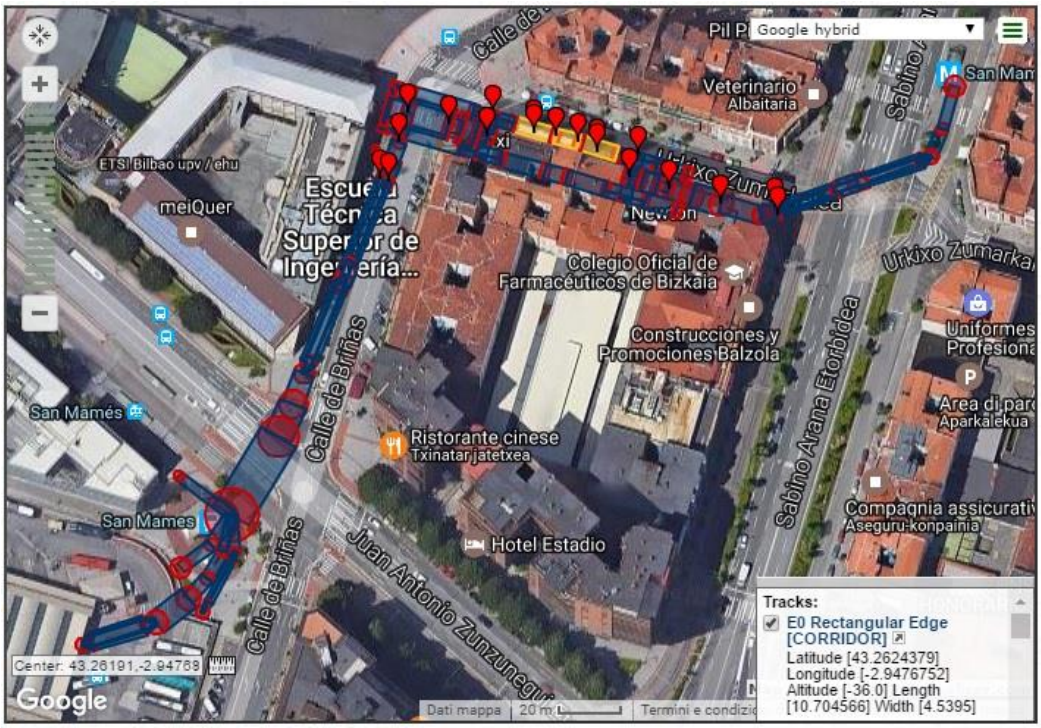


Figure 24: METB map venue including Exit Signs and Propagation Effects

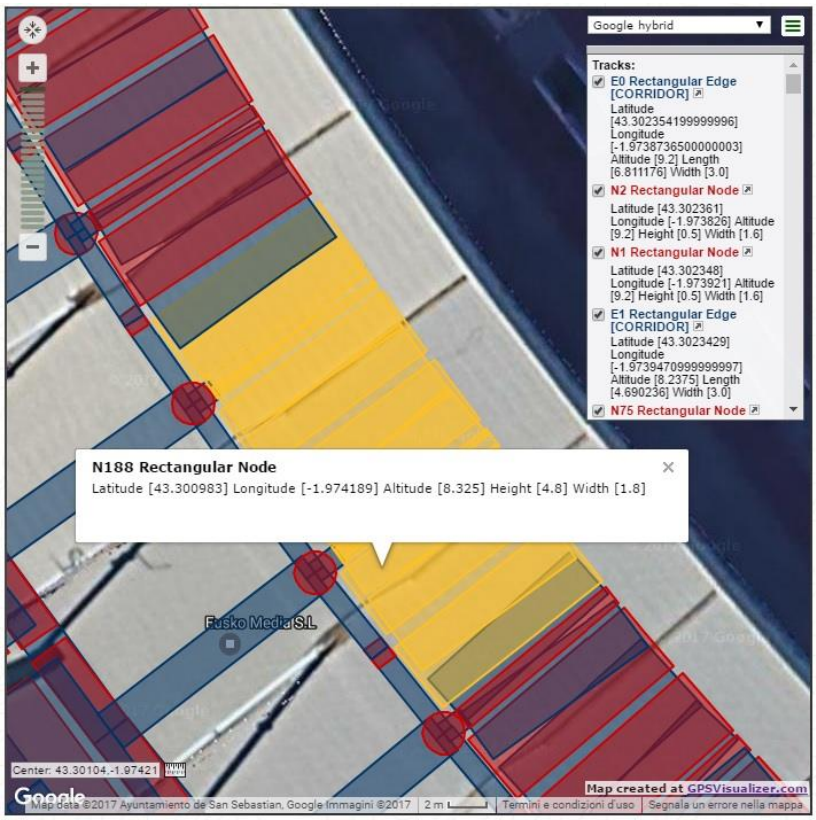


Figure 25: ANOETA map venue including Propagation Effects

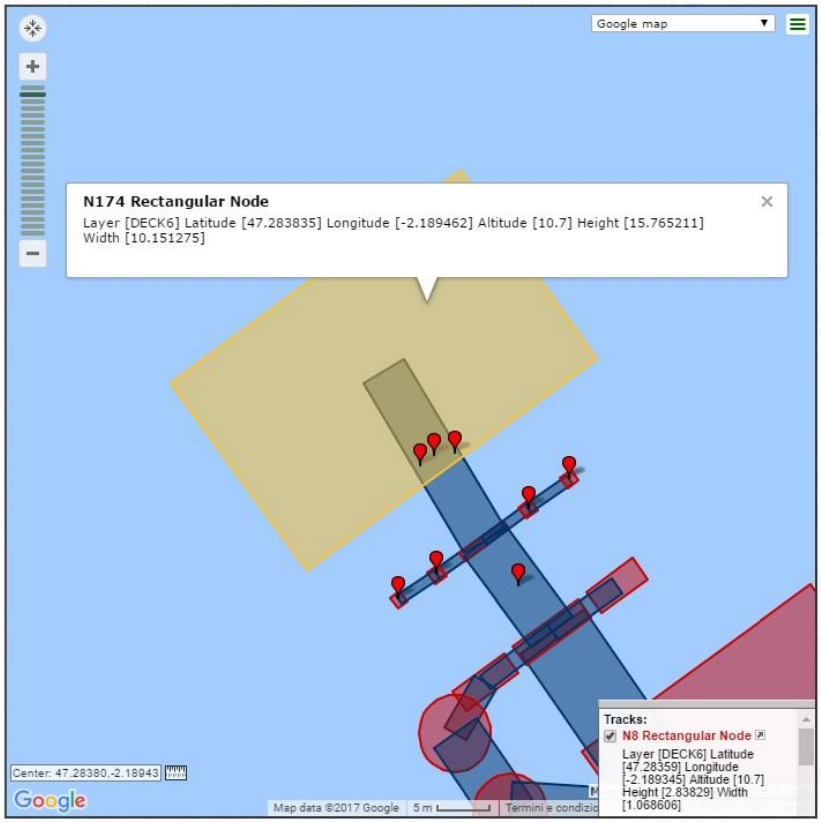


Figure 26: STX map venue including Exit Signs and Propagation Effects

Most important are the dynamic map created when DFMS receive the Evacuation Routes Data and calculate the direction of all Intelligent Signage. These maps display all route vectors, all Exit Signa and the indication of direction for tasking.

Figure 27: AIA map venue including Exit Signs indication, Propagation Effects and Route Vectors

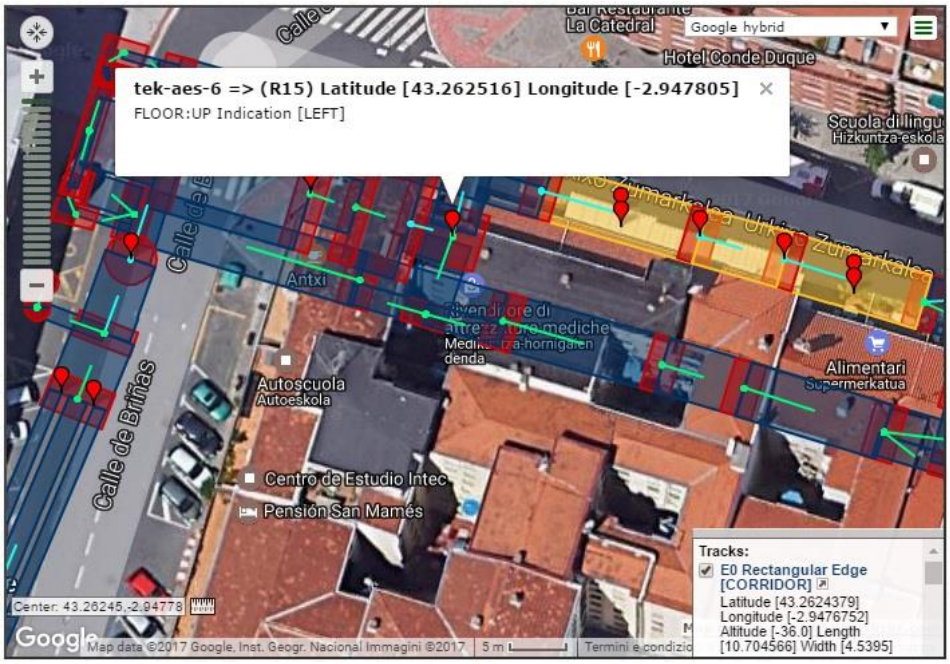


Figure 28: METB map venue including Exit Signs indication, Propagation Effects and Route Vectors

3.2.6 Log Management

DFMS dispose a large set of log files. They are placed into log directory log. In order to be compliant with SOFIA2 platform, we have choose to use log4j system for trace relevant event and implement some requirements. The configuration file for log4j is placed into resource directory with the standard name log4j.properties, follow the list and specification of the log files.

- **DFMS_EvacuateSystem.log** : It includes all events incoming and outgoing of DFMS and critical part of elaboration like density, actuators commands, change status system and change status mediation layer.
- **LDR_StaticData.log** : It includes all steps inside the jobs of loader such as loading map, building area data context, upload devices, download devices, building sensor data context and kml static map creation.
- **WP3_CrowdBehaviours.log** : It includes the elaboration of permanent thread WP3_CROWD_BEHAVIOURS, for reduce the I/O throughput and increase the performance of DFMS, it performing only critical trace such as start-up phase, incoming and outgoing events and errors.
- **WP4_EvacuationRoutes.log** : It includes the elaboration of permanent thread WP4_EVACUATION_ROUTES, for reduce the I/O throughput and increase the performance of DFMS, it performing only critical trace such as start-up phase, incoming and outgoing events and errors.
- **WP5_EvacuateCop.log** : It includes the elaboration of permanent thread WP5_EVACUATE_COP, for reduce the I/O throughput and increase the performance of DFMS, it performing only critical trace such as start-up phase, incoming and outgoing events and errors.
- **WP7_SmartSpaces.log** : It includes the elaboration of permanent thread WP7_SMART_SPACES, for reduce the I/O throughput and increase the performance of DFMS, it performing only critical trace such as start-up phase, incoming and outgoing events and errors.
- **WP8_DecisionMaking.log** : It includes the elaboration of all permanent threads but trace only the outgoing ontology messages from DFMS such as data fusion, inference, system, tasking and geo-events.
- **ITF_OntologyTracking.log** : It includes the elaboration of all permanent threads but trace only the incoming and outgoing ontology messages from/to DFMS in order to have the historical data exchanging of DFMS.

- **CEP_DataFusion.log** : It includes the script elaboration of Rules in the CEP related to data fusion process jobs. In order to reduce the I/O throughput and increase the performance of CEP, it performing only critical trace such as error occurrence.
- **CEP_CreatePreNotificationAlert.log** : It includes the script elaboration of Rules in the CEP related to steps of pre alert jobs. In order to reduce the I/O throughput and increase the performance of CEP, it performing only critical trace such as error occurrence.
- **CEP_CreateNotificationAlert.log** : It includes the script elaboration of Rules in the CEP related to steps of alert jobs. In order to reduce the I/O throughput and increase the performance of CEP, it performing only critical trace such as error occurrence.
- **CEP_StartEvacuationCommand.log** : It includes the script elaboration of Rules in the CEP related to steps of Start Evacuation jobs. In order to reduce the I/O throughput and increase the performance of CEP, it performing only critical trace such as error occurrence.
- **CEP_EndEvacuationCommand.log** : It includes the script elaboration of Rules in the CEP related to steps of End Evacuation jobs. In order to reduce the I/O throughput and increase the performance of CEP, it performing only critical trace such as error occurrence.
- **CEP_Threat.log** : It includes the script elaboration of Rules in the CEP related to steps of Threat Management jobs. In order to reduce the I/O throughput and increase the performance of CEP, it performing only critical trace such as error occurrence.

3.2.7 Startup and Shutdown DFMS

3.2.7.1 Startup

For run DFMS you have sure that artefact jar file is present into the directory jar, and then execute the batch script **startDFMS.bat**.

The command file simply set an environment variable used to localise the properties file of DFMS, and then launch the executable jar.

set WP8_DFMS_PROPS=resources\WP8_DFMS.properties

java -jar jar\WP8_DFMS_EvacuateSystem.jar

DFMS have a log of execution on standard output that coincide with DFMS_EvacuateSystem.log log file.

3.2.7.2 Shutdown

For shutting down DFMS we have two modalities:

1. Type Control-C directly on window where we have started up DFMS.

DFMS at start-up armed a Control-C interrupt in order to apply shutdown procedures and close all resources for all processes (clean shutdown).

2. Logging on web console of SOFIA2, o to function ontology->crude ontology and select eVACUATE_WP8SystemStatus ontology, after the browser display the data, change the value of status to SHUTDOWN.

DFMS SPV_SUPERVISOR_PROCESS is subscribed for any modification of eVACUATE_WP8SystemStatus, and when found the status set to SHUTDOWN, it start the procedures and close all resources for all processes (clean shutdown).

3.3 SMART SPACES

3.3.1 Instructions to develop and deploy a specific agent

This section describes the used versions, compilers or libraries needed to develop and deploy a specific agent. Furthermore, it offers the user a guide to compile and test its implemented agents.

The Agent_FRMK has been tested in two different Linux machines:

Linux distribution → Debian Squeeze	**Linux distribution → Ubuntu 12.04**
Kernel version → 3.4.4, amd64-bit	**Kernel version → 3.11.0-15-generic**

3.3.1.1 Compilation of the agent SDK

The SDK integrates:

- Encapsulation of communications with the SIB using SSAP protocol
- Encapsulation of communications with the eVACUATE GW using DDS protocol
- Integration of the datamodels and ontologies defined for eVACUATE agents
- All the results of the Agent_SDK compilation are copied to a directory with two subfolders
  - **Include** folder: where the .h files are located.
  - **Lib** folder: where the library is located.

