CRAMSS APPLICATION

Project Title	RESOLUTE
Project number	653460
Deliverable number	D5.3
Version	9.1
State	Final
Confidentially Level	Public
WP contributing to the Deliverable	WP5
Contractual Date of Delivery	M24 (30/04/2017)
Finally approved by coordinator	19-06-2017
Actual Date of Delivery	19-06-2017
Authors	Mr. Alexandros Zamichos; Dr. Anastasios Drosou; Mr. Ioannis Symeonidis; Mr. Aristotelis Spiliotis
Email	drosou@iti.gr
Affiliation	CERTH-ITI
Contributors	**CERTH**: Dr. Ilias Kalamaras; Mr. Athanasios Tryferidis; Mr. Triantafyllos Tsirelis; **FhG**: Jan-Paul; Leuteritz; **CMR:** A. Candelieri; **UNIFI:** E. Bellini; P. Nesi, **SWARCO**: R. Di Vincenzo; **THALES** A. Grifoni

Funded by the Horizon 2020
Framework Programme of the European Union

Project Context

Work package	WP5: Platform front-end and end user application
Task	T5.2: CRAMSS application
Dependencies	WP5; T5.3; T5.4

Contributors and Reviewers

Contributors	Contributors	Reviewers
A. Drosou (CERTH)	I. Symeonidis(CERTH)	Manfred Dangelmaier (FhG)
A. Zamichos (CERTH)	A. Spiliotis (CERTH)	Emy Apostolopoulou (ATTIKO)
T. Kalamaras (CERTH)	J.P. Leuteritz (FhG)	
A. Tryferidis (CERTH)	A. Candelieri (CMR)	
T. Tsirelis (CERTH)	E. Bellini (UNIFI)	
Manfred Dangelmaier (FhG)	R. Di Vincenzo (SWARCO)	
Emy Apostolopoulou (ATTIKO)	A. Grifoni (THALES)	

Version History

Version	Date	Authors	Sections Affected
V0	4/4/2017	A. Drosou, A. Tryferidis	All – ToC proposal
V1	11/5/2017	A. Drosou, A Zamichos, I. Kalamaras, A. Spiliotis, I. Symeonidis	All -1st draft circulated for input contribution
V2	20/5/2017	A. Drosou, A Zamichos, I. Kalamaras, A. Tryferidis, T. Tsirelis	1, 2, 3, 8
V3	29/5/2017	E. Bellini, P. Nesi	6, 7

V4	4/6/2017	A. Candelieri	Section 3.6.2.1.3
V5	5/6/2017	R. Di Vincenzo	4
V6	12/6/2017	A. Drosou, A. Tryferidis, Jan-Paul Leuteritz	All – Pre-final version
V7	13/6/17	A. Grifoni	5
V8	13/6/17	A. Drosou, T. Tsirelis, I. Symeonidis	All – Final for internal review
V9.1	16-6-17	P. Nesi	all

Abbreviations

Abbreviation	Full term
CCRP	Capacity Constrained Route Planner
CRAMSS	Collaborative Resilience Assessment and Management Support System
CSM	Contextual Simulation Module
CSV	Comma-Separated Values
ERMG	European Resilience Management Guidelines
ESSMA	Emergency Support smart mobile app
GBTA	Game-based Training app
GIS	Geographic Information System
GS	Graph-based Simulator
GUI	Graphical User Interface
SAVE ME	System and actions for vehicles and transportation hubs to support disaster mitigation and evacuation (SAVE ME project; Grant Agreement No. 234027 of the European Commission)
TSM	Traffic Simulation Module
UI	User Interface
UTS	Urban Transport System(s)

Gender writing statement

This deliverable contains gender-specific terms. They have been avoided where possible. Where not possible, the male and the female form were used at random, for reasons of shortness and readability. In all cases, it is assumed that the respective person could be of any gender.

Table of Content

List of Figures

List of Tables

1 INTRODUCTION

1.1 Scope of this Deliverable

The scope of this deliverable is to present the result of Task 5.2 which is the *CRAMSS (Collaborative Resilience Assessment and Management Support System)* application. The CRAMSS has been implemented considering the ERMG guidelines developed in WP3 and it is primarily a concept or an idea of a collaborative workspace in which DSS operators of the urban transport system (UTS) can share their outputs of or information about their work among each other. In this direction, three front-end applications have been implemented that allow DSS operators to communicate, composing CRAMMS's front-end interface, namely: *Dashboard, Evacuation DSS (eDSS), and Resilience DS Tool.* The eDSS and the Resilience DS Tool also hold a back-end which in conjunction with information derived from other DSSs (i.e. *UTM (Urban Traffic Management)*, *UPT (Urban Public Transport)*), the Emergency Support Smart Mobile App (ESSMA), and/or the Data Management Layer provide information to the front-end. The secure exchange of data among the CRAMSS's components is achieved utilizing the *ESB (Enterprise Service Bus)* described in D4.5.

1.2 Relation to other Deliverables

The outcomes of the Task 5.2 "CRAMSS application" are strictly connected with the backend implementation and integration Task 4.1, Task 4.3, Task 4.5 and with the available data identified in Task4.2 and the produced data of Task 4.4. Moreover, the Task5.2 is related with the outcomes of the Task5.1 where the designs for the interface of the CRAMSS application were developed. The produced designs were used as the basis for the development of CRAMSS's interface. Finally, the task is related with Task 5.3, which concerns the development of the Emergency Support Smart Mobile App (ESSMA), and the outcome of Task 5.4, which is the Game-based Training app.

1.3 Deliverable Structure

The deliverable is structured and organized as follows. In the second section, the overview of the CRAMSS is described. The section starts with a general description of what CRAMSS is and continues with the description of the involved users and their roles. Next, the objectives of the application are analysed and the application's features that cover these objectives are discussed in detail. Section 2 closes by providing the CRAMSS architecture and locating it within RESOLUTE's overall architecture. The next five sections are dedicated to the detailed description of the core components of the CRAMSS, namely: the EVACUATION DSS, the UTM DSS, the UPT DSS, the Resilience Dashboard and the Resilience DS TOOL. Each section details the functional description of the corresponding component and its outcomes. Finally, section 8 concludes the deliverable and gives a brief summary of the most important aspects.

2 CRAMSS OVERVIEW

2.1 Introduction

The main purpose of the CRAMSS as it was reported in D5.1 is "*to support reference actors at the UTS, such as infrastructure managers, with their decision making under both, standard operating conditions and emergency conditions. The CRAMSS displays information from different sources or independently running web-applications, together with the results of the decision support*". Based on this and due to reasons that were analysed in detail in D5.2, the CRAMSS was decided to be *"primarily a concept or an idea of a collaborative workspace in which DSS operators can share their outputs of or information about their work among each other"* (D5.2 – section 2.1). Thus, it was decided that the CRAMSS UI would consist of three different UIs (i.e. Dashboard, eDSS UI, Resilience DS) besides a common one.
Except from the UIs (front-end), the CRAMSS is also has a backend. The backend is composed of several DSSs (i.e. eDSS, UTM, UPT, and Resilience DS) that processes information related to the UTS, exchange this information among them by utilizing the ESB, and provide it to the front-end. The eDSS is responsible for calculating the optimal evacuation plan in case of great danger and provide it to the operator and to the citizens through the ESSMAs. The UTM DSS provides traffic related information such as traffic measures; traffic events and traffic strategies (e.g. close/open a road), while the UPT DSS provides information regarding the means of public transport. Finally, the Resilience DS operationalizes the FRAM model in order to provide operators with an overview of which resources are necessary for which decision, which guidelines apply to the specific decision-making, and who are contact persons in other organizations that provide resources for one's own decision.

The outcome information of the backend is visualised in a human-friendly manner via the front-end. Moreover, the UIs are provided as interactive means where the operator may play an active role not only to the management of the received information, but also to its production (e.g. evacuation planning). Thus, the front-end and the backend of the eDSS are inseparable, as well as those of the Resilience DS tool. The UIs may also display information coming from the data layer or the rest RESOLUTE's components (e.g. application framework). The Resilience Dashboard is responsible for displaying mainly information coming from the data layer, the UPT and the UTM DSSs. The evacuation plan computed by the eDSS is also sent to the Dashboard in order to be approved by the Central Decision Maker (CDM).

2.2 Users and Roles

The Figure 1 presents an overview of the three CRAMSS UIs (described in the previous section) and of the involved users. As it is shown, the Resilience Dashboard and the Resilience DS tool are available for use by all the involved **DSS operators** of the UTS, while the eDSS is reserved for one specific operator (**eDSS operator**). Each DSS sends information to the ESB, which redistributes the information to the different instances of the Dashboard. Regarding the use of Dashboard there are two kind of user: a) the operators/stakeholders who have a read-only access to the data provided in the Dashboard, and b) the **central decision maker (CDM)**, who can read all the available data provided in the Dashboard, as well as, he/her is the final authority regarding the evacuation of citizens. So, when the evacuation plan is produced by the eDSS it is sent to the CDM for approval. If the CDM approves the evacuation plan, the evacuation routes are sent to the citizens through the ESSMAs. The CDM is only one per city and may be for instance the city's mayor. Considering the first category (the operators), they are expected to be members of different organizations and have an individually considered dashboard and access to the Resilience DS. A more detailed description of who the targeted operators of the CRAMSS are can be found in D5.2.

In addition to the group of the operators, there also the group of the **ESSMA users** that communicates with the eDSS, as Figure 1 shows. The ESSMA is a cross platform application that is addressed to every citizen that owns a smartphone (either android or iOS). During the installation of the application, each user defines profile details as well as his/her preference for voluntary help during emergencies. Thus, the ESSMA users can be categorized in two sub-groups a) the **non-helpers** and b) the **helpers (or voluntary helpers)**.

On the one hand, the "non-helpers" are users that are not subscribed as voluntary helpers and thus they receive information relevant only to their self-rescue. On the other hand, the "helpers" are common citizens or professional rescuers that are subscribed as voluntary helpers. Except from information regarding their self-recue, the latter may also receive guidance in order to help/save other trapped/ injured people that need help. A more detailed description of the communication between the ESSMA users and the eDSS can be found in Section 3, as well as in D5.4.

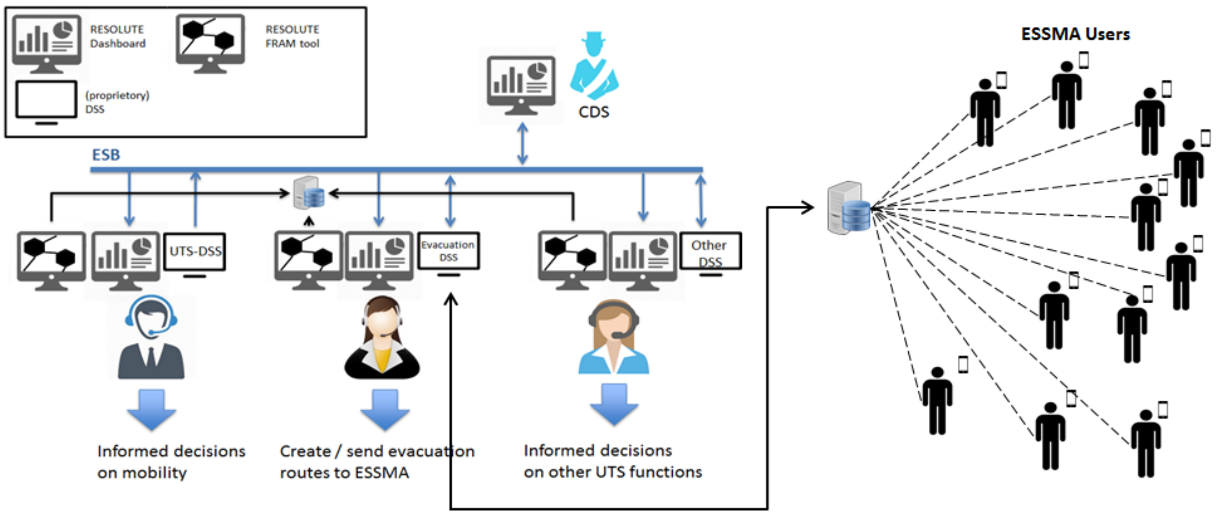

RESOLUTE Dashboard
RESOLUTE FRAM tool
(proprietory) DSS
CDS
ESSMA Users
ESB
UTS-DSS
Evacuation DSS
Other DSS
Informed decisions on mobility
Create / send evacuation routes to ESSMA
Informed decisions on other UTS functions

Figure 1. CRAMSS users & roles

2.3 Application Objectives

As it has been already mentioned, the main objective of the CRAMSS is to support the UTS stakeholders to take critical decisions under both standard and emergency conditions. According to the DOW, the CRAMSS has also to address the following objectives:

a) **Intervene in the information exchange between T4.2 & T4.4 modules & the Back-End Platform, through the integration framework (T4.5):** The front-ends of the CRAMSS are the means for visualizing the data collected in T4.2 and the produced data of T4.4. Moreover, the back-end modules of the CRAMSS process this information and exchange the produced knowledge by utilizing the ESB (T4.5).

b) Adaptive and constantly learning decision support system: Due to the continuous exchange of valuable information among the CRAMSS's components, the system can be adapted in time to the needs of each emergent event. For instance, when a critical event relative to the UTS occurs (e.g. car accident) the UTM DSS can inform the eDSS so as to exclude this particular road from the calculation of the evacuation routings, avoiding thus to guide the crowd through the affected road.

c) **Real time risk assessment/detection:** The data collected from the city's sensors (e.g. traffic sensors, river level sensors, etc.), as well as data retrieved by the application framework are used by the CRAMSS components in order to alert the operators about emergent events. Alerts in form of notifications are displayed through the front-end of the CRAMSS, informing the operators about the situation and the severity of the events.

d) **Identify individuals or groups of individuals as rescuers or to-be-rescued:** The eDSS in conjunction with the ESSMA are used for the identification of groups of individuals as rescuers or to-be-rescued. By utilizing the smartphones' sensors, the ESSMA provides useful information to the eDSS regarding the location of all the ESSMA users. As it has been described in Section 0 the ESSMA users can be categorized in helpers and non-helpers. Therefore, when there is the need for help the system recognizes the voluntary helpers as possible rescuers. Before sending guidance to a voluntary helper the system reassures that the user are available to help through a short communication procedure. Regarding the to-be-rescued users, the ESSMA provides to all users a mean for calling for help when it is necessary. Except from their location and their need for help, they can also send additional information regarding their condition and the severity of the situation. Thus having the knowledge of the location and the number of each group of users, the system calculates the optimal action plan.

e) **Multiple (modality) Input-multiple (modality) Output (MIMO):** The system receives multiple inputs from different kind of sources (e.g. sensors, maps, smartphones, etc.) processes them and provide multiple (modality) outputs (e.g. evacuation plan (visual), alert messages (text & sound), etc.).

f) **Guidance in Emergencies:** Additionally to the groups of individuals as rescuers or to-be-rescued the system can also identify users that need to be evacuated. The operator can select/define the areas that have to be evacuated, so as to users within these areas to receive guidance in order to exit the hazardous areas. The system computes the optimal evacuation plan with the minimum cost for the UTS. Depending on the type and the location of each user, they receive the corresponding guidance. The ESSMA users that are not included in one of the aforementioned categories (e.g. non-helpers users that are away of the areas in danger) receive information about the emergency, in order to guide them passively not to approach the affected areas.

g) **Communicate & display to users/public the optimal resilience strategies, from a set of Pareto strategies:** The CRAMSS application provides different means-applications to communicate with each kind of user. Regarding the users-operators the UIs of the CRAMSS provides the interactive means for bi-directional communication among users and the CRAMSS. Considering the public, they can communicate with the CRAMSS via the ESSMA. They are able not only to receive guidance and to be informed about the emergencies but they can also provide useful real-time information about the condition of the emergency. The information is displayed to users in different layers of abstraction. This means that the ESSMAs receive less or filtered information about the emergencies and the evacuation plan (they receive only the route that they have to follow). On the contrary, the application gives the whole view of the situation to the users-operators.