3.3.1.1.1 Preparing the environment

Before compiling the Agent_SDK, the dependencies described in the following points must be satisfied. The user should precisely follow the steps in the order described in this document.

C++ compiler

The C++ compiler (g++) is not installed by default in most Linux distributions. To install g++, open a terminal and type either:

- **sudo apt-get install g++**
  or
- **yum install gcc-c++**

OpenDDS

The Agent_SDK uses OpenDDS as the base library for the DDS communications. Currently, the AGENT_SDK depends on the OpenDDS library version 3.5.1.

Download it from:

http://download.ociweb.com/OpenDDS/previous-releases/OpenDDS-3.5.1.tar.gz.

After uncompressing the OpenDDS library open a terminal and navigate to the OpenDDS folder. Execute **"./configure**" followed by "**make**". The compilation will take a while.

After the OpenDDS library has been compiled, the OpenDDS environment variables must be set:

- Open the script "**setenv.sh**" located in the root of the OpenDDS folder.
- Copy its contents to the file **".bashrc**" located in the home directory.
- If a development environment other than the terminal (such as Eclipse) is going to be used, copy the contents of "setenv.sh" to the file **".profile"** (also located in the home directory).

3.3.1.1.2 Compiling the Agent_SDK

Open a terminal and navigate to the Agent_SDK folder and execute "make". The compilation will take a while.

After the compilation finishes, the results are stored in the "build" folder. If desired, the AGENT_SDK can be installed in the system issuing the command "**make install**". The default path is **"/usr/local**". To modify the path use the macron "**DESTDIR**":

```
>> DESTDIR=path/to/installation/folder  make install
```

3.3.1.2 Development of a Specific Agent

3.3.1.2.1 .config File

Each specific agent must define its own configuration file in order to:

- Define different parameters needed to connect with the SIB
- Define different parameters for DDS
- To identify the connection
- To define the subscription to different events

The format of this configuration file is a csv file, where parameters and its value(s) are separated by ';'.

Here is an example of a configuration file for the ExitSigns agent:

```
server_address;www.sofia2.com

server_port;1883

client_id;"AppMobile_A"

token_id;e8beea58aa91496a85ea1bee965b5da2

kp_instance;TEKKP_TestSensor:TEKKP_TestSensor01

event_subscribe;eVACUATE_Alarm_onto

event_subscribe;eVACUATE_ExitSigns_command_onto4

dds_participant;exitsigns
```

Explanation of the .config File Parameters

Parameters for Producers and Consumers
• **server_address: the address where Sofia2 server is listening** • **server_port: the address where Sofia2 server is listening** • **client_id: identifier for client** • **token_id: identifier for token** • **kp_instance: identifier for KP instance**
Parameters for the SIB configuration

• **event_subscribe: the name of the event ontology that the specific agent wants to subscribe to. For example in the above case, the ExitSign agent has subscribed to two events**
  ○ **An Alarm Event and**
  ○ **A Command event**

Parameters for the DDS configuration
• **dds_participant: the name of the DDS participants the agent wants to subscribe to. The participant defines the topics the agent can produce and subscribe to.**
Comment Line
In a configuration file, the developer may include a comment lines that start with the ";" character

3.3.1.2.2 Main Application Source

.h File to include

- AgentFRMK.h
- DataModels\DataModel_List.h

NOTE: These files are in the subdirectory **include** of the Agent_SDK (described in chapter "Development of the Agent_SDK").

Define the complete path of the .config file

The application must define the complete path and name of the config file that will be used.

Example:

#define DEFAULT_CONFIG_PATH "./exitsigns.config"

Using AgentSDK API

The agent application must make use of the AgentSDK API to setup and manage the communication with the SIB and devices.

The normal sequence to use the API is:

- Define a callback function to process the asynchronous events produced upon the reception of new data.
- Call the AgentFRMK_Init : it must be the first call to the API in order to initialize the Agent_SDK
- Call the AgentFRMK_Start
- Loop
    - Create the appropriate structures that represent the desired datamodels
    - Call the AgentFRMK_SendToSIB() to send data to the SIB
    - Call the AgentFRMK_SendToDevice() to send data to devices
    - Concurrently, process the events received in the callback.
- Call the AgentFRMK_Stop

3.3.1.2.3 Properties of the project "Specific Agent"

If other than the default installation path for AGENT_FRMK is used, the AGENTFRMK_PATH environment variable must be defined to point to the right path. If a terminal is being used, just type:

```
>> AGENTFRMK_PATH= path/to/folder make
```

In Eclipse, in the option **Properties - C/C++ Build – Environment** there are two defined properties:

- CWD: this variable must point to the specific agent work directory
- PWD: this variable must point to the specific agent work directory

Add the AGENTFRMK_PATH variable.

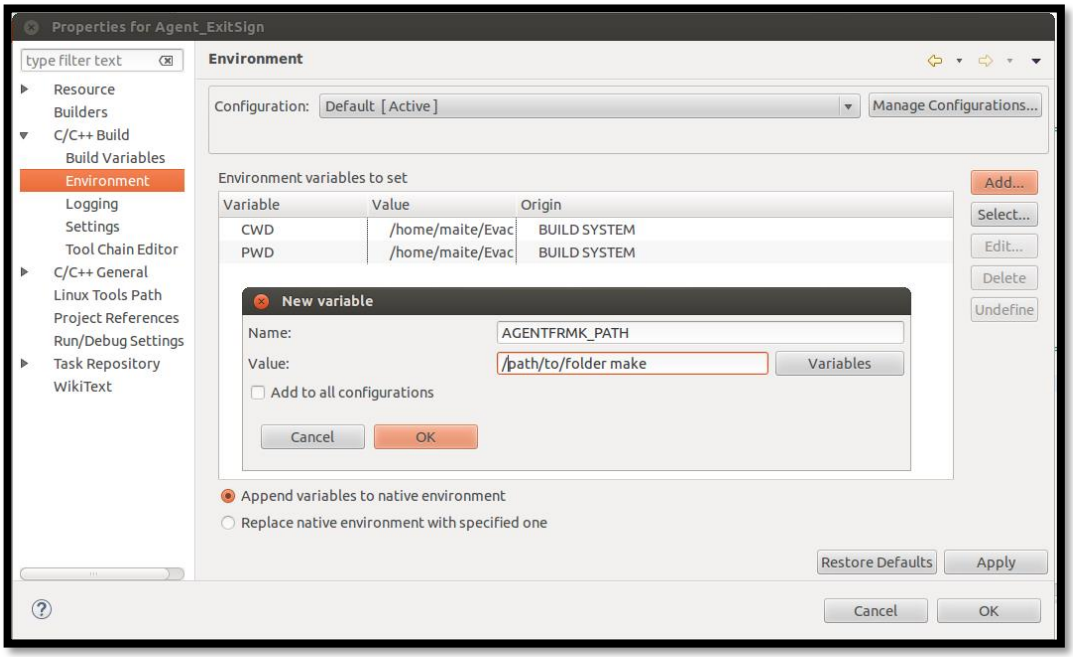


Figure 29: Adding New AGENTFRMK_PATH variable

3.3.1.2.4 Run the Specific Agent

In order to run the specific agent from Eclipse, a "**Run Configuration**" must be defined.

In the main menu **Run-Run Configurations** and below **C/C++Application**, the developer must define a new run configuration.

- Define a name for the execution configuration
- In the **Main** tab in the right window , the name of the executable program must be indicated
- In the **Environment** tab in the right window, the name of the property LB_LIBRARY_PATH. Its value must match with the subdirectory **lib** of the AgentSDK (described in chapter "Development of the Agent_SDK")

The following figures indicate how to define the configurations mentioned above.

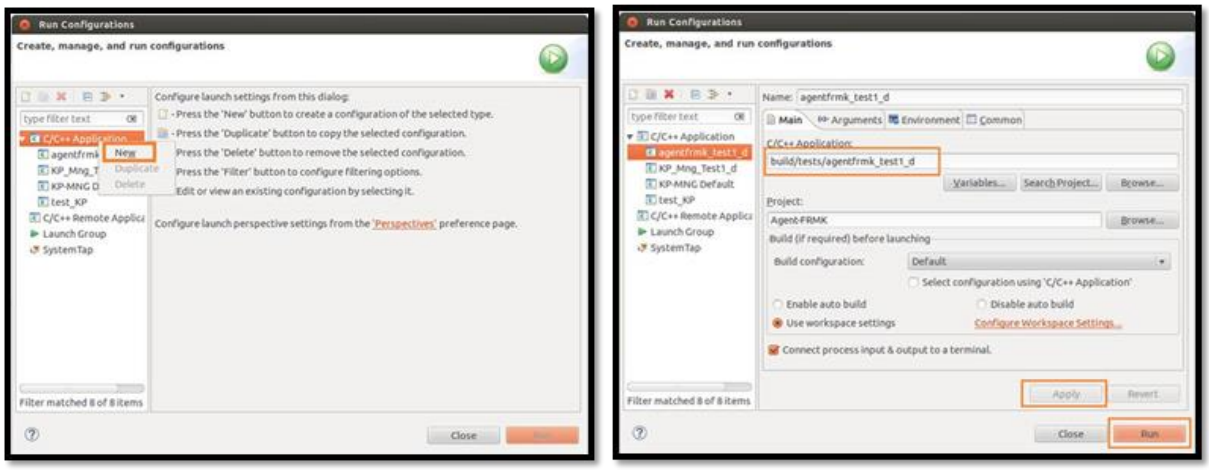
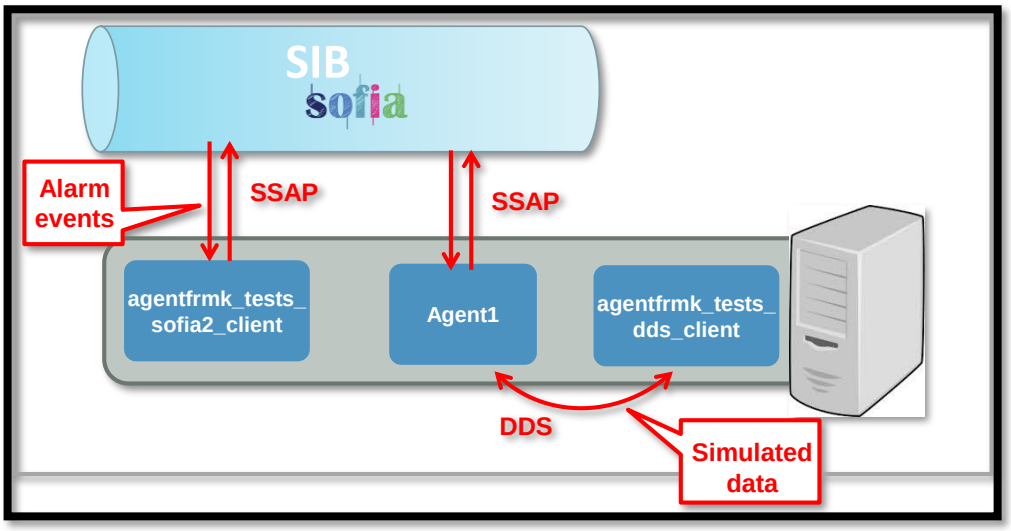


Figure 30: Definition of agents configurations

3.3.1.3 Testing the agents

In order to test the agents without the need of an eGW and a SIB client, two test applications are provided together with the AGENT_SDK: agentfrmk_tests_dds_client and agentfrmk_tests_sofia2_client.



As can be seen in the picture, agentfrmk_tests_dds_client simulates an EGW communicating through DDS while agentfrmk_tests_sofia2_client simulates a Sofia2 KP that feeds alarm events to the SIB.

Both test applications can be found inside the AGENT_SDK "**build/tests**" folder.

3.3.1.3.1 Agentfrmk_tests_sofia2_client

To execute the agentfrmk_tests_sofia2_client test application open a new terminal, go to the "agent_sdk_path/build/tests" folder and type:

```
>>> ./agentfrmk_tests_sofia2_client [-c config_file_path]
```

where "-c config_file_path" is an optional parameter to indicate where the configuration file is located. If no "config_file_path" option is provided the application will load the "agentfrmk_tests_sofia2_client.config" configuration file located in the same folder.

Once the application starts, it will show the following messages:

```
>>> ---------- SOFIA2_Test: Events generator and data consumer ---------
>>> SOFIA2_Test: Waiting for enter...
```

Each time enter is pressed a test alarm message will be generated. Currently, there are three predefined tests messages:

1. SMOKE alarm event in room3
2. FIRE alarm event in room1, room2 and room3

3. eVACUATE event in all the areas.

The agentfrmk_tests_sofia2_client test application will also receive any data updates to the SIB related to eVACUATE, thus, it also allows monitoring the data sent to the SIB by an agent.

3.3.1.3.2 Agentfrmk_tests_dds_client

To execute the agentfrmk_tests_dds_client test application open a new terminal, go to the "agent_sdk_path/build/tests" folder and type:

```
>>> ./agentfrmk_tests_dds_client [-h] [-d domain_id (>0)] [-c agent_class]
```

The "-d domain_id" option can be used to set the domain id for the DDS communication. It must be the same one used for the Agent. Default is '0'.

The "-c agent_class" option specifies the type of tests messages that will be generated and received. Each agent class tests the messages related to the data model of the agent type. The current supported class options are:

exitsigns	Data model for the Exit Signs agent
rfid	Data model for the RFID agent
crowbm	Data model for the crow behaviour agent
appmobile	Data model for the mobile application agent
envwsn	Data model for the environmental WSN agent
digitalsigns	Data model for the Digital Signs agent

The "-h" option prints the application command options as well as the list of available agent classes.

Once the application starts, it will show the following messages:

```
>>> ---------- DDS_Test Start---------
>>> SOFIA2_Test: Waiting for enter...
```

Each time enter is pressed a test message will be generated. The generated test messages will depend on the agent class selected.

The agentfrmk_tests_dds_client test application will also receive and print any data send by the agent (via DDS). However, only data messages related to the agent class data model will be received.

Exit Signs agent tests

Each time enter is pressed a test exist signs message will be generated. Currently, there are three predefined tests messages:

1. Device “EXIT025” current status to OFF
2. Device “EXIT043” current status to ON
3. Device “EXIT033” current status to OFF

RFID agent tests

Each time enter is pressed a test RFID message will be generated. Currently, there are three predefined tests messages:

1. Device “a” with codetag “A”
2. Device “c” with codetag “C”
3. Device “k” with codetag “K”

Crow behaviour agent tests

Each time enter is pressed a test crow behaviour message will be generated. Currently, there are three predefined tests messages:

1. Device “CB001” detected “CBM 4 people”
2. Device “CB025” detected “CBM no people”
3. Device “CB044” detected “CBM Jorge was here”

Mobile application agent tests

Each time enter is pressed a test mobile application message will be generated. Currently, there are three predefined tests messages:

1. Temperature: 20.1; Pressure: 1.013; Light: 3000; Humidity: 50.5
2. Temperature: 21.1; Pressure: 1.018; Light: 10000; Humidity: 34.7
3. Temperature: 21.5; Pressure: 1.014; Light: 7000; Humidity: 65.5

Environmental WSN agent tests

Each time enter is pressed a test environmental WSN message will be generated. Currently, there are three predefined tests messages:

1. Device “20” generate command “3” with data “TEST1”
2. Device “33” generate command “1” with data “TEST2”
3. Device “25” generate command “5” with data “TEST3”

Digital Signs agent tests

Each time enter is pressed a test digital sign message will be generated. Currently, there are three predefined tests messages:

1. Device with id “33” and name “DS001” at location “room1” is playing field “3” and status “ON”
2. Media file with id “22” and name “DSM003”, with type “FIRE” and status “ON”

3. Device with id “55” and name “DS034” at location “room2” is playing field “1” and status “OFF”

3.3.2 Deployment, integration and use of the different agents

3.3.2.1 Chipless RFID printed Tags

Figure 31***: Chipless RFID reader deployment*** shows the deployment of the RFID reader, it is composed of a set of two horn antennas which are mounted on a tripod, and connected to each of the radar transmission and reception ports with two coaxial cables. The radar communicates via USB to the laptop computer containing the MATLAB© graphic user interfaces (GUI) that control the radar calibration and normal operation procedures, to obtain the tags response and perform the detection and decoding afterwards.

The computer is connected to the evacuate network with an Ethernet cable, in order to transmit the detected tag information to the evacuate framework.

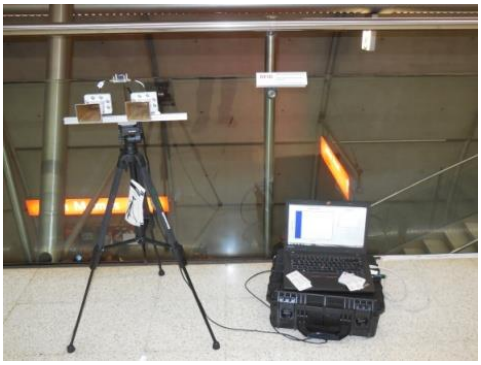


Figure 31: Chipless RFID reader deployment

The calibration screen is opened, and each tag is programmed individually. The tags are either placed in front of the antennas manually holding each of them with the hand or with the help of another support (tripod) as shown in Figure 32***Figure 32 Chipless tags a) calibration procedure, b) detection procedure***. After the tag is placed in front of the horn antennas, the “Load” button is pressed to record its response.

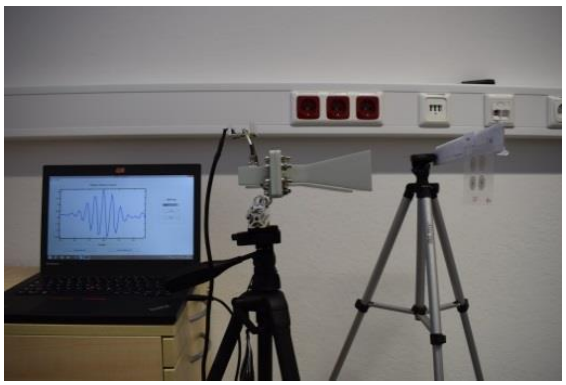


Figure 32 Chipless tags a) calibration procedure, b) detection procedure

Once the calibration procedure is completed, the system is ready to operate. The reading GUI interface is called directly from the calibration one, by pressing the button “Open Reader”. The button “Run” is pushed and the reader starts the scanning procedure, when a tag is placed in front of the antennas, the reader proceeds to its detection and decoding. After the decoding has taken place,

the computer sends the detected code to the ICCS server via internet protocol and the information is displayed in the COP.

The corresponding tag code counter is increased, its correlated image is displayed on the screen as shown in Fig. 2b, and a sound is played to alert the detection. The system waits until the UWB chipless RFID tag has left the identification zone and starts the process again.

3.3.2.2 Computer Vision

3.3.2.2.1 Overview

The Computer Vision (CV) module is based on multi-modal algorithms that process data from cameras working in the visible, thermal and hyperspectral spectrum. The processed data are used to assess the main features of the crowd and provides input to several sub-systems of eVACUATE through SOFIA platform which facilitates the decision support and optimum estimation of active evacuation routes. The CV algorithms will extract information about crowd density, velocity and crowd metrics (multi-scale features) defining usual and unusual behaviour in a given venue and operational context.

The system consists of the following components:

- Optical IP Cameras, (Brickcom OB 500AF,)
- 2 Thermal Imaging Cameras, (FLIR TAU2,)
- 2 Hyperspectral Imaging Cameras, (3D One,)
- One Small Form Factor (ARTiGO A1250,)
- One FPGA (3D One EP12,)

The main HW components are depicted in the following pictures:

Figure 33: Optical IP Camera

Figure 34: Thermal Imaging Camera & Small Form Factor

Figure 3: Hyperspectral Imaging Camera and FPGA

The communication among the CV Module and the communication gateway module will be based on TCP/IP protocol.

3.3.2.2.2 Interfaces, Connectivity & Topology

In order to identify the components of the computer vision module it has been split into two parts, CV Module (A) and CV Module (B) as shown in Figure 35. This is done since the two modules have developed and implemented different computer vision algorithms (more details can be found in the respective deliverables of WP3). In addition to that CV Module (A) will be connected to optical, thermal and hyperspectral cameras all installed on an elevating tripod next to each other at a fixed location at the venue and the same field of view. The CV Module (B) is fed by video streams either from the existing CCTV cameras at the end-users venues or alternatively by simulated CCTV video streams.

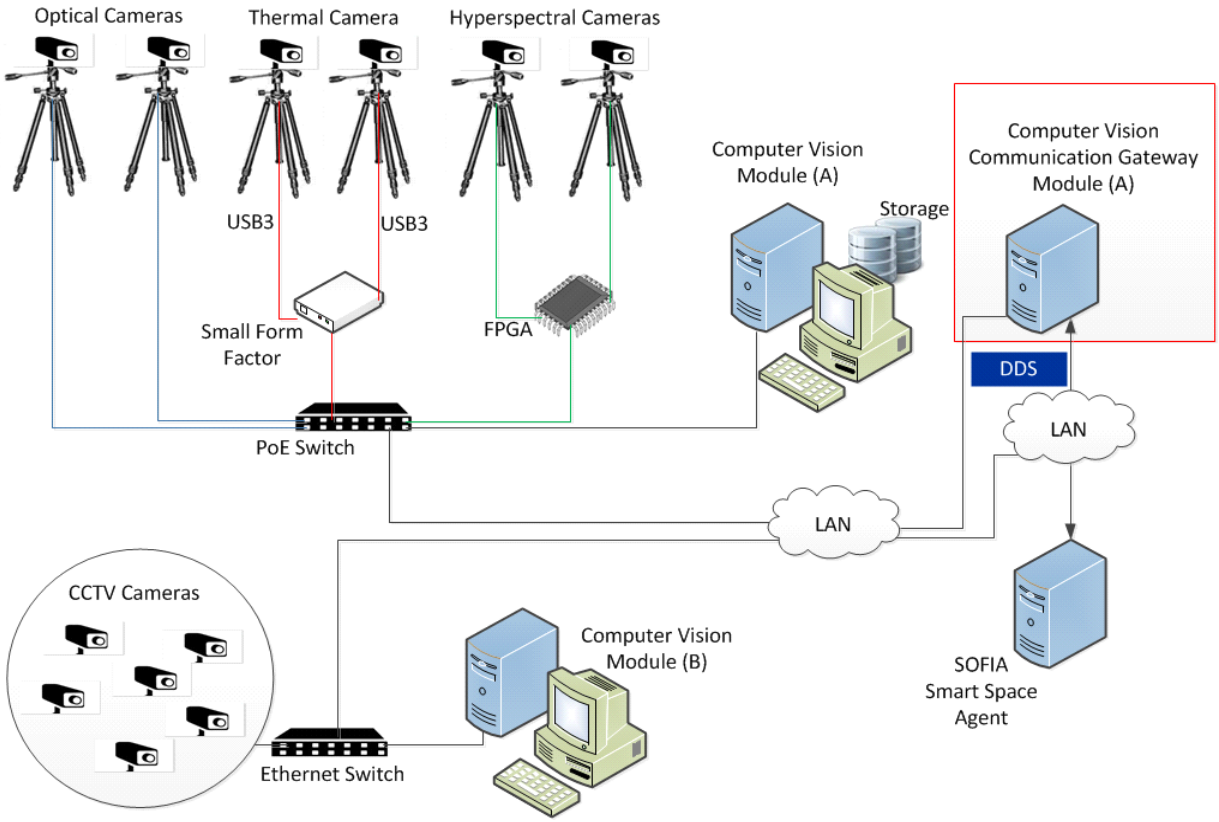


Figure 35: Computer Vision Module for crowd behaviour detection

The optical cameras are IP cameras and have an Ethernet interface so as to be connected directly on a PoE Switch. The Thermal Cameras are connected through USB3 interface to a small form factor (small embedded computer) that is used for pre-processing of the raw data. The Hyperspectral cameras are connected via Ethernet cable to an FPGA board that is used for pre-processing of the raw data. The host PC of CV Module (A) that runs its algorithms and fuses the data between the different camera sensors produces the CV output information to the CV Module (B). CV Module (B), that produces its own output based on its CV algorithms combines and fuses the output of CV module (A) so as to disseminate the combined output data, to the SOFIA platform. In order to enhance flexibility so as to adapt to different eVACUATE demonstration scenarios and venues the CV output data can be disseminated to the respective smart space agent through the communication gateway module. Both deployment options are depicted in the following figure.

The communication gateway module does not process and disseminate the video streams captured by the different cameras but only the extracted features. Nevertheless, the bandwidth requirements for the local network, where the local processing hosts are connected, are at the worst case 250 Mbps, and normally around 175 Mbps.

The output data of the CV module is passing transparently the Communication Gateway Module and communicates directly with the respective smart space agent of SOFIA.

3.3.2.2.3 CV module (A) setup

In order to setup the system a suitable location must be selected first that offers as much of a top down view as possible. Then a tripod with a base is installed to mount Optical, Thermal and Hyperspectral cameras. An example setup is shown in Figure 36 (ANOETA Stadium, San Sebastian, Spain).

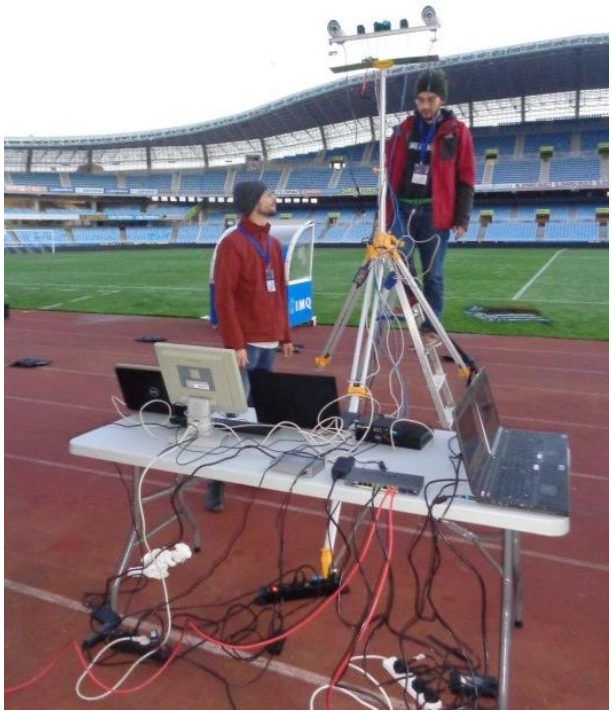


Figure 36: Example Setup of the CV module experimentations

In order for the system to be fully operational, a calibration process needs to be performed by an expert, for adjusting the suitable parameters for the scene geometry, illumination, etc. Then the CV module needs to be connected to the eVACUATE communication layer to SOFIA. This is established by running an adapter (Java):

- A deep learning algorithm (Python) is executed, and performing the detection of moving objects, background estimation, calculation of densities/flow/velocity and the creation of JSON files for messaging towards the SOFIA platform. This is applied for the thermal and hyperspectral vision generated data.

- An intelligent crowd behaviour detection module is also executed for messaging unusual behaviour towards the SOFIA platform. This intelligent crowd detection module operates in the

optical spectrum and has been trained and tested using vision data generated from the project experiments set up in the Anoeta stadium, Athens International Airport Satellite Terminal, Metro Bilbao stations and the STX cruise-ship in St Nazaire.

Further details about the full software development and implementation of the above mentioned crowd behaviour components which message key information on crowd behaviour and also physical motion characteristics to the SOFIA platform can be found in WP3 deliverable D3.7 version 2.0.

3.3.2.3 Dynamic exit signs

The dynamic exit signs system is composed by two different components. The gateway, shown in Figure 37 and the exit signs themselves (Figure 38).

Figure 37: Gateway

Figure 38: Wall exit sign

The gateway is the responsible of receiving the status of the signals and to send them the status the Agent receives from Sofia to allow the dynamic exit signs to show the proper evacuation route. The communication between the gateway and the exit signs is performed wirelessly (802.15.4 protocol) and the communication between the gateway and the agent can be done both by Ethernet or Serial interface.

3.3.2.4 Digital Signs

The Digital Signs consist of the following elements:

- TV or Monitor as output, need to support HDMI and it is not provided as part of the Digital Signs module.
- The Actual Digital Sign, which is based on an ARM microcomputer, example a raspberry pi or similar, running UNIX.
- Power supply for the Digital Sign. This is a normal power supply 220V to 5V, and it is based on a Greek socket.
- Network connection, either Lan or Wi-Fi is supported. Also the connection need to have access to the media server and to the EOC Server.
- Media Server, this is a Docker which contains the Media server where the files are stored and streamed. And another Docker with the Digital Sign API which communicate with the EOC GUI for manual control of the Digital Signs.

Installation of the two Dockers needs to take place before installing any Digital Signs. Once the two dockers are installed we need to know the IP of the media server.