The system allows the operator to have an active role in the selection of the strategy to be sent to the public, as for instance in the selection of the evacuation plan. Based on the operator's actions (e.g. include/exclude a road from the calculation of the evacuation plan, select the evacuation area, etc.) the system provides an optimal evacuation plan. Before sending the evacuation routes to the ESSMA users, the evacuation plan is proposed to the eDSS operator and to the CDM for approval. If they decide that the evacuation plan has to be rejected the whole procedure can be re-initiated again, allowing the operator to change his actions. In this way, the optimal resilience strategy is sent to public based on the operator's decisions.

2.4 CRAMSS Architecture

As Figure 2 displays, the CRAMSS is the central component of the RESOLUTE's architecture. It seamlessly fuses information from different sources (e.g. data management layer, application framework, actuations channels, etc.), providing decision support services and information to Urban Transport related authorities via the front-end applications, facilitating thus, an efficient management of the Urban Transport System and real-time applicable countermeasures in critical situations (e.g. evacuation planning, etc.).

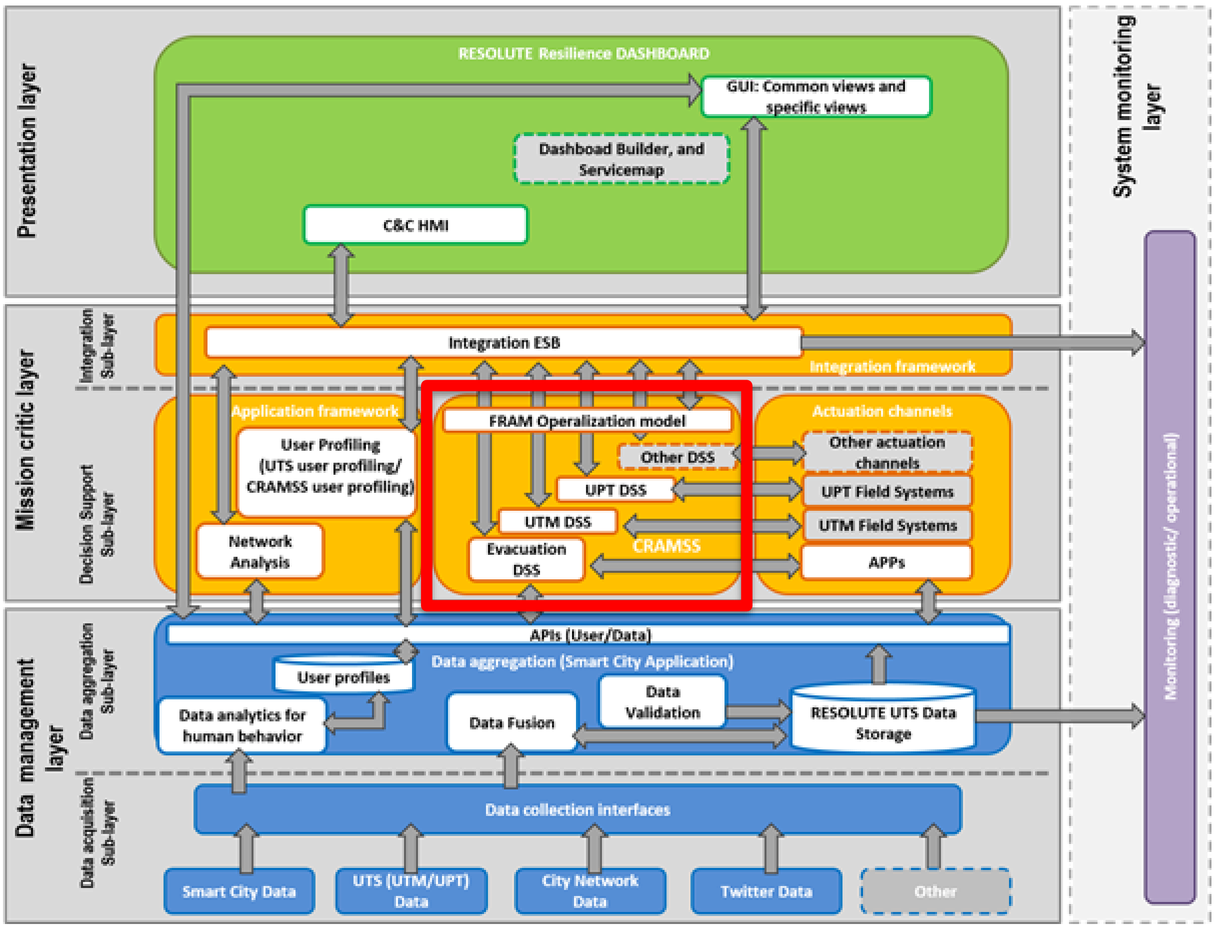


Figure 2: RESOLUTE's architecture with CRAMSS highlighted

Referring to the back-end, the main components of the CRAMSS are the Evacuation DSS (eDSS), the UTM (DSS), the UPT (DSS) and the FRAM and are connected as shown in Figure 2. The next chapters contain a detailed description of each of the CRAMSS components

3 EVACUATION DSS

3.1 Introduction

One of the core components of the CRAMSS is the **evacuation Decision Support System (eDSS)**. The eDSS is the responsible module for providing evacuation planning to the evacuation responsible (eDSS operator) in critical situations, facilitating them to take critical decisions. In order to provide optimal evacuation plans, considering the number of the involved ones and the critical situation, the eDSS co-processes and fuses all the available information from all the existing sources. Thus, the eDSS considers information retrieved from the Data Management Layer, the UTM DSS, the eDSS's front-end, as well as data retrieved from the ESSMAs. Except from the evacuation plans the eDSS is also responsible for identifying possible individuals or groups of individuals as rescuers or to-be-rescued, assigning the appropriate task to each and providing the corresponding guidance. The eDSS algorithms are able to cope with optimal evacuation planning in three modes: a) personalized routing based on user-specific profiles, b) group-wise for general public guidance and c) collaborative rescue for citizens that are willing to help trapped or injured travellers.

Of course, all this information has to be managed and checked by an experienced person, the eDSS's operator. The operator is the person (or a group of people) who has to oversee and evaluate the computed tours of the evacuation plans. The operator can interact with the whole procedure by using the eDSS's front-end (e.g. block a road; mark a danger area, request for evacuation planning, etc.). The computed evacuation plans are also sent to the Central Decision Maker (CDM) (e.g. the mayor) for the final approval, adding thus another layer of evaluation. After, and only after, the acceptation of the latter, the routes can be forwarded to either the travellers (via the ESSMA) or to local authorities for guiding the moving population.

It should be also mentioned that after sending the evacuation plan to the users, even if the eDSS keeps tracking of their location, it does not check whether a user follows the proposed routing. Furthermore, it does not check whether a user does or does not arrive at the end- safe point.

The eDSS is composed by a front-end and a back-end. The front end of the eDSS was developed in AngularJS [19] with the use of the Bootstrap framework [20] and the Leaflet [23] library for utilizing the map. Furthermore, the WebSockets [21] technology was utilized. WebSockets is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply. WebSockets was used in order to notify the operator of new chat messages, new comments at their posts, the status of the voluntary helpers (available helpers or occupied), as well as ESSMA users that need help. The back-end is a C++ implementation, which communicates with the front-end with the use of the C++ REST SDK [22] for the development of the RESTful web services that enables this communication. The C++ REST SDK is a Microsoft project for cloud-based client-server communication in native code using a modern asynchronous C++.

3.2 Supporting resilience

Within an urban transport system, streets and places and their ability to contain a maximum number of people (max capacity), as well as the continuously moving citizens that affect the variance of the values of the street's capacities, can be considered as resources. The eDSS supports the resilience of the whole system by a) helping the urban transport related authorities to make optimal use of these resources and thus contribute to minimizing damage during a critical event (RESPOND) or b) preventing critical events from happening by evacuating areas before a mass panic can even start (ANTICIPATE).

3.3 State of the art

3.3.1 Evacuation related Decision Supporting Systems

In the related literature, a number of different approaches can be identified that provide evacuation modelling. The majority of these approaches use Decision Support Systems (DSS) in order to help the operators to make decisions about emergency conditions that may be rapidly changing and not easily specified in advance. There are two main conceptual kinds of models in the evacuation management systems namely: the macroscopic and the microscopic.

On the one hand, the **macroscopic** model treats the evacuees (those to be evacuated) as a homogenous group for which only common characteristics are considered, for instance an average human being is assumed. The macroscopic model aims to succeed lower time of the evacuation plan execution without taking into consideration the personal requirements of each actor that participates in the evacuation procedure [1].

On the other hand, the **microscopic** models are more personalized to each actor providing to them more specified evacuation paths based on their special needs. These models consider individual parameters such as walking speed, reaction time or physical abilities as well as the interaction of evacuees during the entire evacuation process. Additionally, the microscopic approach of evacuation planning is more complex, because it does not only consider the characteristics of the users, but also the movement and the behavior of other travelers within the network [2]. Within the RESOLUTE project, the eDSS combines the macroscopic models methodology with microscopic analysis for the production of the evacuation routes.

In the related work several evacuation decision support tools were implemented. The Oak Ridge Evacuation Modelling System (OREMS) has the ability to analyze and evaluate large-scale vehicular emergency evacuations, calculate estimation for the evacuation time and the development of evacuation plans [3]. OREMS can be used for identifying traffic bottlenecks, and for the evaluation of traffic management strategies. However, it can only assign passenger cars, since it does not perform modal split [4]. Another approach is the Personal Computer based Dynamic Network Evacuation (PCDYNEV), a macroscopic model which consists of two main parts: integrated TRaffic Assignment and Distribution model (TRAD), and Interactive Dynamic Evacuation (IDYNEV). TRAD is responsible for generating, through the application of user-equilibrium theory, and distributing to the stakeholders, the destination and the routes of the evacuation plan. Moreover, the IDYNEV component is a simulation model of the traffic flow in the form of time-varying statistical histograms on each road segment [5].

A number of related works have used NETSIM and an improvement of this, the NETVACI methodology for evacuation of large transportation networks [6]. NETVACI models traffic patterns, using dynamic route selection and simulation of alternative evacuation scenarios in term of weather conditions, intersection controls and lane management strategies. The EXITUS system takes into consideration the uniqueness of each actor, dealing with disabilities explicitly in terms of physical and psychological attributes [7]. Another interesting work is a mechanism that provides both notification alerts and personalized evacuation paths for indoor working environments. This system consists of three components CAP-ONES for notifying emergency alerts, NERES for defining emergency plans and generating personalized evacuation routes, and iNeres as the interface to receive and visualize these routes on smartphones [8]. In the field of personalized guidance for emergencies, the research work in the Wuhan University of Technology uses the Wardrop Equilibrium Model in order to calculate and create personalized evacuation paths for the entire crowd in the campus of the University [9]. Furthermore, the work of [10] presents a way for providing group-wise optimal routes to the exits, along with personalized routing [10].

3.3.2 Evacuation Planning Algorithms

There are two main categories of evacuation planning methods. The first category contains Integer/Linear Programming (ILP) methods, which provide optimal evacuation solutions. For example the NETFLO [11], RELAX [12], and Cost Scaling [13] are all optimal evacuation route planners that can generate optimal evacuation plan by

performing the following three steps: creating a time-expanded network, applying a minimum cost flow algorithm, and extracting an evacuation time. Although these methods provide optimal route planning they present two major drawbacks, their poor scalability and the requirement of prior knowledge of the upper bound of the evacuation time *T.* These methods require time-expanded networks to produce a solution. If the original network has *n* nodes and the time upper bound is *T*, the time-expanded network will have at least (T+1)*n nodes. The computational cost of an ILP approach depends on the method used to solve the problem. By using the ellipsoid method, the work that is described in [16] showed that the problem can be solved optimally with computational cost $O(n^6)$. Thus, the ILP methods may be useful for evacuation scenarios with small size networks like indoor evacuations (e.g. buildings evacuation), but they do not perform well when applied to big networks such as the urban road networks. Moreover, the upper time bound of evacuation *T* has to be predetermined before the evacuation process, something that is difficult and dangerous to be done since if *T* is under-estimated, the system is probably to fail; otherwise, the system has a large graph to apply the optimization techniques.
The second category contains heuristic techniques, like Capacity Constrained Route Planning (CCRP) [14] and MRCCP [15], which have eventually been implemented and tested thoroughly. While the MRCCP provide good solutions with better running time than the ILP approaches its computational cost of $O(p \cdot n^2 \cdot logn)$ is worse than the CCRP's, where *p* is the number of evacuees and *n is* the number of nodes. The CCRP models capacity on edges as a time series and uses a capacity constrained routing approach to incorporate route capacity constraints, providing thus sub-optimal solutions for the evacuation planning. The worst-case time complexity of CCRP is $O(p \cdot n \cdot logn)$ which is better than both the ILP and the MRCCP approaches.

3.4 Evacuation DSS Architecture

In daily conditions, the problem of moving from one point to another can solved by providing the optimal route. However, in emergencies the optimal route is not always the most preferable route. Other factors should also be carefully examined for the selection of the routes, such as the location and the impact of the emergency event, road damages, the special needs of the involved persons, etc. Figure 3, displays the eDSS architecture that addresses all the aforementioned requirements, by processing various input data from different sources.

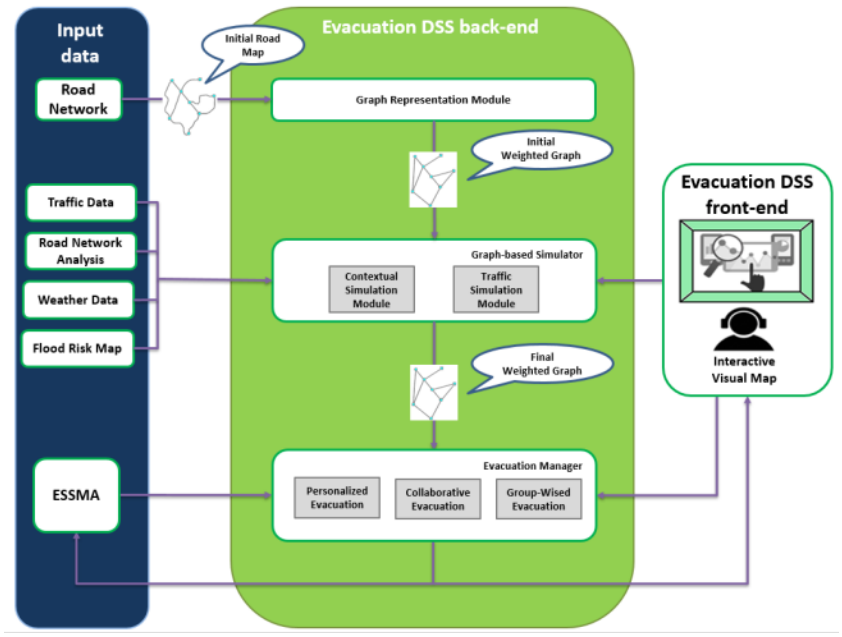


Figure 3. Evacuation DSS architecture

The eDSS developed within the RESOLUTE project, aims at helping the UTS responsible to calculate evacuation paths for the public, taking into account profile information and the special needs of each involved person in case of emergencies, as well as, to provide action routes to voluntary helpers in order to reach trapped/injured citizens.

3.5 Data Sources & Data processed by eDSS

As shown in the eDSS's architecture displayed in Figure 3, the module receives input data coming from different sources. A brief overview of the basic data sources and the provided data follows in Table 1.

Table 1: Input Data

Source	Data
Local Storage	• road infrastructure • flood susceptibility maps
Data Management Layer	• traffic related data • people waiting areas
ESSMA	• users' profile • users' location • users' situation
Application Framework	• users' profiling • flood hazard • network analysis
UPT	• real-time location of people on trams and on platforms connected to tramway public Wi-Fi (via Data Management Layer)
UTM	• traffic data • traffic events • traffic strategies (e.g. close/open a road)
eDSS's front-end	• Events' information (e.g. location, severity, etc.) • Areas to be evacuated • Information regarding the people to be evacuated (e.g. special needs, location, number, etc.) • request for evacuation planning • block/unblock a road

3.5.1 Urban Road Network & Maps

3.5.1.1 Firenze Map

Regarding the city of Firenze and the provided road data, the road infrastructure was expressed in terms of nodes and edges. For each road the following parameters was available, as they are described in Table 2.