Then for each Digital Sign perform the following steps:

1. Connect the Digital sign with the Output screen using the provided HDMI cable.
2. Connect a temporary Mouse and Keyboard for initial setup.
3. Insert into the Digital sign the SD card with the operating system correspond to the correct Digital Sign ID.
4. Connect the power adapter with the Digital Sign.
5. Plug in the Power adapter to the power socket.
6. Once the Digital sign finish loading check if the IP show on the Upper corner is the same as one from the installation of the media server docker file, if not change it.

The Installation is finished and the digital sign is ready to receive commands from the EOC GUI or the Sofia.

3.3.2.5 Wireless Sensor Network - Temperature, Humidity and Light sensors

The Environmental Sensor Network is based on ZigBee technology. ZigBee is an established set of specifications for low data rate wireless personal area networking, thus enabling digital radio connections between related devices located on the ZigBee radio Network and coordinated by the ZigBee main device called Gateway. The Gateway (GW) as well as functions of coordinator of all

ZigBee network devices also provide to send the collected environmental data to SOFIA platform as part of eVACUATE system. In the figure below is shown the typical ZigBee Environmental Wireless Sensor Network possible topologies.

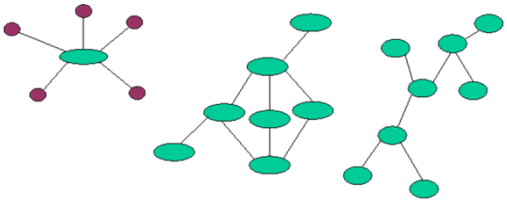


Figure 39. ZigBee Environmental Wireless Sensor Network topology

In the eVACUATE system

Temp/humidity sensor

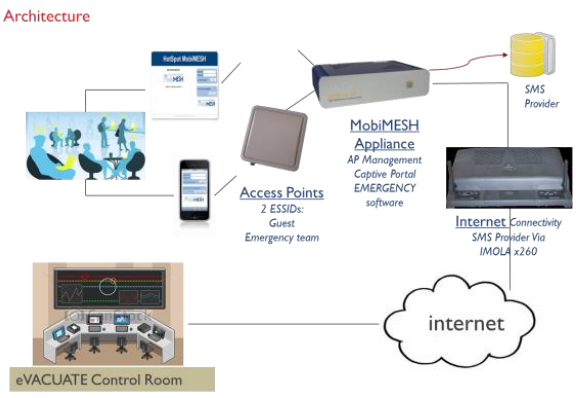
Gateway node

Router/repeater

3.3.2.6 Information and services from Network Operator (MobiMESH)

Requirements: Wi-Fi system with emergency management functions

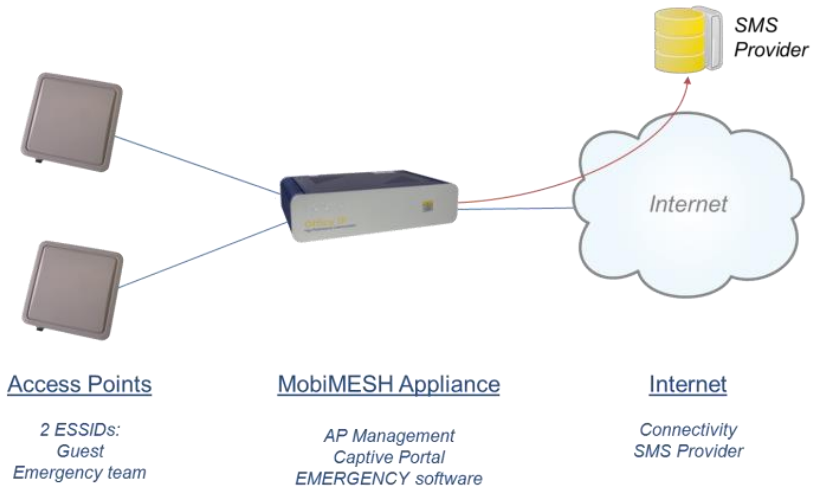
- Wi-Fi authentication
- User identification and localization (AP)
- «Emergency mode»
  - SMS notification
  - Traffic inibition for Guests
  - Control panel



Indicative information includes the load of the “base stations” (Wi-Fi AP in this case), the number (quantity) and the location of attached phones. We do not require individual subscriber data, those are excluded from our search due to privacy reasons.

The system should expose all the requested information to 3rd party systems via the API provided. However the WEB GUI proposed could be useful for testing or visualization purposes. Traffic prioritization should be included as a requirement. It should be prioritized the “Emergency team” traffic during an emergency, but the available bandwidth should be provided to the “guests” or citizens, not inhibited. We need to have a mechanism in place, so that the first responders and the M2M Wi-Fi devices bypass the authentication process. From our discussion it seems that SMS is the main message type used for asynchronous messaging by MobiMESH.

Pilot Architecture:



This means that a mobile connection is needed to use the system. However in many cases there is no mobile network coverage (metro stations, cruise ships) or there are no mobile data (stadium) due to high user density and congestion. In such cases Wi-Fi networks are used for connectivity.

Broadcast messages to predefined groups: this means that we would like to send a message to the “Emergency team”, or to the citizens that are attached to an specific AP, or to all the citizens … Always predefined groups of people.

3.3.2.7 Social Networks Monitoring Platform

The Social Networks Manager (SNM) Platform allows finding and evaluating the conversations that take place in the Social Networks and are related with the pilot, and also to provide statistics. The technology goal is to detect and measure the occurrence of a specific term (Topic Detection and Tracking).

The practical end goal however is to offer alerts based on events of statistical importance, i.e. in the case that there are more mentions/posts/tweets than average regarding a specific venue (e.g.

Anoeta Stadium, Metro Bilbao) or mentions about fire or floods in the area of proximity of one of the above events or facilities. This could indicate there is something going on.

Although such alerts will be handled by the agents of the eVACUATE/SOFIA architecture, an additional functionality is offered in a dashboard that allows for a better review of the situation unfolding.

3.3.2.7.1 Intergration

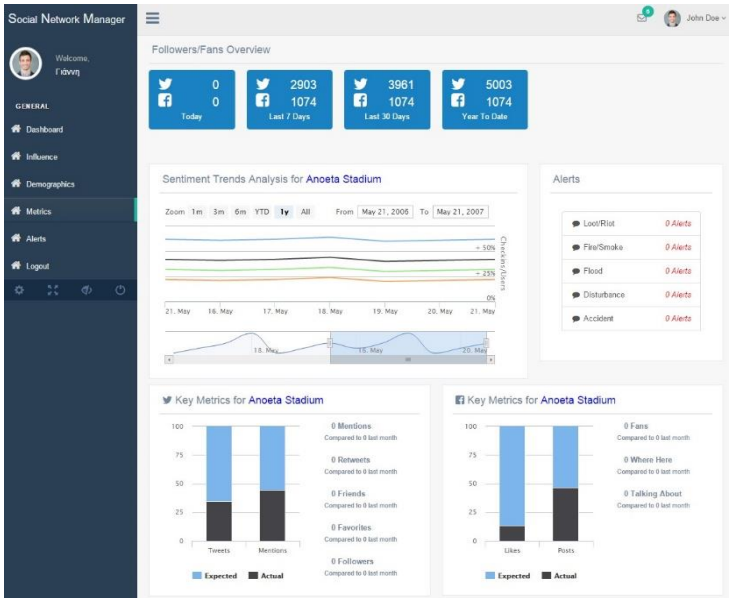
The system has a Rest API that it is integrated with the EOC platform.

3.3.2.7.2 Functionality

The social service runs continuously with pre-configured keywords and hashtags. The service auto fetches all the new tweets within a short time interval (2 minutes) and populate the data to a database. Then the API calls fetch these data and makes the logical condition check for alerts.

3.3.2.7.3 Dashboard

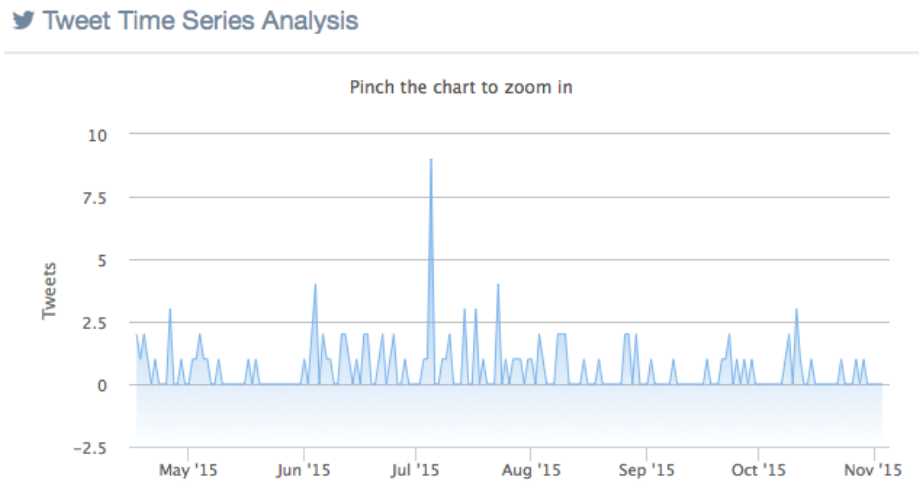
The main page is what we call the dashboard. Allows the user to select a term, in this case the term is “Anoeta Stadium” a term that represents one of the users of eVACUATE system.



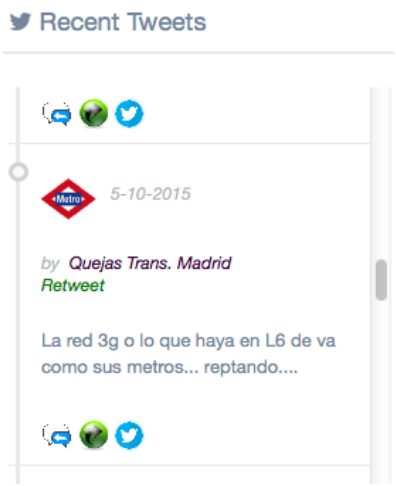
For the selected term, we are offered an immediate overview of the key figures and statistics. The higher row contains the key figures for (a) today (b) last 7 days (c) last 30 days (d) up to one year ago. The selection of which key figures will be presented is totally arbitrary and responds to the specific user’s needs and interests. In the example below, the number of Facebook friends and Twitter followers is presented.

Several benchmarks may be offered as part of the home page, dashboard.

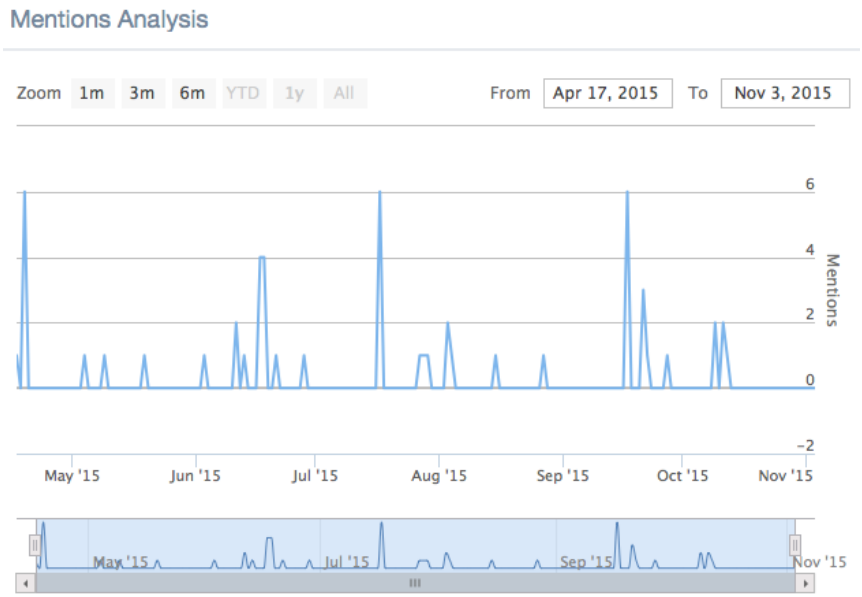
Tweet Time Series Analysis: Indicates which day the most tweets appeared about a term or a page. A timescale selection tool allows the user to zoom in and out of a selected time period. This function allows the operator of the system to **understand if a burst of tweets is abnormal and may indeed represent a threat**, or it could be a periodic increase (e.g. a burst of reports about the Anoeta Stadium on a Sunday is expected given it is a match day).



Recent Tweets: Offers the most recent tweets about the selected term, **this allows to further investigate the reported reason for a burst in tweets frequency**.



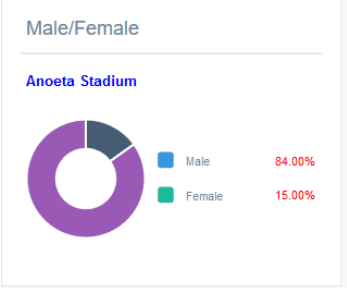
Mentions Analysis Time Series: A daily “mentions” timeseries.

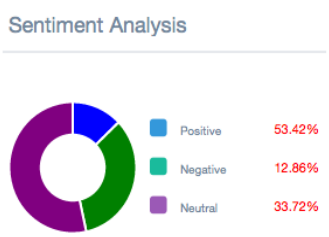


Male/female ratio: Has a complementary value.

Sentiment analysis: Offers an approximate estimate of the public’s perception of a term. The outcomes are “Positive”, “Negative”, “Neutral”.

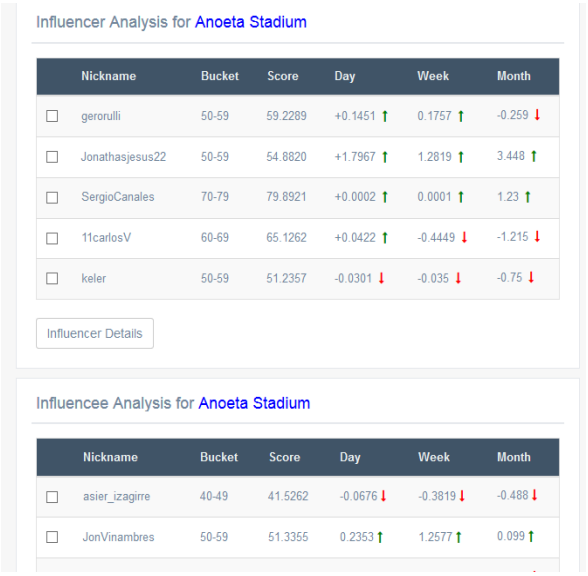
Tweets by country: Shows from which country of origin are the most authors mentioning a term.





Influence analysis: The “influencee” analysis indicates the influence the term has to end users. The top influencees will retweet and thus influence other users in their network. The “influencers” table indicates users with the most influence according to their Klout score. Daily, weekly and monthly variation is also shown

Visual aid for interpreting the Klout score of a certain influencee. By clicking on the "Score" link a daily, weekly and monthly horizontal bar graph is displayed.