Table 2: Available parameters of Firenze's roads

Type	Example	Description
Road Element Unique identifier	RT05100251420ES	Defined according to the following rule: • Characters 1, 2: RT • Characters from 3 to 8: ISTAT code of the municipality where is localized the road element • Characters from 9 to 13: progressive starting from the value of the characters 3 to 8 • Characters 14,15: ES
Road Element Type	0205	1) 0100 = carriageway trunk 2) 0200 = area of structured traffic 3) 0201 = the toll / motorway barrier 4) 0204 = square 5) 0205 = roundabout 6) 0206 = crossing 7) 0207 = structured parking 8) 0300 = area of unstructured traffic 9) 0301 = park 10) 0307 = in the area of relevance 11) 0400 = pedestrian 12) 5100 = connection, link road, junction 13) 5200 = service road 14) 5300 = ferry (a dummy element)
Technical functional classification	0300	1) 0100 = Highway 2) 0200 = Main interurban 3) 0300 = secondary suburban 4) 0400 = scroll Urban 5) 0500 = urban neighbourhood 6) 0600 = for private use
Element location	0200	1) 0100 = flush 2) 0200 = bridge 3) 0400 = ramp 4) 0500 = gallery
Official name written out in full	S.S. SENESE ARETINA (73)	Official road names
Initial Node (NOD_INI)	RT04801725396GZ	Initial node junction code
Latitude of initial node	43,748728	Latitude of the initial node expressed on the WGS 1984 geographic coordinate system

Longitude of initial node	11,233767	Longitude of the initial node expressed on the WGS 1984 geographic coordinate system
Destination node (NOD_FIN)	RT04801725397GZ	Destination node junction code
Latitude of destination node	43,749446	Latitude of the initial node expressed on the WGS 1984 geographic coordinate system
Longitude of destination node	11,235391	Longitude of the initial node expressed on the WGS 1984 geographic coordinate system
Flooring condition	0100	0100 = paved 0200 = Unpaved
Free flow rate	50	Free flow rate in km/h
Average speed	50	Average speed in km / h
Length	14	length of the element expressed in meters, -1 = Undefined
Lane type	1	Lane type: • free access to all vehicles = 1 • public transport only = 2 • cycle lane = 3
Lanes number	1	lanes number
Traffic Direction	FT	Direction of traffic: • blank = road section opened in both directions (default) • FT = road section opened in the positive direction (from • Junction NOD_INI to NOD_FIN junction) • N = road section closed in both directions • TF = road section opened in the negative direction (from Junction NOD_FIN to NOD_INI junction)

Based on the given length and the number of lanes the capacity of each road/edge was calculated. For the calculation of the capacities, the minimum travelling entity was assumed a pedestrian. Assuming that the ratio between cars and pedestrians is $1(car){\sim}10(pedestrians)$, we first calculate the cars capacity of each road. Given the fact that the average length of cars is approximately 5.0 meters (avg_car_length) the capacity of a road, regarding vehicles, can be computed by the following type:

$$capacity_{cars} = \left\lfloor \frac{\text{length} \cdot \text{lanes\_Num}}{\text{avg\_car\_length}} \right\rfloor$$

While the capacity of pedestrians can be calculated as:

$$capacity_{pedestrians} = 10 \cdot capacity_{cars}$$

3.5.1.2 Athens Map

Regarding the city of Athens, again, the road infrastructure was expressed in terms of nodes and edges, as it is displayed in Figure 4.

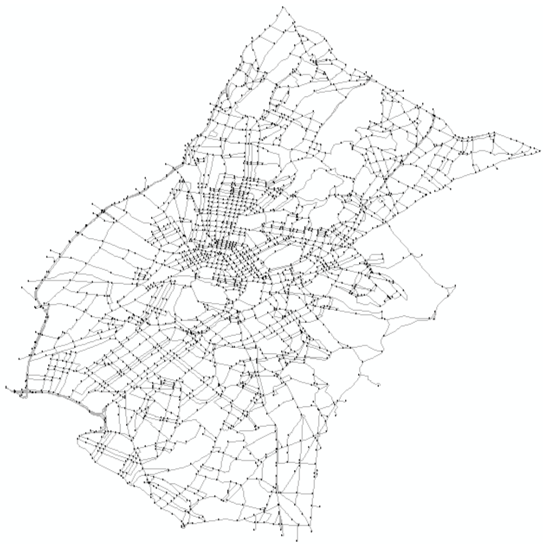


Figure 4: Map of Athens expressed in nodes and edges

For each road the following parameters was available, as they are described in Table 3.

Table 3: Available parameters of Athens's roads

Type	**Example**	**Description**
Road Element Unique identifier	10540	Unique node id (progressive value)
Initial Node (NOD_INI)	862762	Initial node junction code
Latitude of initial node	37,975857	Latitude of the initial node expressed on the WGS 1984 geographic coordinate system
Longitude of initial node	23,732544	Longitude of the initial node expressed on the WGS 1984 geographic coordinate system
Destination node (NOD_FIN)	862861	Destination node junction code
Latitude of destination node	37.975685	Latitude of the initial node expressed on the WGS 1984 geographic coordinate system
Longitude of destination node	23,734009	Longitude of the initial node expressed on the WGS

		1984 geographic coordinate system
Street name	ERMOU	Street name
FLENGTH	130	Link Length in specified direction (NOD_INI, NOD_FIN) in meters
TLENGTH	130	Link Length in specified direction (NOD_FIN, NOD_INI) in meters
FMODE	p	Mode[1] that uses the link[2] in specified direction (NOD_INI, NOD_FIN)
TMODE	p	Mode[3] that uses the link[4] in specified direction (NOD_FIN, NOD_INI)
FCLASS	c	Street Classification[5] in specified (NOD_INI, NOD_FIN)
TCLASS	c	Street Classification[6] in specified direction (NOD_FIN, NOD_INI)
FLANES	1	Number of Lanes[7] in specified direction (NOD_INI, NOD_FIN)
TLANES	1	Number of Lanes[8] in specified direction (NOD_FIN, NOD_INI)

The combined node pairs define unique directional links. The pair (NOD_INI, NOD_FIN) defines the link's digitized direction and the pair (NOD_FIN, NOD_INI) defines the opposite direction of the same link.
As with the Firenze's case, the capacity of each road can be calculated using the available data. Specifically, the length and the number of lanes can be used, as it is described in:

$$capacity_{cars} = \left\lfloor \frac{\text{length} \cdot (\text{FLANES} + \text{TLANES})}{\text{avg\_car\_length}} \right\rfloor$$

Based on the type of each road the speed limits were defined for each road in Table 4

Table 4: Average speeds for the roads of Athens according to their classification

Road Classification	Speed limits(km/h)

[1] c : Car (private Car and / or Bus, or Bus only), p: Pedestrian
[2] If mode "c" exists only in one direction and not in the opposite direction (FMODE="cp" and TMODE="p"), then this is an oneway link in the direction (NOD_INI, NOD_FIN).
[3] c : Car (private Car and / or Bus, or Bus only), p: Pedestrian
[4] If mode "c" also exists in TMODE (FMODE="cp" and TMODE="cp"), then this is a two way link.
[5] (F)reeway, (E)xpressway, (P)rimary, (S)econdary, (C)ollector.
[6] (F)reeway, (E)xpressway, (P)rimary, (S)econdary, (C)ollector.
[7] The width of the streets can be calculated, based on some assumptions and using the number of lanes, the mode and the street classification.
[8] The width of the streets can be calculated, based on some assumptions and using the number of lanes, the mode and the street classification.

Freeway	130
Expressway	120
Primary	90
Secondary	50
Collector	30

3.6 eDSS Back-End Modules

3.6.1 Network Representation Module

A common way to mathematically model & represent networks are *graphs (G)* that are composed by sets of *nodes (N)* and sets of *edges (E)*. In most cases dealing with graphs, the first question that has to be addressed is to decide which locations should be represented as nodes. In our case, considering the road network, each node represents a junction of two or more roads, while their inter-connections (roads) are represented as edges. The second critical question regarding the construction of the graph refers to the attributes describing the nodes and the edges. The main attribute, which describes the nodes during the evacuation, is the initial number of travellers located at the node at the beginning of the evacuation. In order to provide personalized evacuation plans, the role (i.e. rescuer, to be rescued) and the type (e.g. average, blind, wheelchair, vehicle, etc.) of each user are also important. Regarding the edges of the graphs several attributes can be used for describing them (e.g. length, width, travelling time, direction, name, classification, etc.). Thus, the description of the edges is depending on the available data that describes each road. In most cases, when dealing with graphs and routing problems, the most important parameters are the weights and the directions of each edge. Usually, in order to attach a weight to an edge, we consider the travelling time required to travel from one node to another. Although, travelling time information is difficult to be provided for the entire road network and especially for such a kind of constantly changing environment as the road network is during emergencies. So when the travelling time for all the edges are not available, the most applicable way of calculating them is by taking into account attributes such as the length, the width, the traffic, the condition of the road, the type of road, etc. over the speed of the travelling entity. Another important parameter that must be considered is the capacity of each road. When dealing with the routing of big groups of travellers the road network's capacity have to be adjusted and considered appropriately in order to avoid traffic congestions. After determining the nodes' & edges' attributes and modelling the road network as a directed, weighted & capacitated graph *G= (N,E)*, different algorithms can be applied (e.g. routing algorithms).

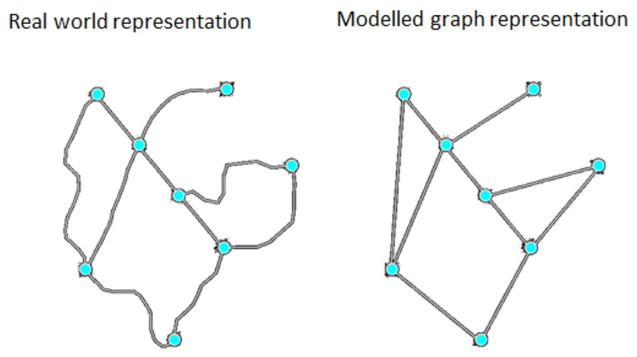


Figure 5: Real word and graph representation of an area

For the needs of the RESOLUTE project, the eDSS had to be adapted to the available data and to the RESOLUTE pilots. In this direction, a specific sub-module of the eDSS was developed, namely the *Graph-based representation module*. Since the two cities where the pilots will take place are, the city of Firenze and the city of Athens graph representation of their road networks had to be constructed. Due to this, map representations of the cities of Firenze and Athens where available, as shapefiles[9], from the RESOLUTE partners. The current version of the eDSS is able to receive road network related input data in form of CSV files. For this reason, all the needed information was extracted from shapefiles to CSVs with the use of GIS commercial tools. Furthermore, the spatial shape of each road was extracted, to be able to create a real world representation of the map for visualization purposes. Figure 5 displays at left the real world representation of an area, while at right the modelled graph in which the algorithm are applied.

3.6.2 Graph-based Simulator

By their nature, emergent events are continuously changing situations that may cause several problems to the normal function of the urban transport system, while in extreme cases they may cost the lives of many people. Due to this, and in order to be able to adapt the modelled graph to these challenging situations, the Graph-based Simulator (GS) was developed.

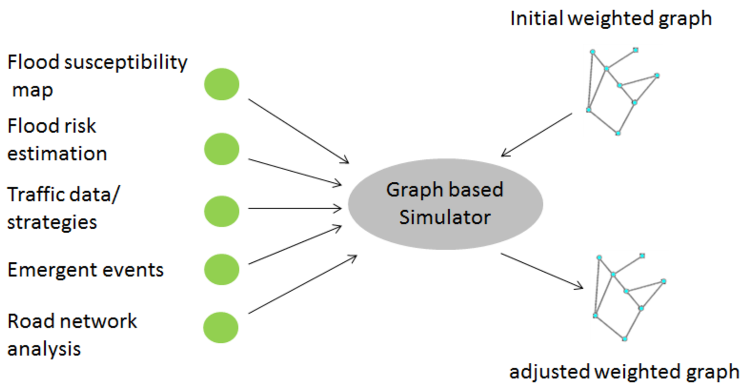


Figure 6: Graph based simulator

As shown in Figure 6, the GS receives various input data from various sources, processes them and adjusts the weights of the modelled graph accordingly. In this way, the eDSS's routing plans will not guide the evacuated population through traffic congested areas or areas that are in great danger. The GS is composed by two sub-modules, the *Contextual Simulation Module (CSM)* and the *Traffic Simulation Module (TSM)*, which are described in the following sections.

3.6.2.1 Contextual Simulation Module

The Contextual Simulation Module (CSM) is responsible for modelling the effect of the emergency depending on the kind of the disaster. More specifically, the eDSS can be informed of an emergent event through its interactive front-end, where the operator can define manually the area affected by an event, the severity of the event, as well as the kind of the event. Furthermore, the *Application framework* and specifically the *Weather Risk Estimation Module* can be used in order to update the values of the graph. In this way, if the operator requests for evacuation planning, the computed evacuation routes will avoid to guide people through the areas in danger.

[9] The shapefile format is a popular geospatial vector data format for geographic information system (GIS) software.

3.6.2.1.1 Marking a hazardous Area

The Front-end of the eDSS provides an easy way for the operator to interact with the computation of the evacuation routes. Being informed of a hazardous event the operator can define over the map a circle representing the limits of the affected area. He can also add information regarding the severity of the event and other additional information. Such actions can lead to the update of the weights of the graph, so as to weights of roads within the specified area to be adjusted, as Figure 7 displays.

The update of the weights is based on the affection of a 2d Gaussian distribution. The new weights of the graph are described by the following equation:

$$\boldsymbol{w_{new} = w_{old} * (1 + g)}$$

Where w_{new} is the new weight of the road, w_{old} the previous weight of the road and *g* the affection of the event based on the Gaussian distribution as it is calculated by using the following equation:

$$\mathbf{g} = \boldsymbol{A} * \mathbf{exp}\left(-\left(\frac{(\boldsymbol{x}-\boldsymbol{x_0})^2}{2\boldsymbol{\sigma_x}^2} + \frac{(\boldsymbol{y}-\boldsymbol{y_0})^2}{2\boldsymbol{\sigma_y}^2}\right)\right)$$

Where A is the amplitude and equals to the severity of the event (an integer number between 1(low severity) and 10(high severity)), defined by operator, σ_x and σ_y are the x and y spread of the blob $\sigma_x = \sigma_y = R/2$,R the radius of the marked area , $(x_{0,}y_0)$ the coordinates of the centre of the hazardous event, and (x, y) the coordinates of the road segment's closest node to the centre of the event. Due to the fact that the radius is calculated in meters while the x,y are geographical coordinates (y: latitude, x: longitude) and the numerator express the distance of two points the equation can be written as:

$$\mathbf{g} = \boldsymbol{A} * \mathbf{exp}\left(-\frac{\boldsymbol{d}^2}{2\left(\frac{\boldsymbol{R}}{2}\right)^2}\right)$$

Where d the distance of the two points calculated using the haversine formula[24].

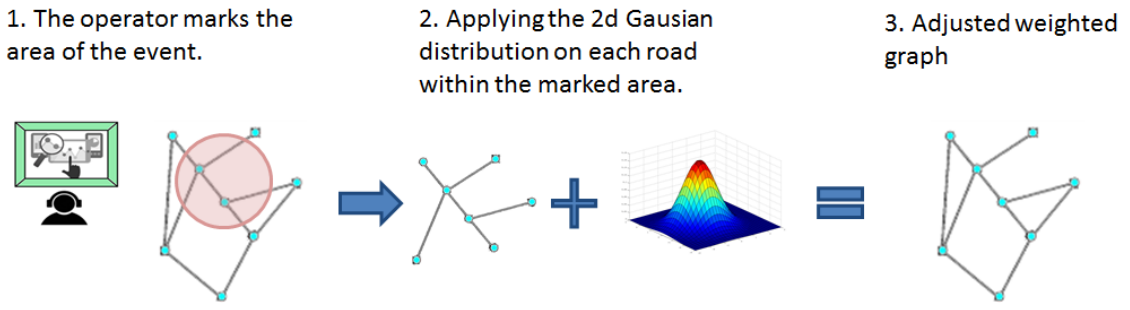

1. The operator marks the area of the event.
2. Applying the 2d Gausian distribution on each road within the marked area.
3. Adjusted weighted graph

Figure 7: Graph weights adjustment regarding the emergent event

3.6.2.1.2 Weather Risk Estimation from Application Framework

Moreover, the Application Framework can provide useful information regarding the weather severity and especially the flood hazard. The provided flood hazard is a number value between 0.0 and 1.0. The latter combined with extracted information from the susceptibility flooding map (provided by UNIFI) can be used in order to update the weights of the most susceptibility to floods road segments.

The provided susceptibility-flooding map contains information regarding the most vulnerable areas within the city of Firenze. The data of the flood map and of the road network was pre-processed such as the most susceptible roads to be extracted. By fusing the extracted information about the susceptible road and the risk estimation of flood the adjustment of the road weights are given following the equation. Figure 8 shows the whole procedure.

$$w_{new} = w_{old} * (1 + flood\_risk)$$

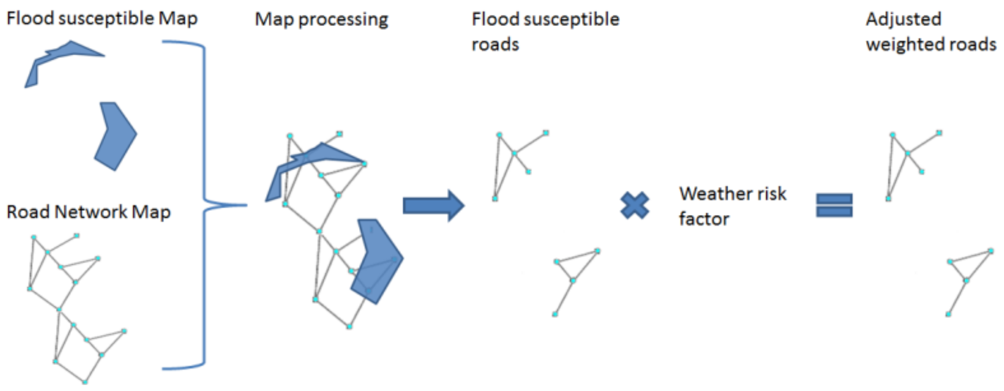


Figure 8: Graph affection based on flood risk estimation

3.6.2.1.3 Network Vulnerability Estimation from Application Framework

An analysis aimed at identifying the possible vulnerabilities of the UTS, according to the current situation, is performed by the Network Analysis module, placed within the “Mission critic layer” and, more specifically, into the “Application framework”. It retrieves the basic input data from the “Data management layer”, in particular structural and service information about the UTS, and its analytical functionalities can be accessed via the “Integration ESB”. Results consist of a list of the most critical components of the UTS and the analysis is performed on demand (i.e. performing a request to the corresponding web service).

An internal graph-based representation of the UTS is obtained starting from the structural and service information, but the graph is dynamically modified according to modifications to the UTS, which can occur overtime, allowing for an estimation based on the current condition of the UTS.

Finally, the output of the network analysis module (i.e. a list of nodes and edges identified as “critical” with respect to a set of measures computed on the current status of the UTS) can be then suitably interpreted by the module, which has performed the analysis request. Such a service-oriented paradigm allows for using the network analysis module in different contexts (generating a graph model for public transportation network, road network, rail network, etc.). Furthermore, and more importantly, it allows implementing visualization, as well as decision support, just according to Figure 9. More details are provided in the deliverable D4.3 “Application Framework”, where the design, development and initial validation on the available sets of data have been reported. In particular, preliminary results on network analysis have addressed the public transportation networks in Florence and the Attika region: the most critical stops/stations (i.e., nodes of the graph-based model) and connections (i.e., edges of the graph-based model). The following section reports the basic data structure of the results provided by the network analysis.

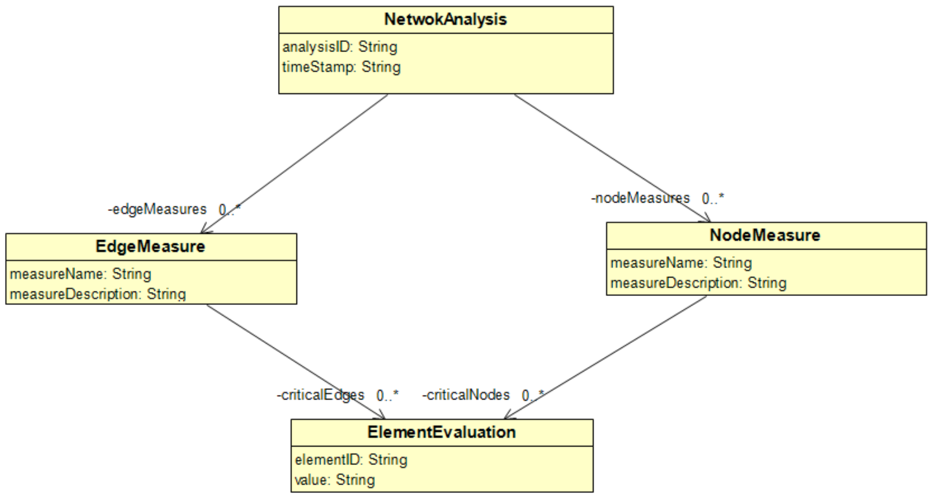
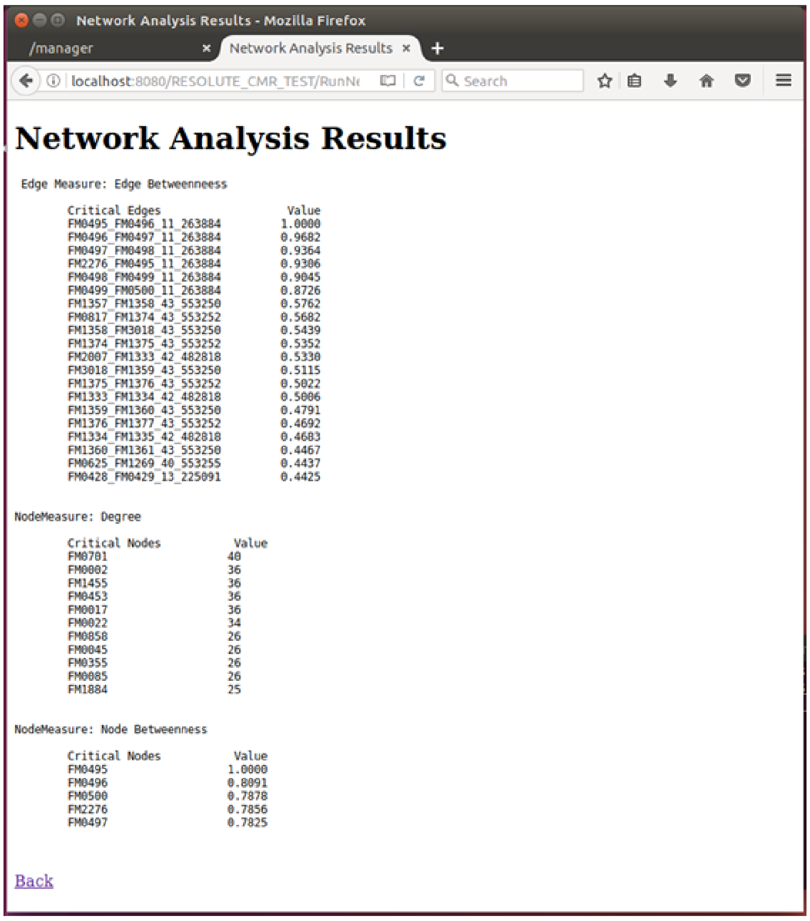


Figure 9 Data model of the network analysis result: message exchanged through the ESB follow this model.

Furthermore, as an example, the following Figure 10 reports the “basic” output provided by the network analysis service. The figure shows the critical nodes and edges according to some graph-based measures and the current condition of the public transportation network. All the communications (i.e. request and response) go through the ESB. The web page is just in charge to visualize – as text – the results of the analysis, in order to show how “simple”, or “basic” is this information and how it can be easily integrated with other data/consideration for more sophisticated – for instance geo-localized – visualization.

The eDSS communicates with the network analysis module, retrieves its output and provides the latter to the eDSS UI in order to visualize the most critical parts of the road network as a new layer over the map.


Network Analysis Results - Mozilla Firefox
/manager
Network Analysis Results
localhost:8080/RESOLUTE_CMR_TEST/RunNe
Search

Network Analysis Results

```
 Edge Measure: Edge Betweenneess

        Critical Edges              Value
        FM0495_FM0496_11_263884     1.0000
        FM0496_FM0497_11_263884     0.9682
        FM0497_FM0498_11_263884     0.9364
        FM2276_FM0495_11_263884     0.9306
        FM0498_FM0499_11_263884     0.9045
        FM0499_FM0500_11_263884     0.8726
        FM1357_FM1358_43_553250     0.5762
        FM0817_FM1374_43_553252     0.5682
        FM1358_FM3018_43_553250     0.5439
        FM1374_FM1375_43_553252     0.5352
        FM2007_FM1333_42_482818     0.5330
        FM3018_FM1359_43_553250     0.5115
        FM1375_FM1376_43_553252     0.5022
        FM1333_FM1334_42_482818     0.5006
        FM1359_FM1360_43_553250     0.4791
        FM1376_FM1377_43_553252     0.4692
        FM1334_FM1335_42_482818     0.4683
        FM1360_FM1361_43_553250     0.4467
        FM0625_FM1269_40_553255     0.4437
        FM0428_FM0429_13_225091     0.4425


NodeMeasure: Degree

        Critical Nodes         Value
        FM0701                40
        FM0002                36
        FM1455                36
        FM0453                36
        FM0017                36
        FM0022                34
        FM0858                26
        FM0045                26
        FM0355                26
        FM0085                26
        FM1884                25


NodeMeasure: Node Betweenness

        Critical Nodes         Value
        FM0495                1.0000
        FM0496                0.8091
        FM0500                0.7878
        FM2276                0.7856
        FM0497                0.7825
```

Back

Figure 10 A sample of response from the network analysis module

3.6.2.2 *Traffic simulation Module*

The Traffic Simulation Module (TSM) is responsible for applying changes to the weights of the graph regarding the traffic volume of the road network. The traffic sensors that are distributed within the city of Firenze provide real time data to the Data Layer. The TSM communicates with the Data Layer and receives information about the location and the measures provided by each sensor (e.g. vehicle flow, vehicle concentration, avg. speed, etc.). The TSM utilizes this information by setting the average speed of the roads according to the retrieved by the sensors average speed. In this way, the weights of the roads are directly connected will real time traffic data and are adjusted accordingly.

Except from the traffic sensors and the Data Layer, the TSM communicates, through the ESB, with the UTM DSS. The UTM DSS can provide real-time information considering the condition of the UTS based on data retrieved by its actuation channels. Thus, the UTM DSS can suggest the blockage of a road or the blockage of an entire area due to critical traffic events. This kind of information can be retrieved and be handled by the TSM in order to adjust the structure of the graph to avoid producing evacuation routes that include the blocked roads. The opposite procedure (unblock a road) is also handled by the TSM after being triggered by the UTM DSS.

3.6.3 Evacuation Manager

3.6.3.1 *Evacuation Planning Algorithms*

Due to the analysis of Section 0, the eDSS core algorithm has been based on the CCRP.

eDSS evacuation algorithm

```
Input:
  1) G=(N,E): a graph G with a set of nodes N and a set of edges E;
     Each node $n \in N$ has:
        Initial_Node_Occupancy(n): non-negative integer
     Each edge $e \in E$ has two properties:
        Maximum_Edge_Capacity(e): non-negative integer
        Travel_Cost(e): non-negative integer
  2) S: set of source nodes, $S \subseteq N$;
  3) D: set of destination nodes, $D \subseteq N$;
Output: Evacuation plan: Routes with schedules of evacuees on each route
Method:
Pre-process network: add super source $s_0$ node to network, link $s_0$ to each source nodes
with an edge which Maximum_Edge_Capacity()=$\infty$ and Travel_Cost()=0;

While any source node $s \in S$ has evacuee do {
  Find route R $\langle n_0, n_1, \dots, n_k \rangle$ with time schedule $\langle t_0, t_1, \dots, t_k \rangle$using one generalized
  shortest path search from super source $s_0$ to all destinations, (where $s \subseteq S$, $d \subseteq D$,
  $n_0 = s$, $n_k = d$) such that R has the earliest destination arrival time among routes
  between all (s,d) pairs, and
    Available_Edge_Capacity($e_{nini+1}$, $t_i$)>0 $\forall i \in \{0,1,..k-1\}$,

    flow = min(number of evacuees still at source node s,
               Available_Edge_Capacity($e_{nini+1}$, $t_i$), $\forall i \in \{0,1,..k-1\}$ )
    for i = 0 to k – 1 do{
        Available_Edge_Capacity($e_{nini+1}, t_i$) reduced by flow;
    }
}
Output evacuation plan
```

In order the algorithm to be able to calculate the routing plans the knowledge of the travel cost of each road segment (edge) is required. The travel cost of each road segment is computed as the upper bound of the weight of the road segment, as it is affected and defined by the graph affection modules, over the average speed of each user or group of users for the particular road and is described by the following equitation:

$$Travel\_Cost(e) = \lceil w_g(e)/s \rceil$$

Where '$Travel\_Cost(e)$' is the final travelling cost of the road segment, '$w_g(e)$' is the road's weight according its length and the affection of the graph affection modules, and 's' is the speed of the traveler according to the type of the road.The system considers two main groups of travellers, the travellers-vehicles and the pedestrian travellers. Regarding the first group (vehicles) their speeds was defined considering the provided speeds of its road. In case of Firenze the average speed of each road was available from the input map, while in case of Athens the speed limits of each type of road was available as described in Table 4. Considering the second group (pedestrians) two factors can affect their speeds, the type of the road and the profile information of each user. Based on its profile a pedestrian can be described by the following vector:

$$\bar{d} = \{d_1, d_2, \dots, d_m\} \text{ ,}$$

where, $d_i,\ i \in \{1, \dots, m\}$ describes the m different types of disabilities (e.g. no disability, vision impairments, walk disabilities, etc.) for a pedestrian. The different types of roads can be also represented as a vector:

$$\bar{t} = \{t_1, t_2, \dots, t_n\} \text{ ,}$$

where $t_j, j \ \in \{1, \dots, n\}$ describes different types of roads. Based on the types of the users' disabilities and the types of roads a table $V \in \mathbb{R}^{M\times N}$ is constructed where V_{ij}: the average speed of a user of type $d_i$in road of type t_j. The first column of the table $V_{1j}$was set to contain the average speeds of a normal user (user without disabilities) for each type of road.

Next, based on table Vwe compute the normalized table $F \in \mathbb{R}^{M\times N}$,

$$F_{ij} = \frac{V_{ij}}{V_{1j}}$$

WhereF_{ij}: the value of the speed of a $d_i$disabled user over the speed of a normal user for the road of typet_j.
We set a user u and a road r, where $u \in [0,1]^M$, M:the number of disabilities, and describes the percentage of each disability of the vector $\bar{d}$., and r is described by the vector $\bar{t}$ so as to $r \in [0,1]^M$ where Ndifferent types of roads. Thus the final speed $V_{ur}$of the user u over the road r can be calculated from the following type:

$$V_{ur} = V_{1j} \cdot \prod_{i=1}^{M} [1 - (1 - F_{ij})u_i]$$

3.6.3.2 eDSS Modes & Priorities

The Evacuation Manager can provide routing plans concerning three different functional modes, namely: **group-wise evacuation, personalized** evacuation, and collaborative rescue.