The influencer analysis may be used to assess the **credibility of a post**, something especially useful in case that many post/tweets occur about a venue.

Finally, in the **alerts page**, we can set notifications if an event exceeds the threshold we have set and may represent a reason of concern. To this end the user may set specific thresholds for several terms, which when mentioned could require the attention of the safety operator.

3.3.2.8 TETRA Messaging

As show in Figure 40, TETRA deployment consists of:

- Motorola Handheld Terminals (MTH650)
- Motorola Desktop Terminals (MTM700) and their attached external antenna.

Figure 40: TETRA Radio Devices

The integration of the TETRA SDS functionality with the eVACUATE system is shown in Figure 41.

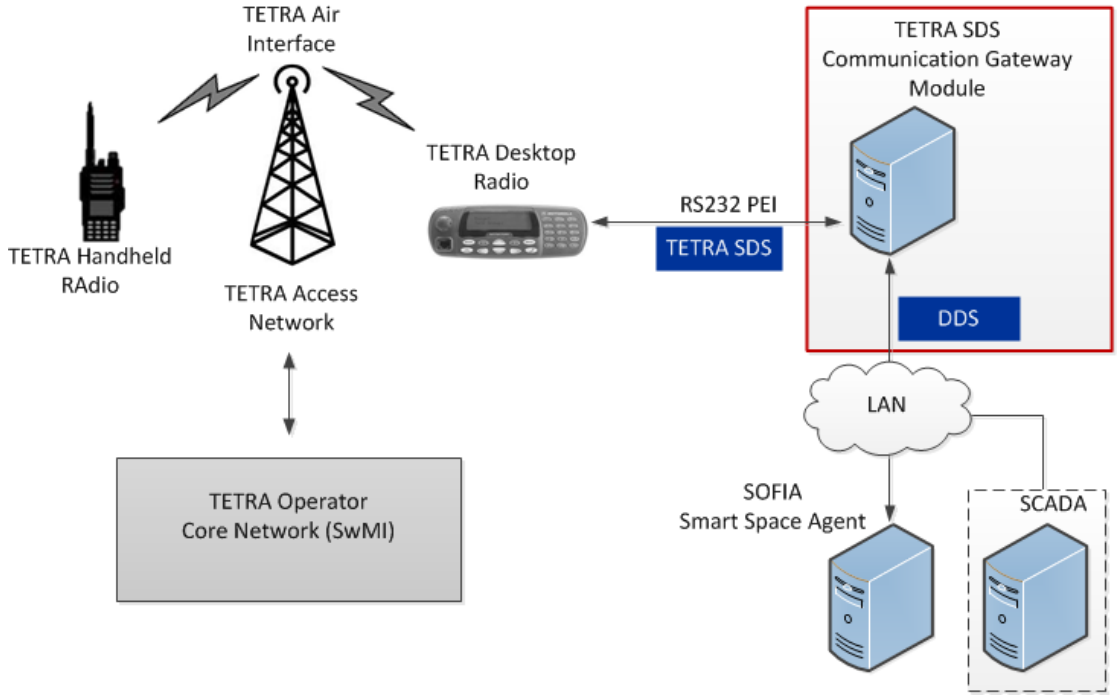


Figure 41: TETRA SDS Communication Gateway Module

Each TETRA radio device has a TETRA RF interface for communicating with other TETRA devices registered in a specific TETRA network over the TETRA air interface. The users of Short Data Transport Service (SDTS) can also access the service at the Peripheral Equipment Interface (PEI) in the Mobile Station (MS). The TETRA PEI provides a link between a data terminal such as a Personal Computer (PC) or specialized data terminal and a TETRA mobile terminal.

The TETRA radio devices, either handheld or desktop (mobile station), must be registered with the operator's network with a valid Individual Short Subscriber Identity (ISSI) and the SDS messaging services must be activated on the devices in order to be able to send and receive SDS messages.

As shown in Figure 41, the main interface of the communication gateway module with TETRA network is the RS232 connection of the mobile station with the PC. Furthermore, for our implementation we choose SDS Type 4, due to the fact that it allows messages of variable length to be sent with a maximum of 2017 bits, and because this is the only SDS service available for the respective terminals for sending messages over the Peripheral Equipment Interface (PEI).

The RS232 connection to the mobile station is a 5-wire connection with TXD, RXD, DTR, CTS and RTS. The TXD and RXD are used to carry the data between the MS and an MS external application. The DTR and CTS are used for flow control between the MS and the external application. The connector details of the RS232 cable are illustrated in Figure 42.

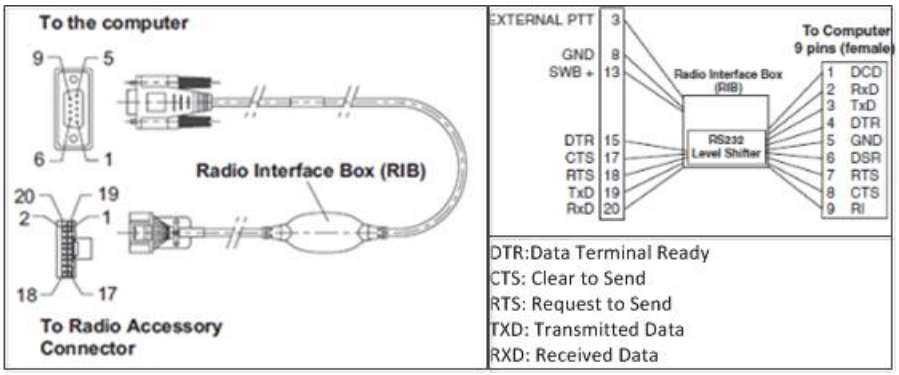


Figure 42: Connector Details of RS232 Cable

The default settings for the RS232 communication between the mobile station and the TETRA SDS Communication Gateway module are:

- Baud rate (line speed): 9600 bps
- Data bits: 8
- Stop bits: 1
- Parity: None
- Alphabet used is ITU_T T.50
- Carriage return value: 13
- Linefeed value: 10
- End of file value: 26
- Escape value: 27
- Space value: 32
- Backspace value: 8
- No echo
- Result codes on
- <CR><LF> used
- Numeric extended error result codes
- TETRA Protocol Data unit (PDU) mode

It must be noted that the mobile station doesn't support changing these values and are mentioned here in order to avoid ambiguity.

The messages received by TETRA desktop device can be forwarded to the TETRA SDS Communication Gateway module, using the PEI. Then the TETRA SDS Communication Gateway module is responsible to forward the received message by the TETRA desktop device to the SOFIA Smart Space agent. If a message for a handheld device arrives to the TETRA SDS Communication Gateway module, it is forwarded to the TETRA desktop device in order to be send to its final destination.

3.3.2.9 Internal wireless DECT & Fixed phones and Smoke/Fire detection

The STX DECT and Fixed Phone system is a system that manages and controls phone calls and emergency messages within the ship, both for the passengers as well as for the crew members. The interfaces, connectivity and topology of the STX Phone System and its integration with the respective communication gateway module is illustrated in Figure 43. The communication gateway module is able to receive alert messages and disseminate them through the phone system to fixed and wireless DECT phones.

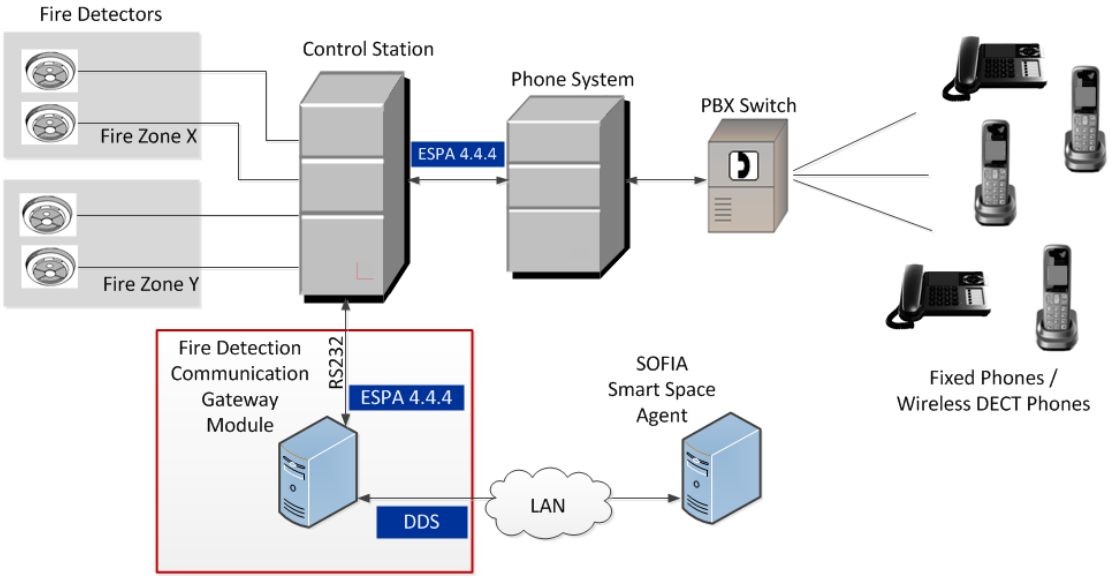


Figure 43: STX DECT & Fixed Phone Communication Gateway Module

The data flow based on ESPA 4.4.4 between the DECT/Fixed Phones System (DFPS) and DFPS Adapter is described in the message sequence chart in Figure 44. The DFPS is the control station. During the polling phase DFPS sends an ENQ with the address of DFPS adapter. The DFPS adapter becomes temporary master because it has a message packet to send. In select phase, DFPS adapter sends an ENQ with the address of DFPS. The DFPS is available, responds with an acknowledge (ACK) and becomes slave. In transfer phase, DFPS adapter sends the message packet and DFPS acknowledges it. For terminating the communication, DFPS adapter sends an end of transmission (EOT).

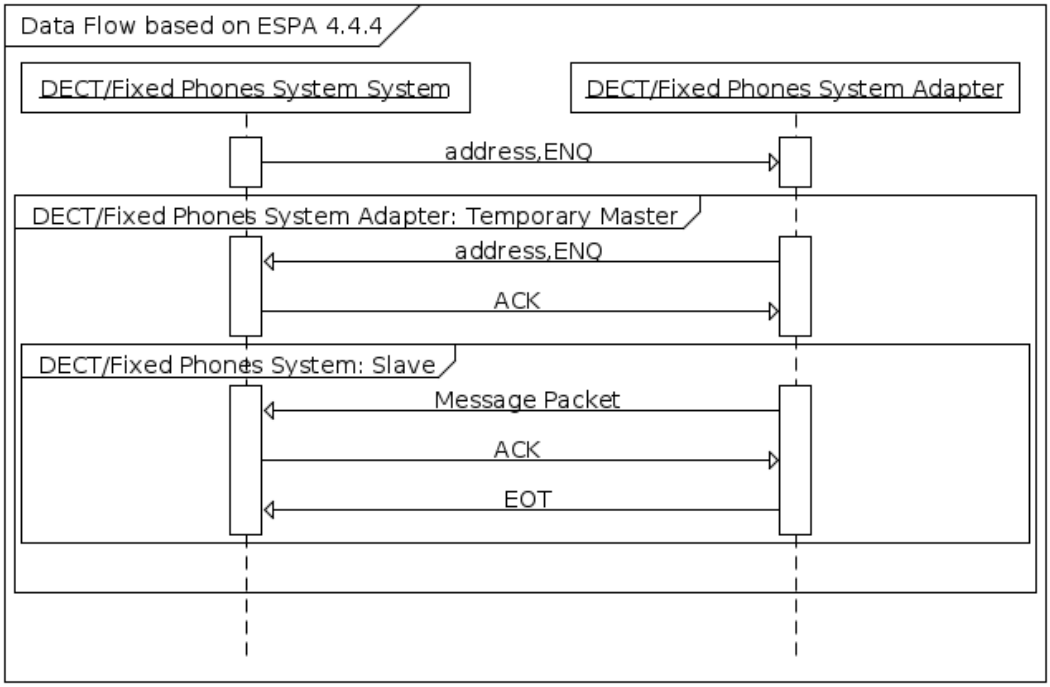


Figure 44: Data Flow between DFPS and DFPS Adapter based on ESPA 4.4.4

The DPFS adapter is responsible to forward the received messages by the DPFS to the SOFIA Smart Space agent and vice versa.

3.3.2.10 Ticket Validating Machines

3.3.2.10.1 Components

Two software components need to be installed. The components are described below:

- Manual Control. This is a .Net application please install it on a windows Machine. Once its installed then navigate to the installation folder and modify the config.xml file. On the section where it says IP please change the 0.0.0.0 to the ticket machine ip provided by the network administrator.
- Ticket machine server. This is a .Net application please install it on windows Machine. Once its installed then navigate to the installation folder and modify the config.xml file. On the section where it says IP please change the 0.0.0.0 to the ticket machine ip provided by the network administrator.

3.3.2.10.2 Installation

No further installation is needed expect to connect the server or PC where the above software is installed to the network that have access to the Ticket Machine ip.

3.3.2.11 Public announcement

3.3.2.11.1 Components

Two software components need to be installed. The components are described below:

- Manual Control. This is a .Net application please install it on a windows Machine. Once its installed then navigate to the installation folder and modify the config.xml file. On the section where it says IP please change the 0.0.0.0 to the pa ip provided by the network administrator.
- PA server. This is a .Net application please install it on windows Machine. Once its installed then navigate to the installation folder and modify the config.xml file. On the section where it says IP please change the 0.0.0.0 to the pa ip provided by the network administrator.

3.3.2.11.2 Installation

Download and install the London Architect latest version. Run the application from the icon on the desktop. Open the London architect file with PA-Evacuate name provided by the evacuate project. Once it is open, navigate to devices and locate the PA device, click connect and then upload the London Architect file inside the PA device.

No further installation is needed expect to connect the server or PC where the above software is installed to the network that have access to the PA ip.

3.4 COMMUNICATION LAYER

The communication gateway acts as a plug-and-play sensor/device abstraction layer, which hides the communication details and heterogeneous hardware of the sensors/devices allowing applications or services, which connect to the communication gateway to collect data or be able to task / configure the sensor/devices if such functionality is provided by the sensor/devices.

The eVACUATE communication gateway modules, which were implemented and demonstrated in the pilots are the following:

- Dynamic Exit Signs Module
- Digital Signs Module
- Environmental Sensors – WSN Module
- Chipless RFID System Module
- Existing Legacy Systems:
  - AIA - Building Management System
  - AIA - TETRA
  - STX – Fire Detection System
  - STX – DECT and Fixed Phones
  - METB – TETRA

There are two different ways that the distributed capabilities of the communication gateway can be taken into advantage. The first option is to split and deploy the developed adapter in two physical servers. An example of this approach is shown in Figure 45.

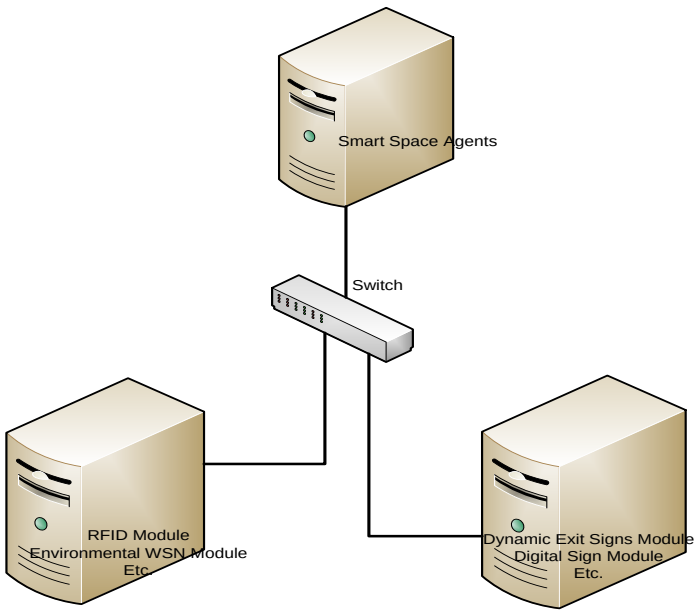


Figure 45: Splitting Communication Gateway's modules in two physical servers

The second option is to use one physical server as hosts for virtual machines (VMs), using the virtualization capabilities of modern CPUs, and one physical backup/mirror server for seamless failover in case of a malfunction in the first server as shown in Figure 46.

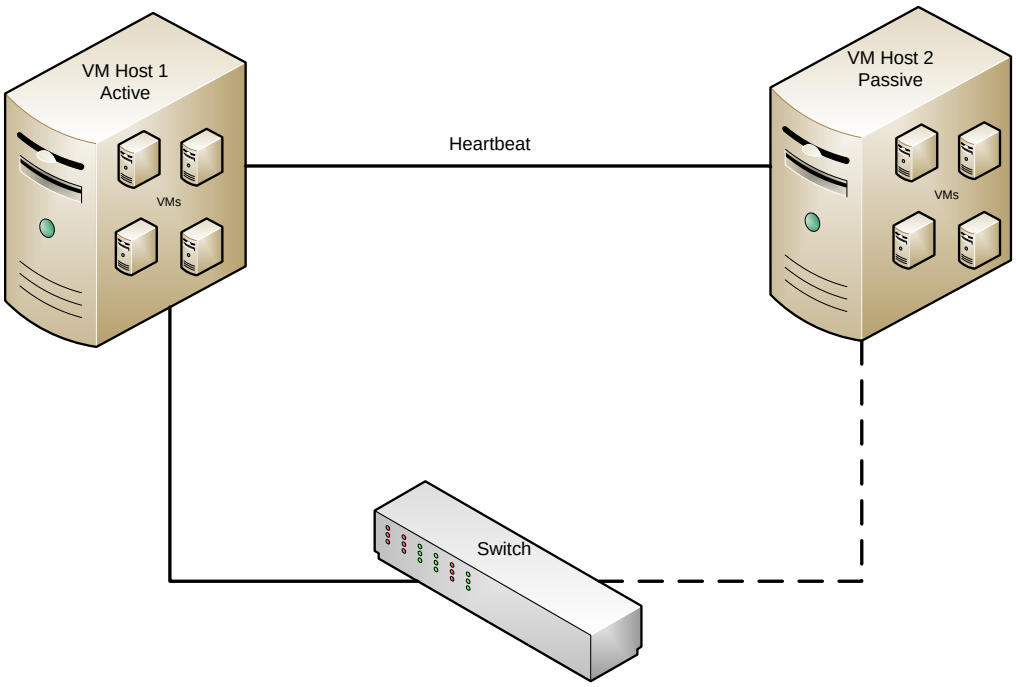


Figure 46: Seamless failover and failback

The communication gateway adapters are executed as services on the background and do not have a user interface, so no action from the user is necessary during their execution. Each communication gateway adapter can be configured and adjusted in order to serve the needs of the specific venue. For this purpose we have incorporated in a configuration file in JSON format, which could be easily accessed and modified using a simple text editor. For example the configuration file for the active exit signs adapter contains the following parameters:

- DDS_DOMAIN: The DDS domain. For symbolic reasons we have chosen the number 313161 (the ID number of eVACUATE project).
- TEK_PREFIX: The prefix to be used in the identifier of the communicated data. In case of active exit signs adapter the prefix is “tek-aes-".
- FEATURE_OF_INTEREST: Self-explanatory. In case of Athens International Airport the value of this field is "Eleftherios Venizelos".
- NETWORK_ADDRESS: The network address of the TEK DES Gateway.
- PORT: The port that the TEK DES Gateway is listening to.
- CYCLE_PERIOD: How often the devices communicate with TEK DES Gateway (in milliseconds).

- CHANNEL_LIST_LENGTH : How many channels are used by the dynamic exit sign devices.
- CHANNEL_LIST: The channels used by the dynamic exit sign devices connected to TEK DES Gateway.
- TIME_FORMAT: The time format used in the representation of time/date elements.
- LOGGING_LEVEL: How detailed the adapter logging would be.

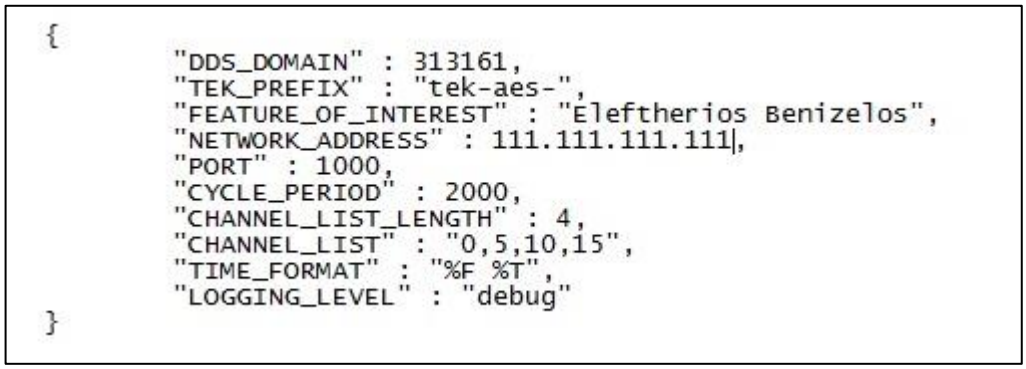


Figure 47: Sample configuration file of Dynamic Exit Signs module

In case of a crisis situation that requires immediate actions and human interventions, the Operator/PPDR gateway enables the operators or the PPDR commanders to take over control of communication gateway. The actuating elements and devices of the eVACUATE system can be controlled directly by the operator or PPDR user, irrespectively of the automatic alerts, instructions and commands provided by the eVACUATE system.

This important feature of the communication gateway is achieved by assigning priority to the operator's/PPDR user's crisis commands in case of an emergency situation. The distinction between PPDR and operator gateway is in terms of operational hierarchy. This operational hierarchy will be embedded in the operator/PPDR gateway functionality.

In more detail the Operator/PPDR gateway has the control over the following devices:

- Dynamic Exit Signs
- Digital Signs
- Phone System (for STX)
- TETRA

The above described feature will be accessible by the operator or PPDR user via the Emergency Operation Centre (EOC) module that deploys the relevant graphical user interfaces. The functional diagram of the operator/PPDR gateway is illustrated in Figure 48.

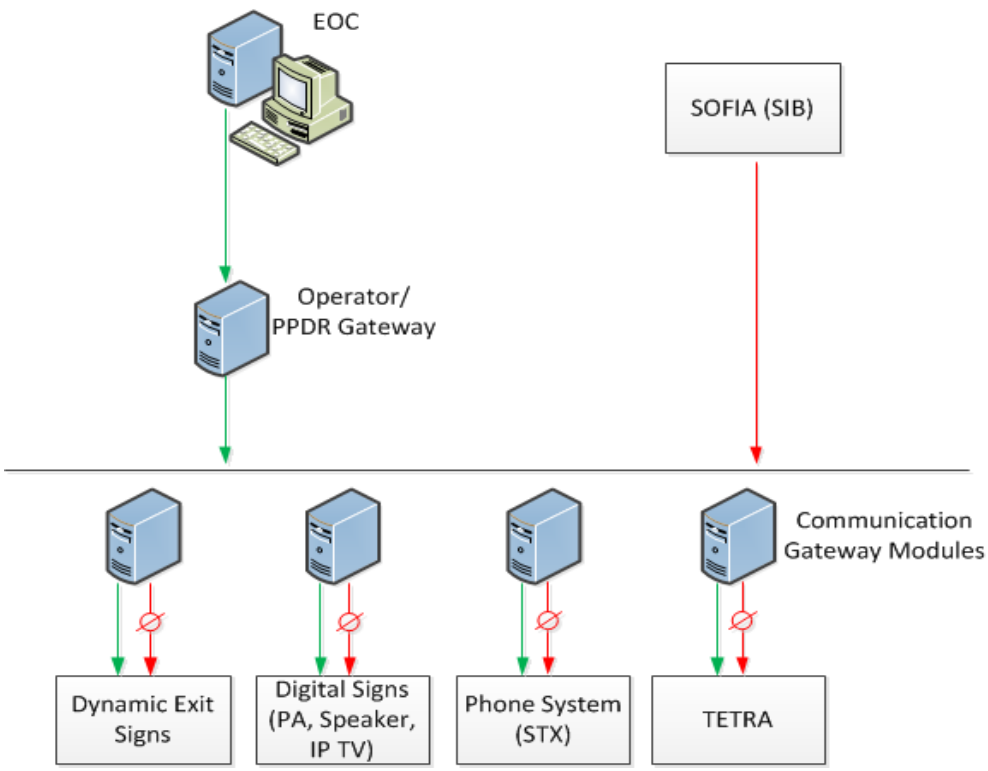


Figure 48: Operator/PPDR Gateway

The operator gateway does not communicate with the devices directly. As shown in Figure 48, the operator gateway is responsible for publishing the received information to DDS and then the corresponding communication gateway module handles the communication with the devices.

The tables below describe the SW Application Programming Interface (API) API for accessing the functionalities of the operator gateway. We adopted a RESTful architectural style, which make the API very flexible, and easily extendable. Json format is used in both requests and responses. The tables below describe the SW Application Programming Interface (API) for accessing the functionalities of the operator gateway.

TETRA Custom Message		
Description	Sends a message to a specific TETRA radio device.	
URL	…/OperatorGateway/api/TETRA_MSG/<ID>	
Method	POST	
Fields	destination	The Individual Short Subscriber Identity (ISSI) of the destination TETRA radio device. It is the same as the <ID>.
	message	The message to be sent.
Returns	200 OK & JSON	The operation succeeded.
	400 Bad Request	Error(s) in parameters.

Table 1: Operator Gateway - TETRA Custom Message

DYNAMIC EXIT SIGNS		
Description	Changes the status of a specific Dynamic Exit Sign (DES).	
URL	…/OperatorGateway/api/DES/<ID>	
Method	POST	
Fields	device	The id of the dynamic exit sign.
	status	The new status that the dynamic exit sign must take. The status can be a string (OFF, BLINK etc) as presented in enumeration DM_ExitSigns_STATUS_e.
Returns	200 OK & JSON	The operation succeeded.
	400 Bad Request	Error(s) in parameters.

Table 2: Operator Gateway - Dynamic Exit Signs

DIGITAL SIGNS		
Description	Changes the status and/or the reproduced media content of a specific Digital Sign.	
URL	…/OperatorGateway/api/DigitalSign/<ID>	
Method	POST	
Fields	device	The id of the digital sign that the status and/or the media content must be altered.
	actionType	The type of action. It can take the values PLAY or STOP
	mediaContentID	The id of the media content that the digital sign must reproduce. Only applicable in case that the actionType is PLAY.
Returns	200 OK & JSON	The operation succeeded.
	400 Bad Request	Error(s) in parameters.

Table 3: Operator Gateway - Digital Signs

STX DECT and FIXED PHONES		
Description	Sends a message to a specific DECT or fixed phone.	
URL	…/OperatorGateway/api/STX_PHONE/<ID>	
Method	POST	
Fields	destination	The id (phone number, group number etc) of the destination. The destination can be either a DECT phone, a fixed phone or a group of phones.
	message	The message to be sent.
Returns	200 OK & JSON	The operation succeeded.
	400 Bad Request	Error(s) in parameters.

Table 4: Operator Gateway - STX DECT and fixed phones

In order to control and manipulate which system the devices are listening to a special mechanism was developed that implements this functionality. EOC can access this functionality by using the following request.

CONTROL		
Description	Defines which system(s) the devices must listen to.	
URL	…/OperatorGateway/api/CONTROL	
Method	POST	
Fields	systems	A string that specifies the system(s) that the devices must listen to for this point of time and on (until further notice). It can take as values OG_SOFIA (listen to both SOFIA and Operator Gateway or OG (listen only to Operator Gateway
Returns	200 OK & JSON	The operation succeeded.
	400 Bad Request	Error(s) in parameters.

Table 5: Operator Gateway - Systems Control

3.5 EVACUATION ROUTES

The evacuation routes and crowd congestion predictions are calculated using a fixed geometry model that should be set up by an expert in crowd simulation. The End User controls the parameters of the simulation via the COP, explained in section 2.3 of this document.

Internally, the expert consultant will build a model for a specific venue using the 'Visualiser' Component, shown below:

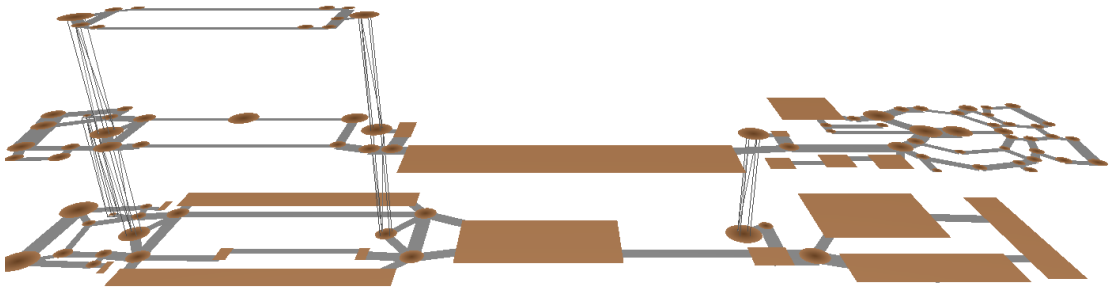


Figure 49: Visualising a crowd model in 3d, using the 'visualiser' GUI

Basic navigation of the GUI uses the mouse. The current mouse position, zoom level, edit mode and rotation is shown in the status bar as shown below:

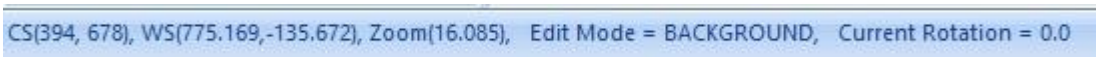


Figure 50: Status bar

In this example, the mouse is at Client Space position (394, 678) (394 pixels from the left side of the screen and 678 from the top). This represents World Space coordinates (775.169, -135.672) - 775.169 meters to the right and 135.672 meters below the origin point. The zoom level is set to 16.085, which means that every 16.085 pixels represents 1 meter.