The group-wise evacuation refers to the evacuation of groups of crowds from different locations over the graph. By identifying users with common characteristics (e.g. speed), the system creates groups of similar users and provides evacuation routes to each one of them. By utilizing the CCRP algorithm, the eDSS secures that all groups reach at the safe points without conflicts. Usually, the formatted groups regard two kinds of groups, the vehicles and the normal pedestrians. It should be mentioned that the guidance provided to vehicle-users respects the traffic directions and also avoids guiding the vehicles through pedestrian or inaccessible by vehicle roads. Information regarding the travelling way (by foot or by vehicle) of each user is retrieved by the ESSMA.

On the other hand the **personalized** evacuation mode treats the most vulnerable pedestrian users, providing to them evacuation routes that fits best to each user's profile. Through its communication with the ESSMA the eDSS retrieves information regarding the users' profiles and their disabilities. Within the RESOLUTE the eDSS concerns two kinds of disabilities for each pedestrian user, vision and walk disabilities. If an ESSMA user has walk or vision disabilities, he can define it to his profile information. Based on this, the user's vector is constructed and the users speed is calculated for each type of road based on the aforementioned equation. The definition of the table $V$was based on the works of [17] [18] that present in detail the average speeds of vulnerable people concerning their kind of vulnerability and the type of road. Thus, following the personalized evacuation mode adjusts the weights of the graph accordingly and provides them with the most fitted, considering their profile, evacuation routes. For instance, if the type of a road segment describes stairs the weight for a normal, a blind and a wheelchair user will be different preventing users with disabilities to follow this path. Finally, the **collaborative rescue** planning is a dynamic functionality that may occur during the evacuation phase. People being trapped, injured or in need of help is not a rare phenomenon during emergencies. These persons may be persons that have received evacuation routing and are unable to reach at the safe points. Concerning the above, the collaborative rescue tries to utilize all the available sources in order to help the unable persons to reach at the safe points. In order to achieve this it utilizes its communication with the ESSMAs of each user. Through the ESSMA, people that need help can send an SOS message to the evacuation responsible. The eDSS operator is alerted about these cases through the eDSS UI (section3.7). The operator can then send an alert message to all the voluntary helpers and ask them if they are willing to help. After gathering the helpers' availability the operator can request from the eDSS to calculate the collaborative rescue action plan. Similar to the group-wise and the CCRP the following algorithm is used for the computation of the collaborative rescue plan.

Algorithm (collaborative rescue)

Input:

```
1) G=(N,E): a graph G with a set of nodes N and a set of edges E;
   Each node $n \in N$ has two properties:
     Initial_Node_Occupancy(n): non-negative integer
   Each edge $e \in E$ has two properties:
     Maximum_Edge_Capacity(e): non-negative integer
     Travel_Cost(e): non-negative integer
2) S: set of source nodes in which the voluntary helpers are initially located, $S \subseteq N$;
   Each node $s \in S$ has one property:
     num_of_available_helpers(s): non-negative integer
3) D: set of destination nodes in which the users that needs help are initially located, $D \subseteq N$;
   Each node $d \in D$ has one property:
        num_of_ helpers_needed(d): non-negative integer
Output: Collaborative rescue plan: Routes with schedules of helpers on each route
Method:
Pre-process network: add super source $s_0$ node to network, link $s_0$ to each source nodes
with an edge which Maximum_Edge_Capacity()=$\infty$ and Travel_Cost()=0;

While (any source node $s \in S$ has helpers && any destination node need help) do {
  Find route R $\langle n_0, n_1, \ldots, n_k \rangle$ with time schedule $\langle t_0, t_1, \ldots, t_k \rangle$using one generalized
  shortest path search from super source $s_0$ to all destinations, (where $s \subseteq S$, $d \subseteq D$,
  $n_0 = s$, $n_k = d$) such that R has the earliest destination arrival time among routes
  between all (s,d) pairs, and
    Available_Edge_Capacity($e_{nini+1}$, $t_i$)>0 $\forall i \in \{0,1,..k-1\}$,

    flow = min(number of available helpers still at source node s,
               number of helpers still needed at destination node d,
               Available_Edge_Capacity($e_{nini+1}$, $t_i$), $\forall i \in \{0,1,..k-1\}$);

   for i = 0 to k – 1 do{
     Available_Edge_Capacity($e_{nini+1}$, $t_i$) reduced by flow;
   }
}
Output collaborative rescue plan
```

The Evacuation Manager calculates the evacuation routes in a **prioritized** way. The system concerns the most vulnerable people as those with the highest priority; the normal pedestrian users have the second priority, while the vehicle users have the lowest one. In this way the system ensures that the shortest paths will be provided to the most vulnerable people, in order to evacuate them as efficient as possible and with the lowest effort from their side.

3.7 eDSS User Interface

Except from the back-end of the eDSS also a front-end was developed in order to facilitate the urban transport authorities to easily take advantage of the back-end functionalities. So, the eDSS provides a control room software which's target users is a person (or a team of persons) responsible for supervising the evacuation procedure. Considering this, the front-end of the eDSS was designed following design standards so as to be user-friendly and to offers an easy way for exploiting the functionalities of the eDSS back-end. For the development of the UI, the mockups provided by Fraunhofer and presented in D5.2 were followed. In this section the eDSS UI is presented.

The functionalities of the eDSS's are listed below:

a) Monitor on the map (e.g. users need help, ESSMA users' location, bus location, etc.)
b) Affect the structure and the travelling weights of the graph representing the city's road network
    i. Exclude/ re-include a road
    ii. Mark areas that are in danger over the map
c) Manage routing procedures
    i. Manage the evacuation procedure
        i. Initiate evacuation
        ii. Select users to be guided

            iii. Add extra users
            iv. Request for evacuation planning
            v. Send the calculated evacuation plan to the CDM for approval
            vi. Send personal guidance, considering the calculated evacuation plan, to each involved user
        ii. Manage the collaborative rescue procedure
            i. Monitor for ESSMA users that need help
            ii. Send message to the voluntary helpers asking for their availability to help
            iii. Receive the responses of the voluntary helpers containing their availability
            iv. Request for the calculation of the collaborative action plan
            v. Send the guidance to each involved user
    d) Communicate with ESSMA users
        i. Chat communication
        ii. Timeline (e.g. post news/updates at the timeline)
    e) Manage settings & receive notifications

3.7.1 Monitoring activities

Figure 12 displays the starting page of the UI, which is displayed to the operator when the latter accesses the eDSS. The implemented UI follows the proposed mockups that were presented in D5.2 and is displayed in Figure 11. The starting page is constructed mainly by four components: i) the map, ii) a sidebar at the left of the page, iii) a sidebar at the right of the page, and iv) a bar at the top of the page. Each one of the components serves different purposes and all together, they aim at giving the operator an overall view of the emergent situation, helping him/her to manage it and facilitating him/her to communicate with the citizens. The view of the implemented UI is very close to the provided mockups, while the right part of the UI that is missing from Figure 11, follows alternative suggestions of D5.2.

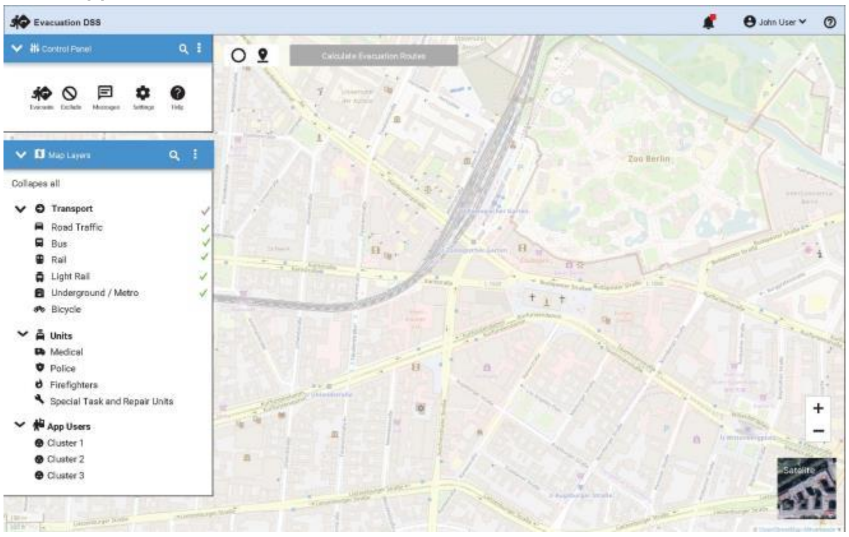


Figure 11. Mock-up of the eDSS UI starting page

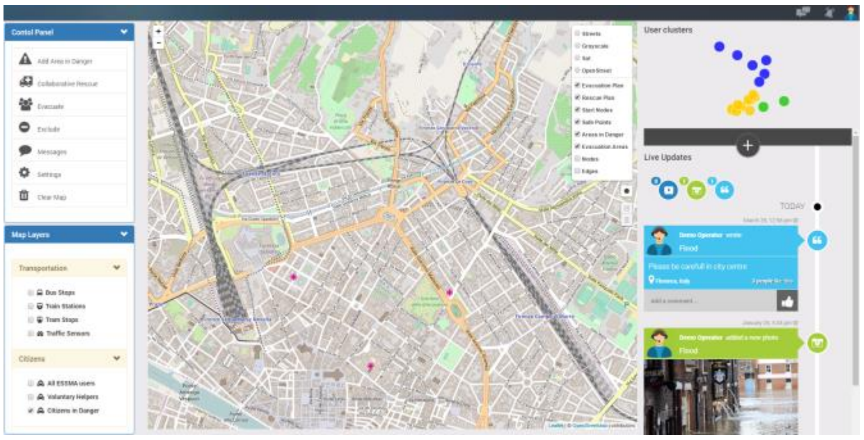


Figure 12: eDSS UI starting page

As defined in the Grant Agreement of the RESOLUTE project, the CRAMMS was meant to integrate a wide variety of Decision Support modules into a unified system. This has been achieved through three (3) different types of integration: Visual, Data and System integration.

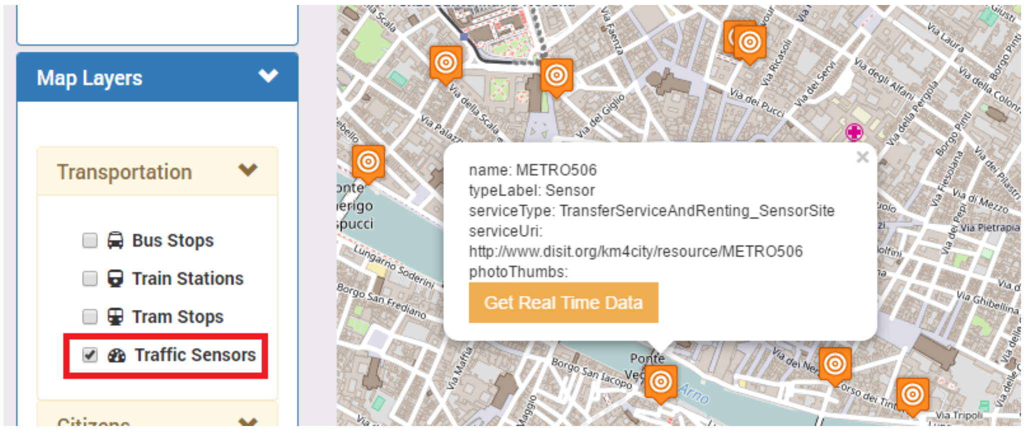
In this respect, the following table shows the type of data derived from the different modules developed within the activities of the project. In particular, Table 5 contains the information that can be displayed over the map for the cities of Firenze and Athens, a short description of them, their source and the way that it is represented. The source of the information can be the data management layer, the ESSMA apps, the back-end of the eDSS or the result of the interaction of the operator with the front-end.

Table 5: Information displayed on the map

Data	Description	Representation	Source	Cities
Bus stops	Firenze's bus stops	🚏	Data Layer	Firenze
Train stations	Firenze's train stations	🚆	Data Layer	Firenze
Tram stops	Firenze's tram stops	🚊	Data Layer	Firenze
Traffic sensors	Traffic sensors placed at various locations of the road network. More information regarding the values of the sensors can be displayed by clicking on the sensors.	◎	Data Layer	Firenze
Buses locations	Firenze's buses' real time locations	🚌	Data Layer	Firenze
All ESSMA users	Current locations of each ESSMA user. By clicking on a user, further information regarding the user can be displayed (e.g. special need, etc.)	🚹	ESSMAs	Athens, Firenze
Voluntary helpers	Current locations of **all** ESSMA users that are declared as voluntary helpers. By clicking on a user more information, regarding the user can be displayed (e.g. special need, etc.)	🚹	ESSMAs	Athens, Firenze
	Available voluntary helpers	🚹	ESSMAs	Athens, Firenze

	Occupied voluntary helpers (e.g. users that have already received an action plan)	🟩	eDSS back-end	Athens, Firenze
Citizens in danger	Current locations of ESSMA users that are in danger. By clicking on a user, further information regarding the user can be displayed (e.g. special need, etc.)	🟥	ESSMAs	Athens, Firenze
	Served users (in danger). Users that have made a request from help and helper(s) were planned to help them.	□	ESSMAs	Athens, Firenze
Evacuation Routes	Routes for guiding the citizens (ESSMA users or not) at the safe point.	Colourful polylines	eDSS back-end	Athens, Firenze
Danger Areas	Red circles over the map declaring areas in danger.	Red circle	eDSS front-end	Athens, Firenze
Evacuation Areas	Blue circles over the map declaring areas to be evacuated.	Blue circle	eDSS front-end	Athens, Firenze
Nodes	Nodes of the road network infrastructure. By clicking on them the users can set them as safe points or starting nodes	📍	eDSS back-end	Athens, Firenze
Edges	Edges of the road network infrastructure. By clicking on them a pop up containing information about the selected road is displayed, as well as options to include or exclude a road from the routing planning procedures.	Black polylines	eDSS back-end	Athens, Firenze
Starting Nodes	Nodes containing people. Set by the operator for including them in the calculation of the evacuation planning.	🟦	eDSS front-end	Athens, Firenze
Safe Points	Safe points. Set by the operator. Locations where the evacuated crowd will be guided to.	🟨	eDSS front-end	Athens, Firenze

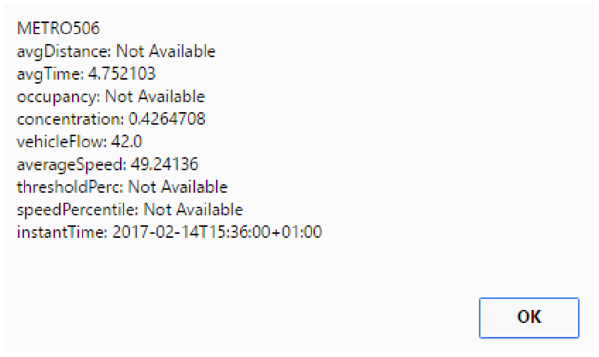
By using the “Map Layer” (sub-component of the left sidebar) the operator can choose which information will be displayed over the map. As in the majority of the online services using maps, markers are used for representing points over the map. For instance, Figure 13 displays the location of the traffic sensors located at the roads of Firenze. Moreover by clicking on a marker (traffic sensor) additional information can be retrieved (e.g. name, vehicle flow, etc.)



name: METRO506
typeLabel: Sensor
serviceType: TransferServiceAndRenting_SensorSite
serviceUri:
http://www.disit.org/km4city/resource/METRO506
photoThumbs:

Figure 13. Illustration of the selected, from the Map Layer, information on the map. By clicking on a marker more information regarding the represented item can be retrieved.

In the traffic sensor example, the operator can get the real time Traffic Data by clicking the button "Get Real Time Data" as it is shown in Figure 14



```
METRO506
avgDistance: Not Available
avgTime: 4.752103
occupancy: Not Available
concentration: 0.4264708
vehicleFlow: 42.0
averageSpeed: 49.24136
thresholdPerc: Not Available
speedPercentile: Not Available
instantTime: 2017-02-14T15:36:00+01:00
```

Figure 14. Real Time Data of Sensor

The Figure 15 offers a way for visualizing the ESSMA users in visual clusters. Users with similar behaviour are positioned close to each other. The information regarding the clusters is retrieved by the user-profiling module of the application framework and the data of users' movements that creates these clusters are retrieved using the same API that the ESSMA uses, in order to feed the system with this information. Through this widget of visual clusters, the operator is able to illustrate users with common movement behavior.