There are 2 types of node; circular and square. To create a circular node the user should select "Nodes" from the "Mode Select" box shown in Figure 49. To create a square node, "Square Nodes" should be selected.

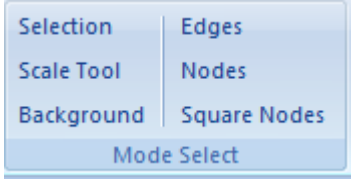


Figure 51: Mode Select

3.5.1 Nodes and Links

Once selected, in the main window, click with the left mouse button and drag to create the selected node type.

An example of the node property widget is depicted in Figure 52.

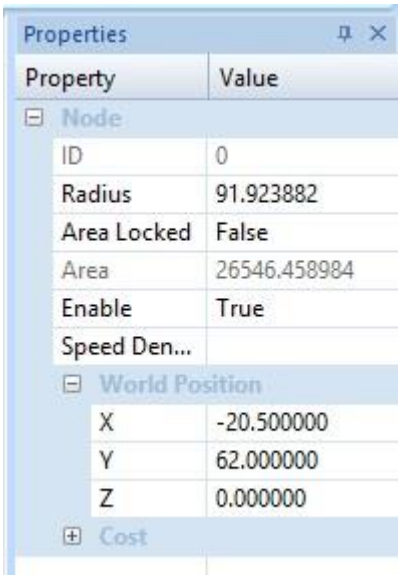


Figure 52: Example of a Circular Node's Properties

The properties of a circular node are as follows:

- The ID is the programs way of differentiating between the nodes and does not need to be edited. By default, the ID will also be displayed in the main window, both in 3D and 2D modes.
- The Radius is the distance from the centre of the Node to the edge.
- The area is automatically calculated from the radius and cannot be edited, unless the "Area Locked" parameter is changed to true,
- If Area locked is true, any value can be entered in the area box and it will no longer be updated by changing the radius.
- The Enable flag is for enabling and disabling certain nodes which may be a required part of any evacuation plan for preventing people entering dangerous areas such as a room which is on fire or flooded.
- The Speed Density parameter is for changing which Speed-density curve agents should use when they are travelling through this particular node (if it is not the standard set up in the xml file. This is explained later).
- The world position gives the local position of this node within the network which gives the number of meters a node is away from the origin (which is the centre of the screen unless the screen has been panned). The Z parameter will be zero for this first node, but this can be changed to represent a different height to the height which the origin represents (normally ground level).

Square nodes are created in a different editing mode, in a similar way to circular nodes, but with additional properties as follows:

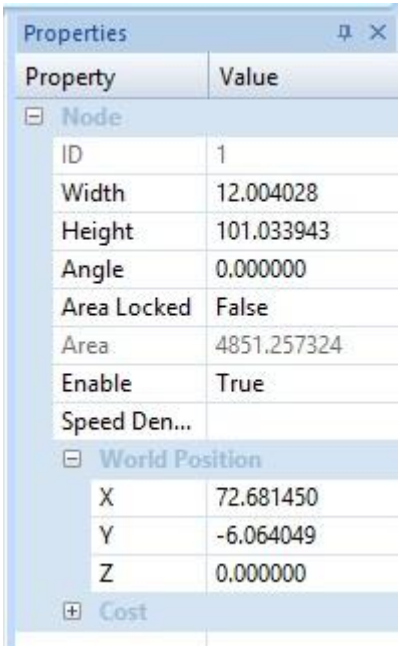


Figure 53: Example of a Square Node's Properties

Links are created in the network in edge edit mode (edge and link are used interchangeably), where a link is drawn between two nodes and then widened by dragging the edges or altering the properties.

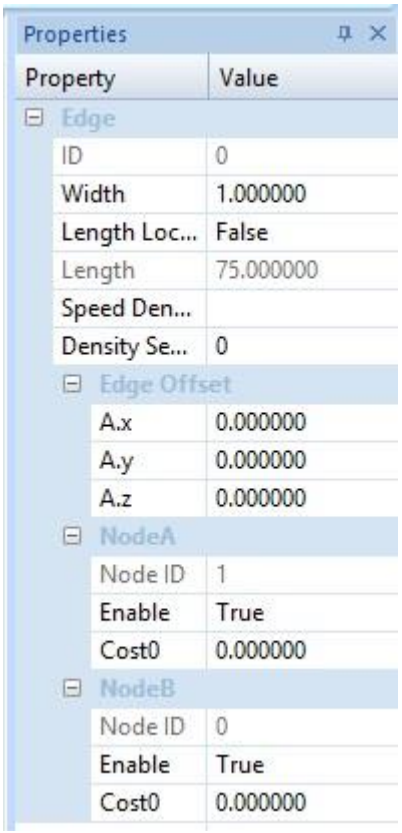


Figure 54: Example of Link Properties

The ID parameter is much the same as the parameter for Nodes and does not need to be edited. The width represents the effective width of the corridor in the network and always starts at a default of 1.0 metres. The length of the link is automatically calculated much as the area was for nodes, with the optional parameter "Length Locked" allowing you to specify a different length to represent a non-straight corridor, such as U-bend stairs. The length represents the distance between the centre points of the two nodes as the crow flies so it is not the actual length of the corridor. This is all accounted for in the movement algorithms, but it an important part of the modelling methodology that is necessary, but unusual. The Speed Density parameter is the same as for Nodes, but the Density Sections is a parameter which represents how may sections to divide the link into when outputting density. If this is set to zero then it will use the defaults in the network settings file.

The whole network can be built by drawing upon an imported image file that can be scaled and rotated as appropriate. The image files can be any image format including jpg, png, bmp. It is scaled and rotated on import to allow easy network creation.

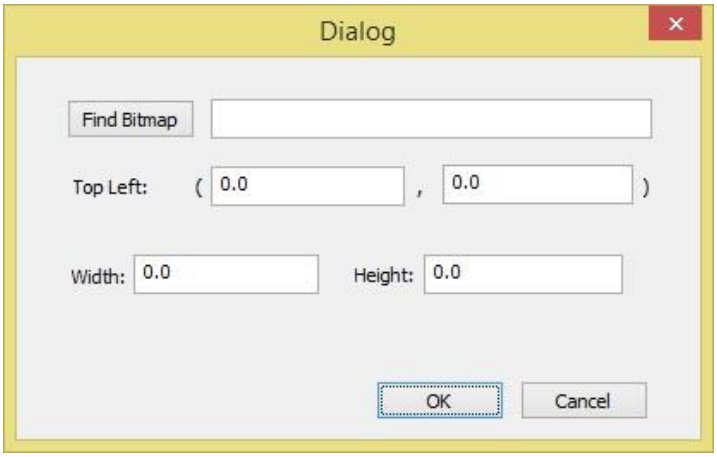


Figure 55: Dialogue showing the import function to scale and rotate an image

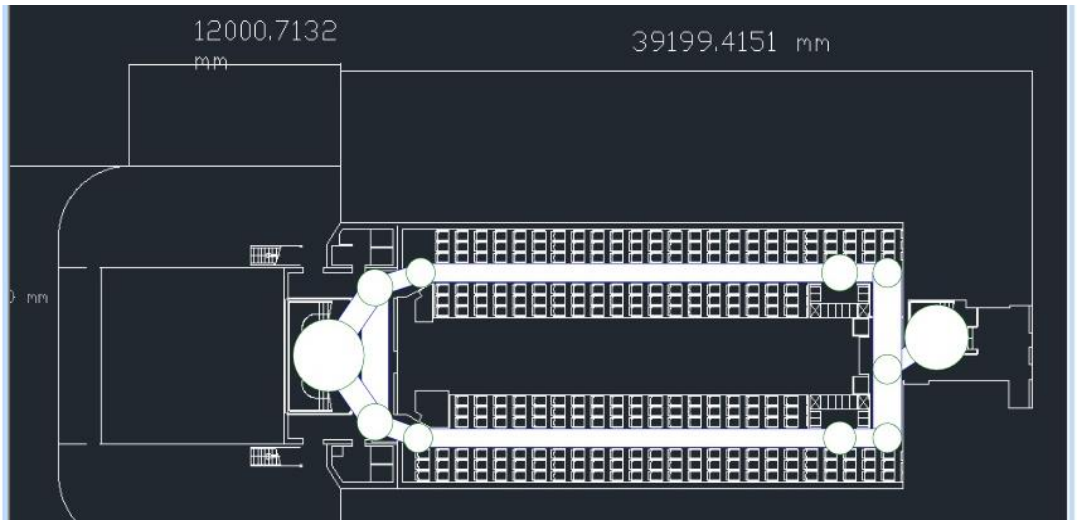


Figure 56: Example links and nodes in the GUI on one floor (scaled bitmap)

Each network can use a different relationship between the density of agents travelling down a link or through a node and the maximum speed those agents can move at. In fact, even within the same

network, a different relationship for each link or node can be used. The speed/density curve is set in the settings xml, in the Movement section there is a parameter called DensityEquations. If this is blank then the network will use the default (which is the "FruinFlow" curve).

The WP4 beta component allows the following set of speed/density curves:

- "fruin" - this is the basic Fruin curve taken from Dr John J. Fruin's book "pedestrian and Planning Design" (8th May 2002)
- "fruinflow" – a curve interpolated from Fruin's more basic 'levels of service'
- "mori" - this curve is from the paper "A new method for Evaluation of Level of Service in Pedestrian Facilities" by M. Mori and H. Tsukaguchi in 1987.
- "weidmann" - this is from the paper " Transporttechnik der Fussgänger, Transporttechnische Eigenschaften des Fussgängerverkehrs (Literturauswertung)" by U. Weidmann in March 1993
- "jamarat" - This is from the paper "Dynamics of crowd disasters: An empirical study" by D. Helbing, A. Johansson and H. Al-Abideen in 2007.
- "rahman" - This is taken from the paper "Weighted Regression Method for the Study of Pedestrian Flow Characteristics in Dhaka, Bangladesh" by Rahman, Khalidur; Ghani, Noraida Abdul; Kamil, Anton Abdulbasah; Mustafa, Adli. This is from April 2013.
- "ship" - this relationship was calculated from the information in the document "Guidelines for Evacuation Analysis for New and Existing Passenger Ships" - MSC.1/Circ.1238 from October 2007. Which is a document from the International Maritime Organization explaining how evacuation of ships should be simulated.
- "hankin" - This is a curve from the paper "Passenger Flow in Subways" by B. D. Hankin and R. A. Wright from 1958. This is most appropriate for underground rail networks.

3.5.2 Ontologies

After creating nodes and links in the network to build the model, it connects to SOFIA using the following input and output ontologies:

CDIL_AgentNetworkInput - Consists of an array of Edges and an array of Nodes, both of which consist of a pair of values for each element - an "ID" and "CurrentNumbers" which represent the ID of the Edge / Node and the number of Agents which are currently on that Edge / Node respectively. An example of a JSON string which can be sent as a NativeInsert message to the KP in order to populate it with values is shown below:

"{\"CDIL_AgentNetworkInput\":{ \"NetworkName\":\"MetroBilbao\",
\"Edge\":[{\"ID\":\"0\",\"CurrentNumbers\":499},{\"ID\":\"1\",\"CurrentNumbers\":45},{\"ID\":\"2\",\"CurrentNumbers\":5
5},{\"ID\":\"3\",\"CurrentNumbers\":7},{\"ID\":\"4\",\"CurrentNumbers\":7},{\"ID\":\"5\",\"CurrentNumbers\":55},{\"ID\":\"
6\",\"CurrentNumbers\":70},{\"ID\":\"7\",\"CurrentNumbers\":9},{\"ID\":\"26\",\"CurrentNumbers\":13},{\"ID\":\"33\",\"C
urrentNumbers\":153},{\"ID\":\"53\",\"CurrentNumbers\":7},{\"ID\":\"58\",\"CurrentNumbers\":50},{\"ID\":\"73\",\"Curre
ntNumbers\":35},{\"ID\":\"89\",\"CurrentNumbers\":55}],\"Node\":[{\"ID\":\"0\",\"CurrentNumbers\":16},{\"ID\":\"1\",\"C
urrentNumbers\":16},{\"ID\":\"2\",\"CurrentNumbers\":43},{\"ID\":\"3\",\"CurrentNumbers\":43},{\"ID\":\"4\",\"CurrentN
umbers\":6},{\"ID\":\"5\",\"CurrentNumbers\":45},{\"ID\":\"6\",\"CurrentNumbers\":45},{\"ID\":\"7\",\"CurrentNumbers\":

6},{\"ID\":\"8\",\"CurrentNumbers\":20},{\"ID\":\"9\",\"CurrentNumbers\":9},{\"ID\":\"16\",\"CurrentNumbers\":60},{\"ID\":\"17\",\"CurrentNumbers\":25},{\"ID\":\"27\",\"CurrentNumbers\":42},{\"ID\":\"30\",\"CurrentNumbers\":38},{\"ID\":\"48\",\"CurrentNumbers\":5},{\"ID\":\"49\",\"CurrentNumbers\":3},{\"ID\":\"50\",\"CurrentNumbers\":25},{\"ID\":\"69\",\"CurrentNumbers\":23}]}}"

This example is for the Metro Bilbao Network, representing a scattering of agents across the entire model. When this information is received from SOFIA it is stored into a std::vector of std::pair's of the ID and the number. This is then read by the system to populate the Agent Information which is used to spawn the correct number of agents in every Edge and Node. When the data is received a delete message can be sent to the KP to flush out all the data which currently resides in it. When the information is received, only the latest data is used in order to ensure the most up to date values are used. The NetworkName parameter must match the name of the network (MetroBilbao in this example) or it will not be returned by SOFIA.

CDIL_NetworkInput - Allows specified edges and nodes to be disabled or have their effective width / area changed as well as any escalators to be disabled as escalators (thus allowing flow in either direction and behaving like regular stairs). There are 3 arrays in the ontology, one for the widths of the edges, one for the widths of the nodes and finally the ids of any escalators which should be deactivated and thus behave as normal stairs. An example of a JSON string which can be sent as a NativeInsert message to the KP in order to populate it with values is shown below:

"{\"CDIL_NetworkParameters\":{ \"NetworkName\":\"MetroBilbao\",
\"EdgeWidthProportion\":[{\"ID\":\"33\",\"CurrentValue\":50},{\"ID\":\"34\",\"CurrentValue\":25},{\"ID\":\"97\",\"CurrentValue\":0}],\"NodeWidthProportion\":[{\"ID\":\"21\",\"CurrentValue\":10},{\"ID\":\"23\",\"CurrentValue\":0}],\"EscalatorsDeactivated\":[{\"ID\":\"92\"},{\"ID\":\"93\"}]}}"

If an edge or node is not listed in the respective width proportion array then it has an effective width of 100%. Passing through an effective width of zero for either a node or an edge will disable the node or edge. The NetworkName parameter must match the name of the network (MetroBilbao in this example) or it will not be returned by SOFIA.

CDIL_PolyInput - The flow and speed along an edge can be controlled based on dynamically read values for the relationship between the density of people moving and the speed at which they move. An ontology has been created for receiving the parameters required for the Polynomial creation, called. It consists of the network name and an array of Polynomials, which has an integer defining the degree of the polynomial, an array of pairs of floats (for x and y coordinates respectively), an array of integers for the IDs of Edges which should be assigned this curve and an array of ids for Nodes using this curve. An example JSON string which will populate this ontology with values is shown below:

"{\"CDIL_PolyInput\":{
\"NetworkName\":\"LoadedPolyTest\",\"Polynomials\":[{\"Degree\":4,\"Values\":[{\"X\":0,\"Y\":1.3},{\"X\":0.27,\"Y\":1.3},{\"X\":0.31,\"Y\":1.3},{\"X\":0.43,\"Y\":1.27},{\"X\":0.72,\"Y\":1.22},{\"X\":1.08,\"Y\":1.14},{\"X\":2.17,\"Y\":0.76},{\"X\":3,\"Y\":0}],\"Edges\":[0,2,4,6,8],\"Nodes\":[0]}]}}"

This example creates a polynomial of degree 4 with the values from the Fruin curve. This can be used to create a bespoke speed/density curve from sensor detection values to model the actual crowd dynamics occurring in the venue, or from historical data gathered in the venue itself as the eVACUATE system is used over time.

CDIL_DensityOutputFullRun - Stores the name of the Network which has produced the output and then an array of timesteps called ***CDIL_DensityOutputRun*** which each hold an array of Edges, which consist of an "ID" to represent the ID of the edge and a number of sections (currently limited to 3 but this could be changed) which contain the "ID" of the section, the current density of that section of the edge and the proportion of the agents in that section going towards Node A of the edge. Next each element of the array holds an array of Nodes which consist of two properties - the "ID" and the Density of that Node.

The JSON string is constructed piece by piece in our Density output code every Density output period in a separate thread to the main loop which allows the simulation to continue while this is being constructed. When the run is complete this is then sent via a Native insert message to SOFIA and this has been tested and works.

The Ontologies can be streamlined and tweaked to reduce the size of the JSON string required to populate the messages in order to speed up the sending / receiving of the information.

CDIL_RouteData – It's used to send the AER to the COP. For each node in the network, it defines the next edge and node that represents the next link in the path towards the final goal (evacuation destination). On top of this, it also lists the second best pathway, if one exists, and also the average evacuation time from each node (this is used to colour code the results in the COP). Lastly, the Network Name is also included, and the total evacuation time for all agents to leave the entire network. An example of a JSON string representing the AER for a Test network is shown below.

“{\"CDIL_RouteData\":{
\"Node\":[{\"ID\":\"0\",\"Destination\":36,\"NextNode\":11,\"AlongEdge\":10,\"AverageEvacTime\":171.136353},{\"ID\"
:\"1\",\"Destination\":36,\"NextNode\":0,\"AlongEdge\":1,\"AverageEvacTime\":499.717926}]},\"NetworkName\":\"Te
stNetwork\",\"EvacTime\":3.1}”

3.5.3 Running the Software

Finally, the command line program Continuousapp.exe /{insert network name}.xml will continuously loop to provide SOFIA with the necessary information for other systems to function.

3.6 3D INTERACTIVE COMMON OPERATIONAL PICTURE

3.6.1 Functionalities

3.6.1.1 Crowd Simulation

- Simulation server communicates through UDP
- Simulates single individual movement and decision taking
- Support for >1000 persons simultaneously
- Dynamic person creation and removal
- Dynamic incident creation and removal with route recalculation
- Dynamic target (exit) creation and removal with route recalculation
- Two level route planning mimics human behaviour
- Individuals respond to emergent situations (i.e.: overcrowding of an exit) and try to find alternative routes

3.6.1.2 Incidents

- Fire
- Explosion
- Smoke
- Brawl

3.6.1.3 Tools and general information

- crowdsim.7z contains the simulator
- testPath.7z contains the test program
- Test program defines in config.ini a second port to send (SendPort2) , this is the one used by the test program so it can run side to side with DXT Visualizer
- The test program expects the simulator to bind to 127.0.0.1 , it currently doesn't run in a remote compute
- crowdsim.exe can load one scene or another by the following ways:
- As a command line parameter
- From Key NavMap in config.ini
- From network through a command. The test program sends it to the simulator
- The accepted values for scene are : Anoeta , Athenas , Bilbao and Meraviglia
- On loading the scene, crowdsim reads a <scene name>.ini file with the projection used should you want to change it, that also includes a ZOffset to apply.
- Meraviglia ship is located in Arkitsa Harbor , near Thermopylae. That can be changed in Meraviglia.ini
- Dying : iAnimation set to 9

- Fighting : iAnimation set to 7

3.6.1.4 Test Program

- Camera can be moved pressing the right mouse button and moving the mouse and with keys WASD
- Ctrl+Click executes the tool selected in the right menu, parameters can be changed with the two sliders at the bottom
- The Delete command deletes the nearest exit or incidence

3.6.1.5 Anoeta projection

- SouthParallel = 33
- NorthParallel = 45
- OriginLatitude = 43.301381
- OriginLongitude = -1.973597

3.6.1.6 Bilbao projection

- SouthParallel = 41
- NorthParallel = 46
- OriginLatitude = 43.262474293758
- OriginLongitude = -2.9476085631378
-

3.6.1.7 Athenas projection

- SouthParallel = 33
- NorthParallel = 45
- OriginLatitude = 37.930494
- OriginLongitude = 23.944497
-

3.6.1.8 Meraviglia projection

- SouthParallel = 33
- NorthParallel = 45
- OriginLatitude = 38.751549
- OriginLongitude = 23.023677

3.6.1.9 Interface

For completeness and correctness, this document enumerates the current commands send/received by the simulation along with structures that Simulator expects with these commands.

In order to support dynamic changes of scene, Simulator support a special msg “EVAC_MSG_SIMULATION_LOAD_SCENE”. The parameter iScene expects the following values:

1. Bilbao
2. Anoeta
3. Athenas
4. Meraviglia

Common Structures:

```
// Identifier
struct t_Identifier
{
   // Creator of the element
   int32_t iCreator;
   // Unique Id.
   int32_t lId;

   void Swap()
   {
      SwapB(&iCreator);
      SwapB(&lId);
   }
};

// Position
struct t_Position
{
   // Latitude (º)
   double dLatitude;
   // Longitude (º)
   double dLongitude;
   // Altitude over sea level (m)
   float fAltitude;
   // Height over terrain (m)
   float fHeight;

   void Swap()
   {
      SwapB(&dLatitude);
      SwapB(&dLongitude);
      SwapB(&fAltitude);
      SwapB(&fHeight);
   }
};

// Orientation
struct t_Orientation
{
   // Yaw (rad)
   float fYaw;
   // Pitch (rad)
```

```
    float fPitch;
    // Roll (rad)
    float fRoll;

    void Swap()
    {
        SwapB(&fYaw);
        SwapB(&fPitch);
        SwapB(&fRoll);
    }
};
```

Constants:

```
enum EVAC_CLASSES
{
    EVAC_CLASS_SIMULATION   = 1,
    EVAC_CLASS_ENTITY                   = 2,
    EVAC_CLASS_DETONATION    = 6,
    EVAC_CLASS_INCIDENTS = 8,
};

enum EVAC_OBJECTS
{
    EVAC_OBJ_SIMULATION       = 1,
    EVAC_OBJ_ENTITY             = 2,
    EVAC_OBJ_DETONATION  = 6,
    EVAC_OBJ_INCIDENTS = 8,
```

Common Structures:

```
};
```

Messages:

```
EVAC_MSG_SIMULATION_UPDATE_STATE          = 1000,
EVAC_MSG_SIMULATION_ON_UPDATE_STATE    = 1001,


struct t_Msg_Simulation_Update_State
{
    // Engine state (any of EVAC_STATES)
    char cEVacState;
    // Pause / Unpause
    bool bPause;
};



EVAC_MSG_SIMULATION_REQUEST_SYNCH = 1601,
(No data)
```

```
EVAC_MSG_SIMULATION_LOAD_SCENE = 1899,
struct t_Msg_LoadScene{
   int32_t iScene;
};



EVAC_MSG_ENTITY_CREATE          = 2000,
EVAC_MSG_ENTITY_ON_CREATE       = 2001,

struct t_Msg_Entity
{
   // Id Entity
   t_Identifier idEntity;
   // Visual model
   int32_t lVisualModel;
   // Enable
   bool bEnable;
   // Side
   int32_t iSide;
   // Health status (any of EVAC_LEVELS)
   char cHealthStatus;
   // Entity position
   t_Position pos;
   // Orientation
   t_Orientation orientation;
   // Template
   char cTemplate[60];
   // Id de la crowd a la que pertenece
   t_Identifier idCrowd;

   int32_t iAnimation;
};

EVAC_MSG_ENTITY_UPDATE          = 2004,
EVAC_MSG_ENTITY_ON_UPDATE       = 2006,

struct t_Msg_Entity_Update
{
   // Id Entity
   t_Identifier idEntity;
   // Enable
   bool bEnable;
   // Side
   int32_t iSide;
   // Health status (any of EVAC_LEVELS)
   char cHealthStatus;
   // Entity position
   t_Position pos;
   // Orientation
   t_Orientation orientation;

   int32_t iAnimation;

}


EVAC_MSG_EXIT_CREATE = 8016,
EVAC_MSG_EXIT_ON_CREATE = 8017,
```

```
struct t_Msg_Exit{
   t_Identifier id;
   t_Position   position;
   int32_t        radius;

}


EVAC_MSG_CREATE_FIRE = 9000,
EVAC_MSG_ON_CREATE_FIRE = 9001,
EVAC_MSG_CREATE_SMOKE = 9010,
EVAC_MSG_ON_CREATE_SMOKE = 9011,
EVAC_MSG_CREATE_DETONATION = 9020 ,
EVAC_MSG_ON_CREATE_DETONATION = 9021 ,
EVAC_MSG_CREATE_FIGHT = 9030 ,
EVAC_MSG_ON_CREATE_FIGHT = 9031 ,

 struct t_Msg_Incident
{
   t_Identifier id;
   t_Position position;
   int32_t radius;

}

EVAC_MSG_ENTITY_DELETE          = 2002,
EVAC_MSG_ENTITY_ON_DELETE       = 2003,
EVAC_MSG_EXIT_DELETE = 8018,
EVAC_MSG_EXIT_ON_DELETE = 8019,
EVAC_MSG_DELETE_FIRE = 9002,
EVAC_MSG_ON_DELETE_FIRE = 9003,

EVAC_MSG_DELETE_SMOKE = 9012,
EVAC_MSG_ON_DELETE_SMOKE = 9013,
EVAC_MSG_DELETE_DETONATION = 9022 ,
EVAC_MSG_ON_DELETE_DETONATION = 9023 ,
EVAC_MSG_DELETE_FIGHT = 9032 ,
EVAC_MSG_ON_DELETE_FIGHT = 9033 ,

EVAC_MSG_UPDATE_FIGHT = 9034,

EVAC_MSG_ON_UPDATE_FIGHT = 9035,

struct t_Msg_Delete
{
   t_Identifier id;

}

enum EVAC_STATES
{
   EVAC_STATE_STOP = 0,
   EVAC_STATE_PLAY = 1,
};


enum EVAC_LEVELS
{
```

```
    EVAC_LEVEL_LOWEST       = 0,
    EVAC_LEVEL_LOW          = 1,
    EVAC_LEVEL_MEDIUM       = 2,
    EVAC_LEVEL_HIGH         = 3,
    EVAC_LEVEL_HIGHEST      = 4,
};
```

3.7 APPS FOR SMARTPHONE

3.7.1 System deployment overview

Prior to actual deployment a site-survey needs to be performed in order to identify the indoor-location requirements. The optimal placement of the BLE Beacons needs to be achieved, in order to achieve the position accuracy required in order to guide people in case of evacuation to safety through the active exit route. First we need to assess if there are areas that may be difficult to cover creating potential “blind-spot” conditions and (if possible) avoid the reception of multiple beacon signals, due to signal reflections. Then the beacons are deployed on the points identified, we calibrate the beacon orientation and height. In some cases, the transmit options of some beacons (strength, frequency, etc) need to be reconfigured to improve signal reception.

In order to support the process, we have developed a deployment support tool in order to identify the beacon positions, with actual map-locations in the venue map.

Among other, we also need to ensure that all the possible locations which need to be supported by the application (i.e. all the ship spaces that a passenger can access) have good Wi-Fi coverage. The application is can start without an active network connection, however a connection is needed so that the application may receive the signal that evacuation has been declared, and receive the exit instructions according to the identified Active Exit Route.

For the FR & Missing Person applications, we need to insert into our database the POIs locations, the Mustering stations and the FRs Check-in locations.

3.7.2 Integration with eVACUATE

We deploy our application backend server into the secure network of the Wi-Fi-access network. eVAMAPP requires connectivity with the SOFIA Server.

We configure the options and start our server.

Afterwards, we need to create an eVamapp configuration record in our cloud server so that all the mobile phones can get the local network settings through it.

The First Responder Application is communicating with the eVAMAPP Backend Server, and interchange information with the COP and EOC platforms through the SOFIA.

The Missing Person Application is designed to interchange all the data with an API Server, and this server can be a middleware that communicates real-time with the (Fidelio) Hospitality Management Server of the Ship.