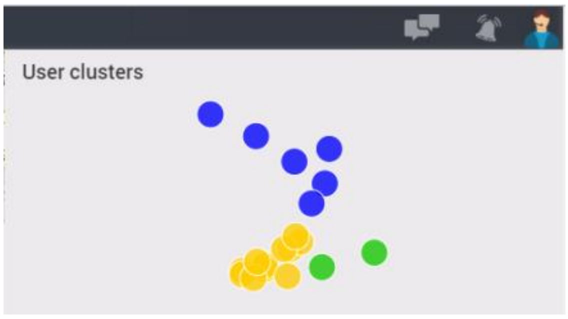


Figure 15. Widget displaying the ESSMA users in visual clusters based on their common attributes.

The operator can clear the map by using the "Clear map" functionality of the control panel. By selecting this, the map is returned to its initial condition. All the added or requested, by the operator, actions (e.g. adding events, setting start/end nodes, evacuation plan, etc.) are cleared.

Finally yet importantly, it is worth mentioning that the map has standard interaction features known from other popular online maps (e.g. zoom in, zoom out by scrolling over the map, move the map using the mouse, etc.).

The operator can choose among four different map layers as a basis of the map (i.e. street, satellite, openStreetMap, grayscale) according to their preferences as shown in the following table.

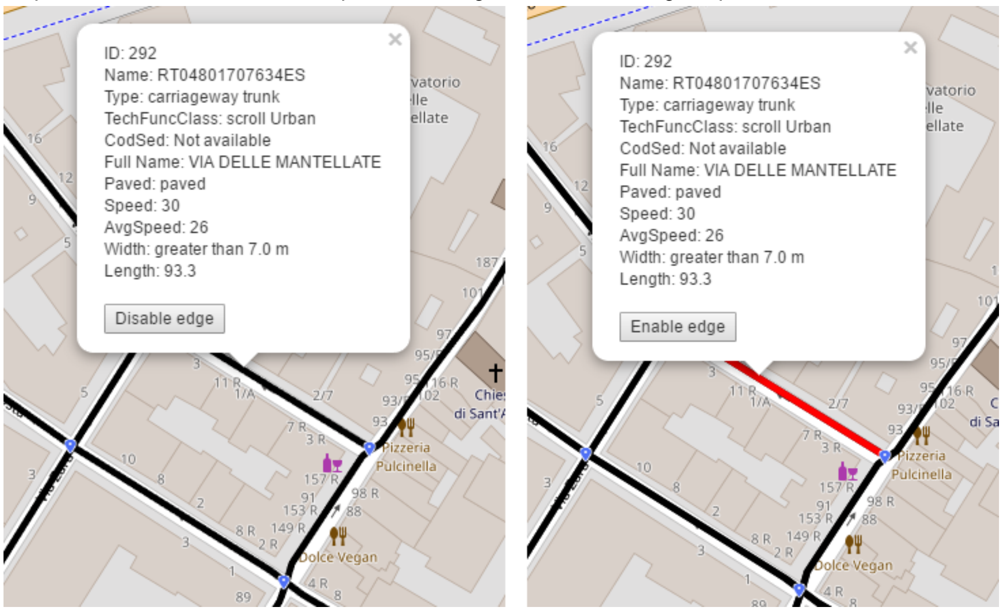
Open Street Map view	Satellite view
Grayscale view	Street view

Table 6. The different available map views of the eDSS UI interactive map

3.7.2 Human Machine Interaction services of the eDSS

3.7.2.1 *Exclude/Include a road*

Through the eDSS's front-end, the operator can affect the structure of the road network's formatted graph. By selecting the "Exclude" option form the control panel at the left sidebar (Figure 12) a layer that contains all the available edges appears. By clicking on an edge, more information is appeared, regarding the particular road segment. Additionally, a button that allows the operator to include or exclude a road from the procedure of calculating the routes that has to be followed. By clicking at the "Disable edge" as Figure 16 displays the operator can exclude a road. The excluded roads are highlighted as red. The operator can re-include the road by following the opposite procedure and select for the particular edge the "Enable edge" option.


ID: 292
Name: RT04801707634ES
Type: carriageway trunk
TechFuncClass: scroll Urban
CodSed: Not available
Full Name: VIA DELLE MANTELLATE
Paved: paved
Speed: 30
AvgSpeed: 26
Width: greater than 7.0 m
Length: 93.3

ID: 292
Name: RT04801707634ES
Type: carriageway trunk
TechFuncClass: scroll Urban
CodSed: Not available
Full Name: VIA DELLE MANTELLATE
Paved: paved
Speed: 30
AvgSpeed: 26
Width: greater than 7.0 m
Length: 93.3

Figure 16: Road network representation and information display of a specific road.

3.7.2.2 *Mark Area in danger*

There is also the "Add Area in Danger" functionality of the control panel that can affect the evacuation procedure. This functionality regards the marking of an area as a hazardous area. When the operator marks an area as hazardous/dangerous the information is sent to the back-end and specifically to the CSM (described in section 3.6.2.1.1), resulting thus to the update of the travelling weights of the graph. Figure 17 displays how the "Add Area in Danger" feature can be utilized by the operator.

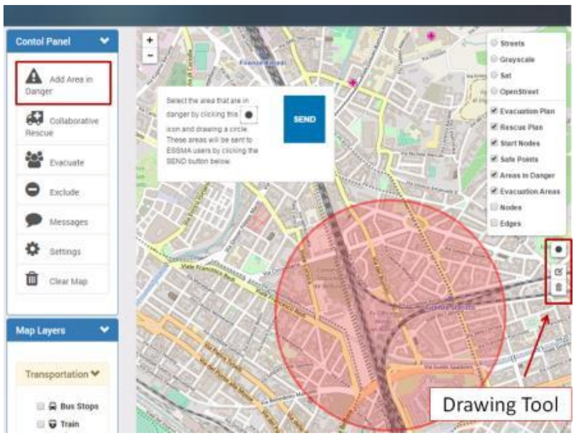
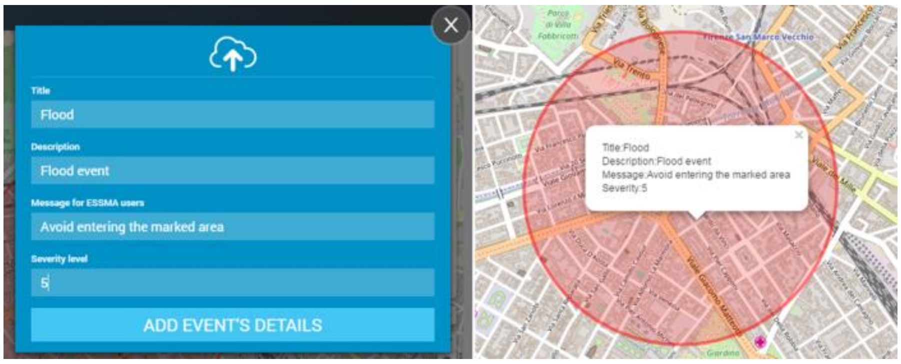

Select the area that are in danger by clicking this ● icon and drawing a circle. These areas will be sent to ESSMA users by clicking the SEND button below.

Figure 17: "Add Area in Danger" operator's view.

By clicking on it, the operator can use the drawing tool in order to mark an area as hazardous. By using the drawing tool, the operator can also edit or delete a previously marked area. The operator can “inform” the back-end about the new marked areas by clicking the send button.
Except from affecting the evacuation and the collaborative rescue, planning the “Add Area in Danger” can be used in an informative way for the ESSMA users avoiding them passively to approach the affected area. When the operator sets the danger areas and push the “SEND” button, the ESSMA users are informed via the app for the location of the event coupled with additional information about the event. The additional information describing the emergent situation (i.e. title, description, severity, message to ESSMA users) can be added by right clicking on an area. Then a form is displayed as Figure 18 and additional information about the event can be added by the operator.


Title
Flood
Description
Flood event
Message for ESSMA users
Avoid entering the marked area
Severity level
5
ADD EVENT'S DETAILS

Title:Flood
Description:Flood event
Message:Avoid entering the marked area
Severity:5

Figure 18: Set additional information about an Area in Danger (left). View of the information by clicking over the marked area (right).

3.7.3 Routing Procedures Management

3.7.3.1 *Evacuation Procedure*

Except from monitoring purposes the user can use the eDSS UI for managing the evacuation procedure. The following figure demonstrates the sequence diagram of the evacuation procedure.

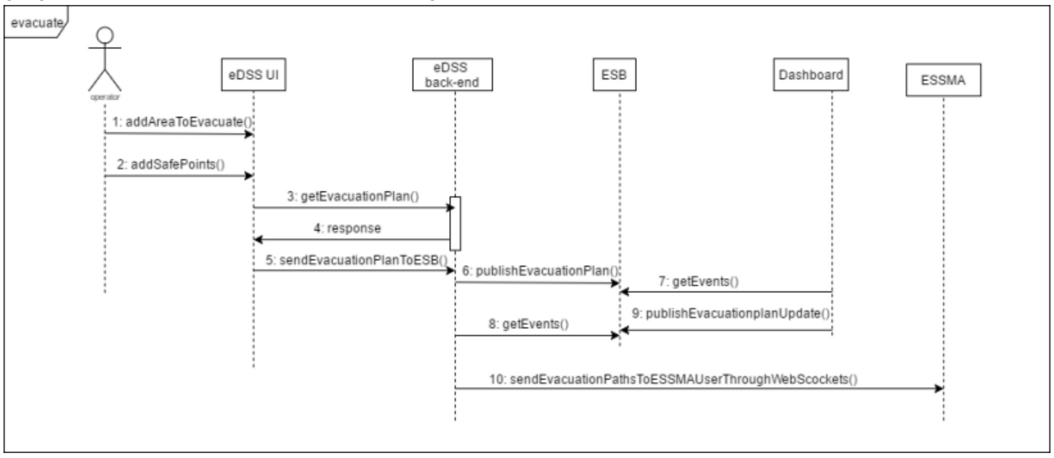
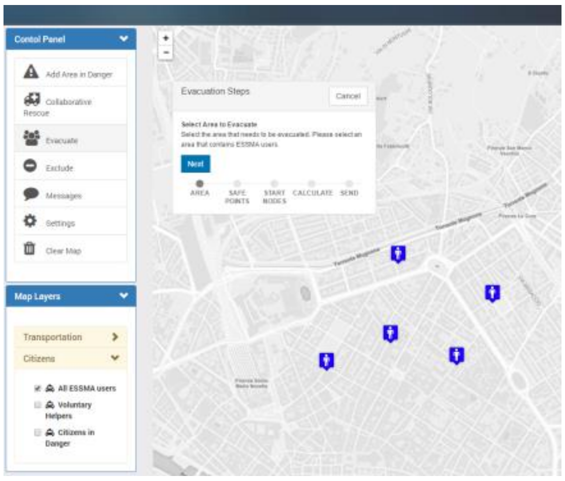


Figure 19. Sequence diagram of the evacuation procedure.

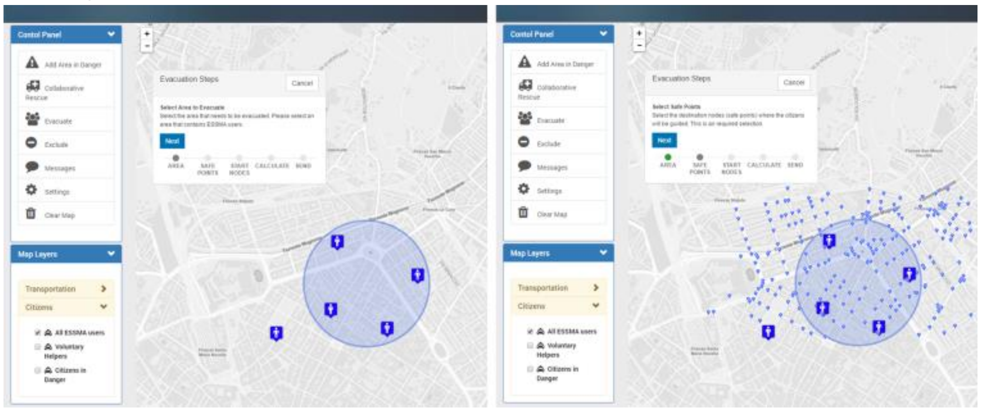
By selecting the “Evacuate” action, from the control panel component, the evacuation procedure is initiated and the application shows the window that is shown in Figure 20. This is a systematic guide for the operator in order to manage the evacuation process properly. Except from the systematic guide, the layer containing all the ESSMA users and their location is displayed.



Select Area to Evacuate
Select the area that needs to be evacuated. Please select an area that contains ESSMA users.

Figure 20: The evacuation procedure is initiated by the operator

As the evacuation guide instructs, at first the operator needs to select the area that will be evacuated.

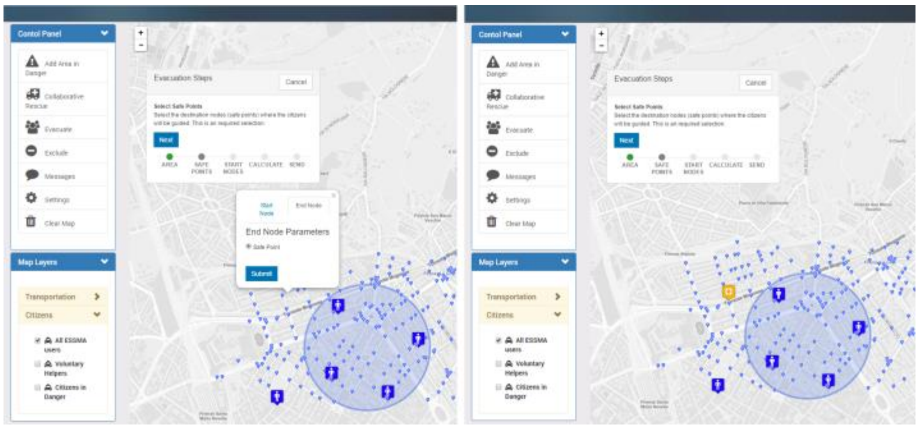


Select Area to Evacuate
Select the area that needs to be evacuated. Please select an area that contains ESSMA users.

Select Safe Points
Select the destination nodes (safe points) where the citizens will be guided. This is an required selection.

Figure 21: Evacuation area marked by the operator

All the ESSMA users within the marked area will be treated as users to-be-evacuated. After marking an evacuation area the operator has to define the safe points where the groups of people will be guided to, and thus the layers containing the nodes of the road network is displayed (Figure 21).



Select Safe Points
Select the destination nodes (safe points) where the citizens will be guided. This is an required selection.

Figure 22: Safe points defined by the operator

By clicking on a node, the operator can define a node as a safe point. The defined safe points are displayed on the map by an orange marker. Figure 22 displays the described procedure for setting a safe point over the map. Except from the ESSMA users the operator can, optionally, define extra start nodes (custom users to-be-evacuated) coupled with the number and the type of the users located to these nodes. This can be done manually as Figure 23 displays for both Athens and Firenze, while for the city of Firenze, the people count APIs can be utilized for retrieving real-time information about the number of the gathered people at a specific location.

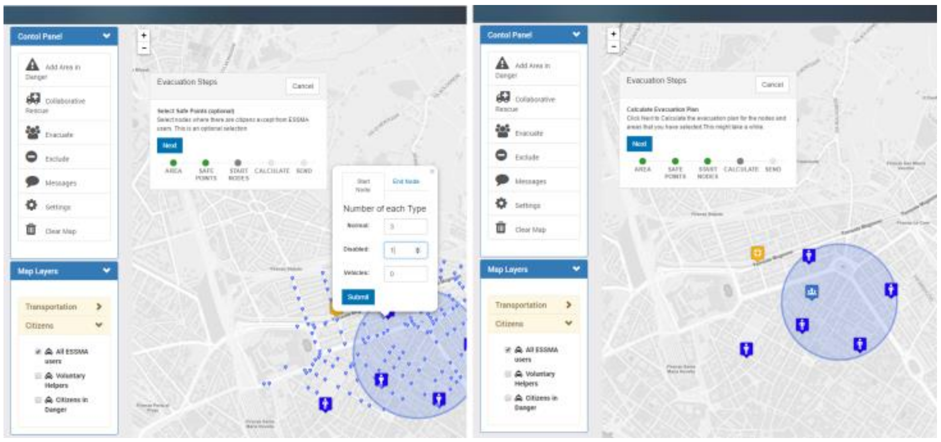


Figure 23: Nodes containing crowd to evacuated defined by the operator

This functionality is useful when the ESSMA is not available to the users (e.g. citizens that do not use smartphones, etc.) and the local bodies undertake to inform the operator about the located crowd and guide the latter. The defined nodes are displayed also with a different marker over the map. After finishing the aforementioned steps, the operator can trigger the eDSS for the calculation of the evacuation plan. The computed plan is illustrated on the map as depicted in Figure 24.

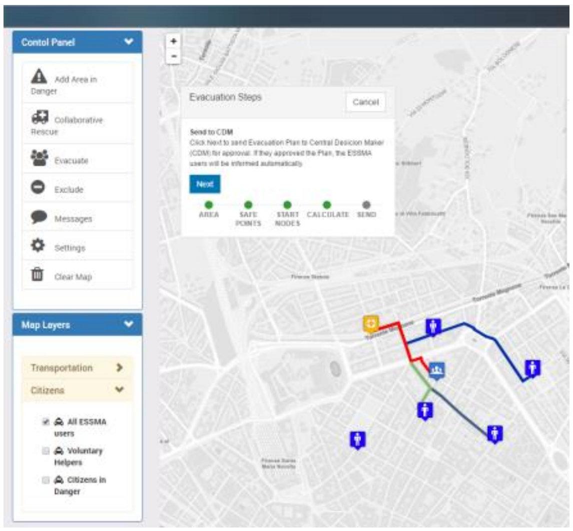
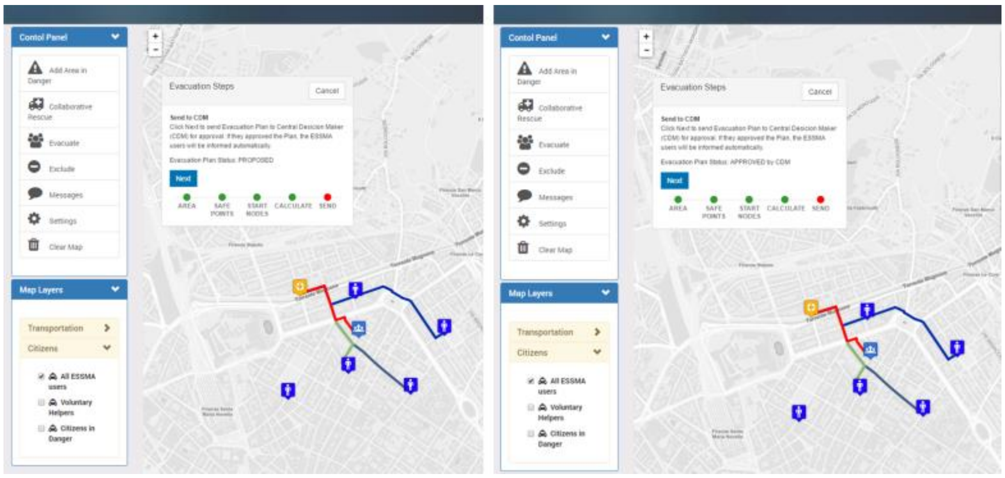


Figure 24: Illustration of the computed evacuation plan over the map.

Figure 24 shows that the eDSS has computed routes for all the ESSMA users that are located within defined evacuation areas as well as for the manually defined nodes. If the evacuation operator approves the computed evacuation plan, he can send it to the Dashboard and the CDB for the final approval. The CDM can either approve or reject the proposed evacuation plan. In each case, the evacuation responsible is notified about the CDM's decision.



Send to CDM
Click Next to send Evacuation Plan to Central Decision Maker (CDM) for approval. If they approved the Plan, the ESSMA users will be informed automatically.
Evacuation Plan Status: PROPOSED

Send to CDM
Click Next to send Evacuation Plan to Central Decision Maker (CDM) for approval. If they approved the Plan, the ESSMA users will be informed automatically.
Evacuation Plan Status: APPROVED by CDM

Figure 25: a) The proposed evacuation plan sent to the CDM (left). b) The CDM approves the evacuation plan (right)

If the CDM approves the proposed evacuation plan, each of the involved ESSMA users receives a personal message with the evacuation route that has to follow.

3.7.3.2 Collaborative Rescue Procedure

The front-end of the eDSS also offers to the operator a mean for watching and managing the collaborative rescue procedure. The idea of the collaborative rescue is based on the assumption that people who are physically capable may be willing to help their fellow citizens that may need help during emergencies. Thus, users that are willing to help others during emergencies can declare it to their ESSMA profile, and be subscribed as voluntary helpers. The following figure demonstrates the sequence diagram of the collaborative rescue procedure.

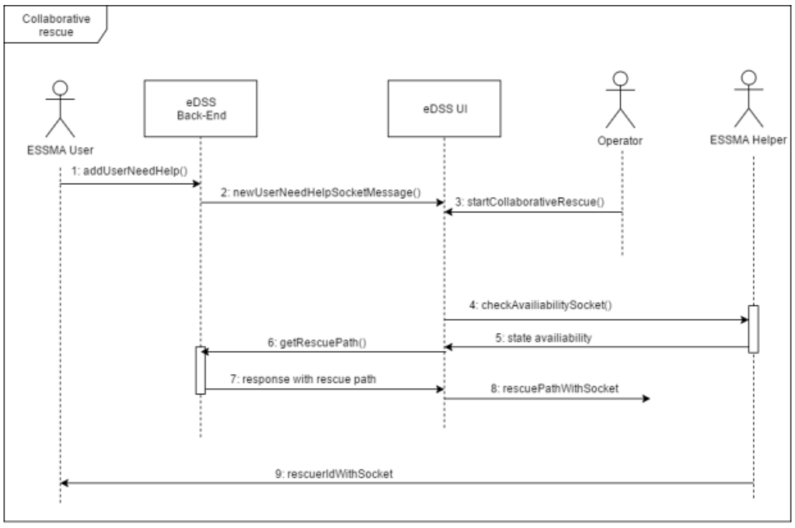


Figure 26. Sequence diagram of the collaborative rescue procedure

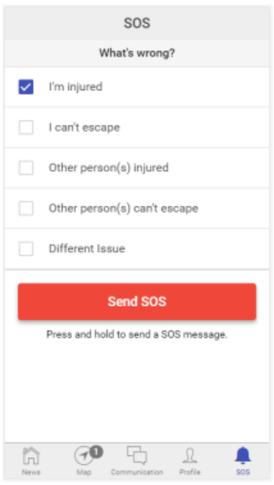
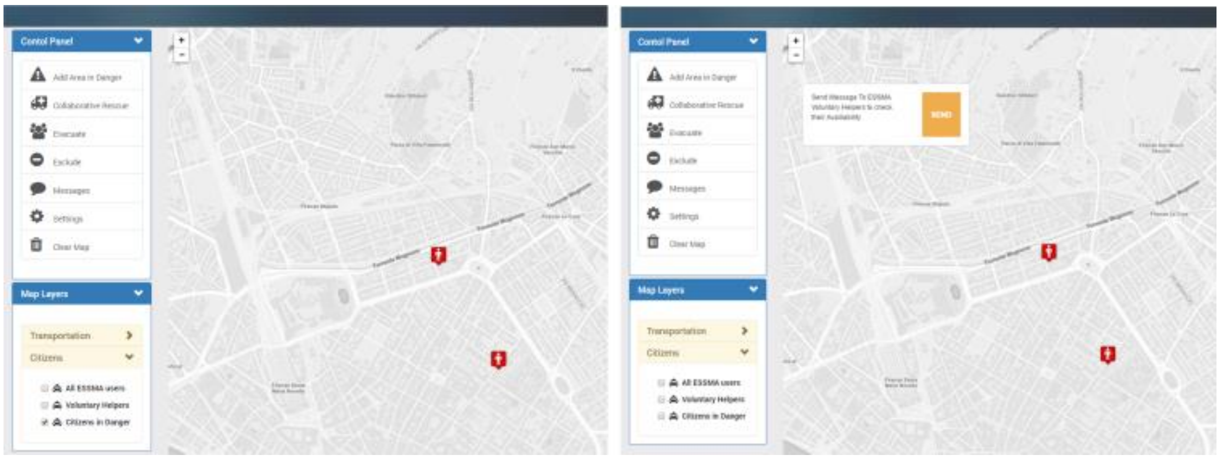


Figure 27: ESSMA SOS button functionality

The collaborative rescue procedure can be triggered only after one or more ESSMA users make an SOS-call. If there are not any users that need help (citizens in danger) and the operator tries to initiate the collaborative rescue procedure, then an alert will be displayed to the operator informing them that they cannot initiate the collaborative rescue procedure.

Figure 27 shows how the ESSMA users can send SOS messages using the app. For sending an SOS message users have to press and hold the “Send SOS” button. Together with the SOS message they can also send more information about their condition to the operator. The ESSMA users that have sent SOS messages are displayed on the map together with the relative information as the left image of Figure 28 displays. In this way the operator is informed at real-time about the users that need help.

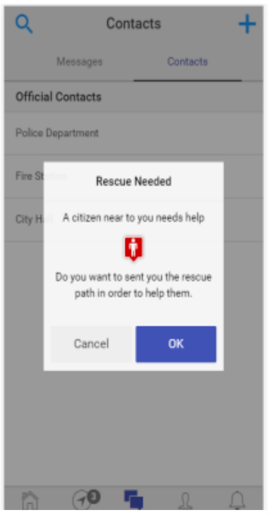
Being informed of users that need help the operator can initiate the collaborative rescue procedure by selecting the "Collaborative Rescue" option from the control panel. At first a button is appeared which enables the operator to send a message to the registered as voluntary helpers ESSMA users asking them if their available to help others.



Send Message To ESSMA Voluntary Helpers to check their Availability

Figure 28: Users that have called the ESSMA's SOS functionality are displayed over the map (left). The operator initiates the collaborative rescue procedure (right).

The sent message is broadcasted to all the voluntary helpers. An alert is displayed to the voluntary ESSMA users as Figure 29 shows. The latter can declare if they are willing to help or not and send their response back to the operator. The users that replied that they are available to help are displayed over the map with green markers as the left of Figure 30 shows. Every time receiving a new reply from the users considering their availability, the map is updated accordingly. When there are at least one or more available helpers, a button that allows the operator to request for the collaborative rescue plan is shown, as it is displayed at the left image of Figure 30. When the operator decides it, he can request for the calculation of the collaborative rescue plan. The resulted plan is displayed over the map (Figure 30 – right image) and the routes that have to be followed by the helpers are sent through the ESSMAs to each one of them. Furthermore, the status of the involved users are updated accordingly so as the available users that have received guidance to help others are marked as occupied users while the citizens in danger that are to receive help are marked as served users. The operator can initiate the whole procedure from scratch when new SOS messages are received. The occupied helpers as well as the served users are not considered as a part of the new plan.



Rescue Needed

A citizen near to you needs help

Do you want to sent you the rescue path in order to help them.

Figure 29. Message sent to the helpers asking them if they are available to help

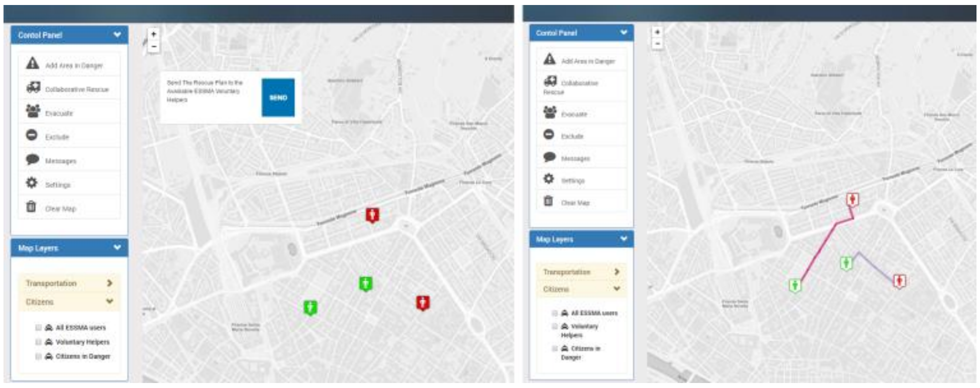


Figure 30: Available users and users that need help are displayed on map, as well as a button that enables the operator to request for collaborative planning (left). The calculated plan displayed on the map and the update of each user's status (right).

A number of choices are given to the involved users after sending guidance to them. Each of the actions, which are described below, can result a sequence of changes:

1. <u>An occupied helper quits from his "mission":</u> An occupied user can quit from his mission by using the ESSMA. The citizen in danger that was meant to be saved by the particular user will be automatically marked as user that needs help.
2. <u>A served citizen in danger informs that he does not need help anymore:</u> A served user declares through the ESSMA that he is safe and he no more needs help. They are removed from both the served users and the citizens in danger. The helper that was guided to this user is informed that his target is safe and there is no more need for helping him.
3. <u>A helper reach at his target:</u> After receiving their targets (citizens in danger), the movements of the occupied helpers are tracked. So when they reach at their targets a new message is displayed to them asking for their next action. They can decide either to get guidance in order to reach a safe point or to declare that they are safe.

3.7.4 Communication with ESSMA users

3.7.4.1 Chat communication

The eDSS in conjunction with the ESSMA offers also another important functionality for the communication of the operator (or the operator's team) with the citizens, namely, the chat communication. As shown in Figure 31, the chat communication is similar to other existing chat mechanisms. This functionality can be triggered by the "Messages" option of the control panel. After clicking on it, a new window is opened where the operator can choose to start a new communication with a specific user or to continue an existing one. Except from plain text messages the operator can receive pictures, which can next download and posting them to the "Live Updates". In this way the user can have an active role in the collection of critical information by early informing them for new hazard events. They can send to the operator or to the other local authorities content that could be vital for the stockholders in order to understand the critical level of a situation. The chat communication can also be used in the evacuation procedure in case the citizens want more details or for informing about an unexpected situation. It can be also utilized by users that are in danger for describing in detail their situation.

3.7.4.2 Live Updates

In the "Live Updates" (Figure 33), which are located at bottom of the right sidebar, the operator is able to add new posts for warning the users about the ongoing critical events. The content of the posts may contain pictures, videos or plain text. Every time the operator creates a new post the ESSMA users are alerted. The "Live Updates" feature offers a kind of communication between users and the operator since users can interact with each post by adding a comment or liking the post. The operator can also reply to a user's comment that will be publically viewed. The whole procedure is empowered with the use of web sockets and push notifications. As we previously mentioned, the ESSMA users are alerted by either web socket or push notification messages.

3.7.5 User setting & Notifications

At the top of the main page there is a bar for facilitating the operator to manage the communication procedure, to sign out from the platform, or to manage their profile setting. By clicking also at the operator's icon at the top bar, the operator can view/update his profile information. The operator can click the operator's icon to sign out from the platform. When a user comments on a post in the "Live Updates" or sends a chat message to the operator the latter is informed by notification icons at the top bar of the main page, as it is displayed in Figure 32. By clicking on the notification a dropdown window appears with the latest "Chat" or "Live Updates" events respectively. By selecting one of the notification messages a new window open with more detail about the event.

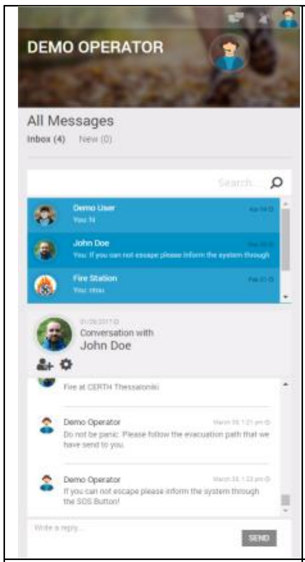
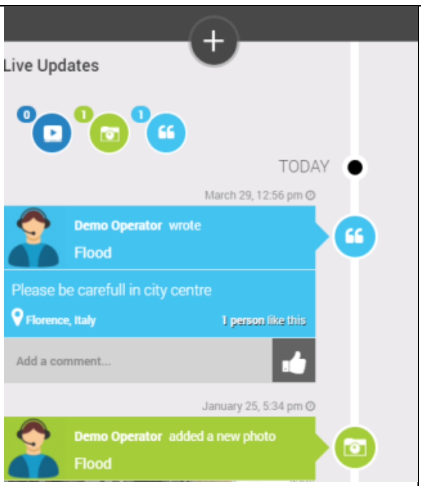
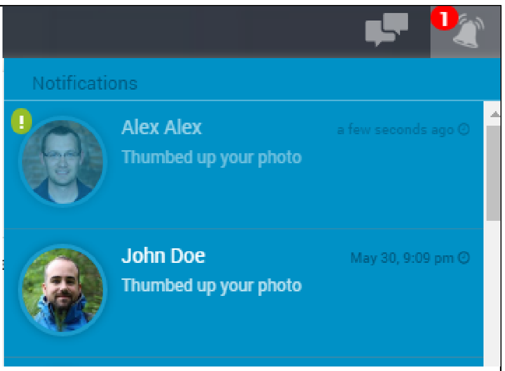
		

Figure 31: Chat communication	**Figure 32: Chat notification**	**Figure 33. Live Updates View**

DEMO OPERATOR
All Messages
Inbox (4) New (0)
Demo User — You: hi
John Doe — You: If you can not escape please inform the system through
Fire Station — You: ntrsu
Conversation with John Doe
Fire at CERTH Thessaloniki
Demo Operator — Do not be panic. Please follow the evacuation path that we have send to you.
Demo Operator — If you can not escape please inform the system through the SOS Button!

Live Updates
TODAY
March 29, 12:56 pm
Demo Operator wrote Flood
Please be carefull in city centre
Florence, Italy — 1 person like this
Add a comment...
January 25, 5:34 pm
Demo Operator added a new photo Flood

Notifications
Alex Alex — a few seconds ago — Thumbed up your photo
John Doe — May 30, 9:09 pm — Thumbed up your photo

4 UTM DSS

The UTM DSS is one of the components of the CRAMSS which, through the implementation of strategic traffic management, enables cooperative operations control by means of definition and automatic identification of control strategies for both daily-life and emergency situations. It covers the following ITS applications (in terms of monitoring, and decision-support system when identifying pre-set network situations):

- Urban traffic control
- VMS control
- Parking management
- Streetlight control

The UTM DSS supports the minimization of damage and increases the resilience of the UTS by adapting the traffic management to current needs.

The main functionalities of the UTM DSS are the following:

- Traffic data (including volumes, flows, occupancy) acquisition and validation from heterogeneous data sources
- Traffic event acquisition from external sources, automatically generated or manually inserted
- Calculation of the O/D Matrix, Level of Service (LoS) on the reference network
- Generation of automatic traffic events based on estimated congestion level
- Identification of the critical points of the road network through off-line simulations for given traffic conditions
- Recognition of specific traffic pattern according to predefined scenarios
- Actuation of strategies through available channels (e.g. VMS, UTC, ESB)

The UTM DSS automatically identifies the critical points on the road network, and uses them to monitor the traffic situation. It supports mobility in an urban area by continuously monitoring the traffic conditions integrating heterogeneous Urban Traffic Systems for a collaborative mobility management.

The UTM DSS has in input:

- the graph of the area of interest
- traffic related data
- traffic events coming from external sources
- O/D matrix

Based on this input and on the off-line identification of critical points of the reference network, the UTM DSS is able to compute traffic predictions across the entire network, therefore being able to provide real-time information in relation to Level of Service, Travel Times, Traffic Events. Furthermore it is able to apply real-time corrective and mitigation actions through all available actuation channels (e.g. create priority corridors through the UTC, display information through VMS) in a coherent way across overall network.

The UTM DSS enhances the awareness at CRAMSS level to the provision of:

- Real-time traffic information
- Critical points of the road network for given traffic conditions
- Real-time traffic events
- Actuated strategies according recognized traffic scenario

4.1 UTM Strategies

The UTM DSS performs a continuous monitoring of selected control points to evaluate the activation conditions for all the defined strategies. When one (or more) strategy activation conditions are verified, all corresponding control actions are activated. Strategy activation can be:

- automatic: control actions are activated directly without prompting the user
- semi–automatic: user is prompted before activating control actions
- manual: strategy actions are started upon user request regardless the activation conditions

Every strategy is defined providing:

- control points
- activation conditions for the control points
- activation conditions for the strategy
- activation type (automatic, semi-automatic, manual)
- actions (operations control for assigned objects)
- priority

UTM DSS offers strategy monitoring services to easily identify:

- running strategies
- strategy condition (activation status, control points status, …)

Running strategies can be terminated:

- automatically, when activation conditions are no longer verified;
- on user request.

Strategic traffic management operations are logged in the RESOLUTE common linking platform logbook where they can be retrieved for offline analysis and statistics.

4.2 UTM Architecture

The following figure shows the UTM DSS components architecture:

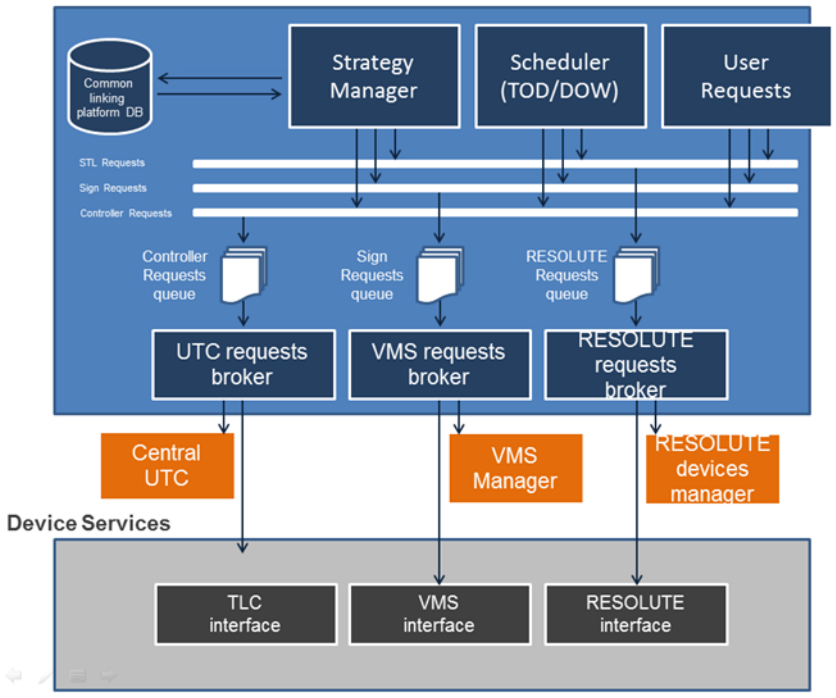


Figure 34 - UTM DSS Component Architecture

The UTM DSS relies on field devices both for data acquisition and control actions; this means that an event that affects field devices or field communication network can compromise the effectiveness and quality of the recognised strategies.

4.3 Component interface

The UTM DSS bi-directionally communicates with the ESB, in order to collect data and update them when needed. Moreover, the UTM DSS sends its output to the ESB. The UTM DSS communicates with UTM actuation channels, as UTC or VMS for the actuation of traffic management strategies. The UTM DSS provides RTTI (Real time Traffic Information) to the RESOLUTE data layer.

4.4 Component API

All objects and methods are exposed using HTTP protocol. Object models can be serialized in XML and Json formats. Object models format can be set by the requesting client by simply specifying expected format directly into the "Accept" request header.

4.5 Retrieve measures for an object

For each object that produce measures (traffic light controller, detection unit, measurement station, detectors) it is possible to retrieve the list of measures calculated by the object. To retrieve measures, send an authorized GET request to the following URL:

```
http://domain/rest/v1/objecttypes/{GUID}/measures?token={token}&fromDate={datefrom}&toDate={dateto}
```

with the appropriate value in place of {*GUID*} and {*dateFrom*} and {*dateTo*} with valid value. Upon success, the server responds with a HTTP 200 OK status code and measures list. Examples of Requests & Responses can be found in Annex I

4.6 ESB interface

The UTM DSS exposes two categories of data: events and strategies. Sample XML can be found in Annex III.

5 UPT DSS

The UPT DSS is the component of the CRAMSS able to combine data flowing from Signalling Controllers and On Board Units in an integrated manner such as to provide a fully featured system to monitor and control the tramway operations. Localization and monitoring of the vehicles and supervisory of the tramway Signalling system is achieved through a “distributed” architecture.

At the OCC, the UPT DSS application is in charge of collecting, managing and dispatching data coming from the Wayside Controllers and On Board Units. Each of the typical Tramway function can be treated, allowing the operators to generate timetables, track and retrieve events, visualize and follow on the display the trams’ positions and their current status, manage train regulation, remotely control wayside devices, receive and acknowledge warnings and alarms.

The UPT DSS is a system that manages the tram localization. It is able to detect the tram position on the line and to manage the tram regulation by early/delay messages. It manages the ground equipment status, sets main line and depot route requests for tram parking. The UPT DSS displays the status of equipment like signals, switches, rail track circuits, trams, wayside controller and road signals and it provides the interface to manage the Timetable. The system provides all functionalities to manage the entire tramway in a single integrated HMI.

The UPT DSS manages also the Wi-Fi Network of the tramway to provide free internet access to all tramway passengers, both on board of trains and on platforms. Through this system (UPT DSS Wi-Fi Connector) it is possible for the Metropolitan City of Florence to display messages and information directly on passengers’ smartphones (on the internet connection splash page). Moreover, the system is able to track the location of each connected device, thus allowing a precise map of number and location of tramway’s passenger (therefore an estimation of the number and location of all tramway passengers)

For the RESOLUTE Project, to avoid disruption of operations on the real-tramway DSS, a simulator has been developed by Thales to simulate tramway operations sending and receiving relative events to the Resolute ESB.

5.1 Architecture

The following figure shows the UPT DSS components architecture:

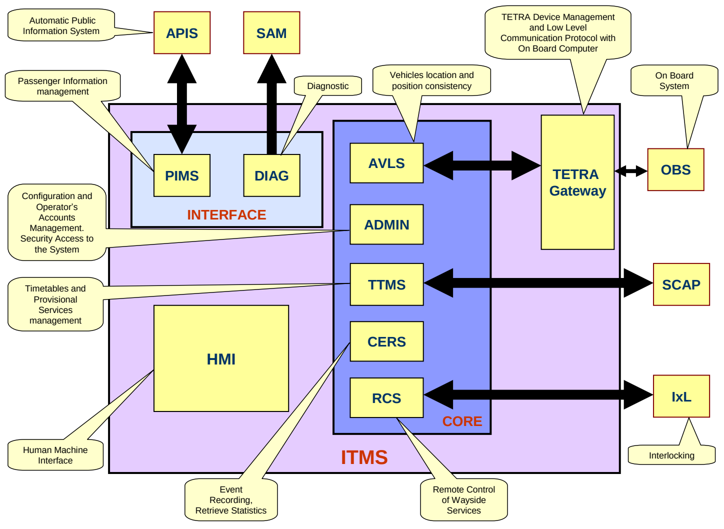

Automatic Public Information System
APIS
SAM
TETRA Device Management and Low Level Communication Protocol with On Board Computer
Passenger Information management
Diagnostic
Vehicles location and position consistency
On Board System
PIMS
DIAG
INTERFACE
AVLS
ADMIN
TETRA Gateway
OBS
Configuration and Operator's Accounts Management. Security Access to the System
TTMS
SCAP
Timetables and Provisional Services management
HMI
CERS
RCS
IxL
CORE
ITMS
Human Machine Interface
Interlocking
Event Recording, Retrieve Statistics
Remote Control of Wayside Services

Figure 35: UPT Architecture

The following figure shows the UPT DSS Wi-Fi Resolute connector:

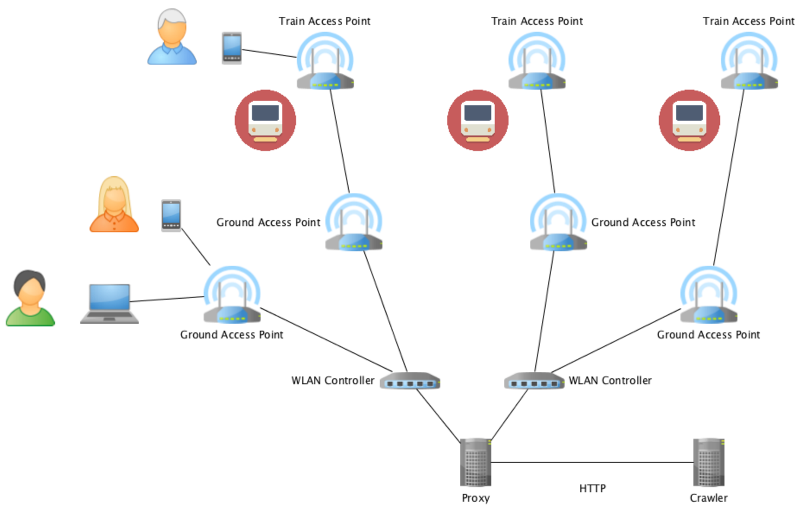


Figure 36: UPT DSS Wi-Fi Resolute connector

5.1.1 Component interface

The UPT DSS communicates with the ESB, in order to send events to other DSSs and to the CRAMSS user interface. The UPT DSS provides real-time number and location of people connected to the tramway Wi-Fi to the RESOLUTE data layer.

5.1.2 Component API

Tramway events from the UPT DSS simulator are sent to the ESB using the Resolute ESB Data format. Real-time number and location of people connected to the tramway Wi-Fi are exposed using HTTP protocol. Object models are serialized in Json formats.

5.1.2.1 UPT DSS Simulator Events Interface

The UPT DSS is able to inject events into Resolute ESB through the following interface…

```
http://domain/rest/v1/objecttypes/{GUID}/measures?token={token}&fromDate={datefrom}&toDate={da
teto}
```

Event List:

Below some examples of the events that could be generated by UPT DSS Simulator:

```
Technical issue

Line suspended

Line separated in two rings
```

Below some events that could be received by the UPT DSS Simulator:

```
Close Tramway Order

Suppress Station Order

Other Big Impact Tramway Decisions
```

The list is not complete, since the type of events that could be generated is completely configurable at runtime, therefore will be updated during the evolution of the project.

5.1.2.2 UPT DSS WiFi Connector Interface

The UPT DSS WiFi connector continuously calculates the number and locations of devices (thus passengers) connected to the tramway WiFi, and exposes the results using an HTTP interface reachable at the following endpoint (the destination port is configurable)

```
http://<WiFiConnectorIP>:8888
```

The data exposed from the WiFi Connector shall conform to the JSON SCHEMA of Annex II.

To avoid any privacy issues, the ID of the devices monitored by the UPT DSS WiFi connector are anonymized before being sent, through a service provided by the Metropolitan City of Florence.

6 RESILIENCE DASHBOARD

The Resilience Dashboard is created through the Dashboard Builder, which is an open source tool developed by DISIT lab in the context of H2020 REPLICATE project, a smart community lighthouse[10]. It represents the synthesis of the status of the UTS and it is thought for being installed in each control room of the interested stakeholders.

The information provided has been selected according to the D4.2, D5.1 and D5.2 outcomes where an extended analysis with the operators has been carried out to identify what are the critical decisions that cause delay and which kind of data they want to have to enhance such decisions. Even if the requirements have been identified with precision, the flexibility of the application is a "must to have" feature. In fact, in view of un-anticipated changing conditions and to support learning and improvement cycle, the Dashboard might need to be periodically updated to respond to every emergent need. To this end, the choice of using the Dashboard Builder application has been considered the most appropriate. The Dashboard Builder web app presents a back office where can be managed the entire dashboard created. To access to the tool it is necessary to obtain a Username and Password.

6.1 RESOLUTE DASHBOARD UI

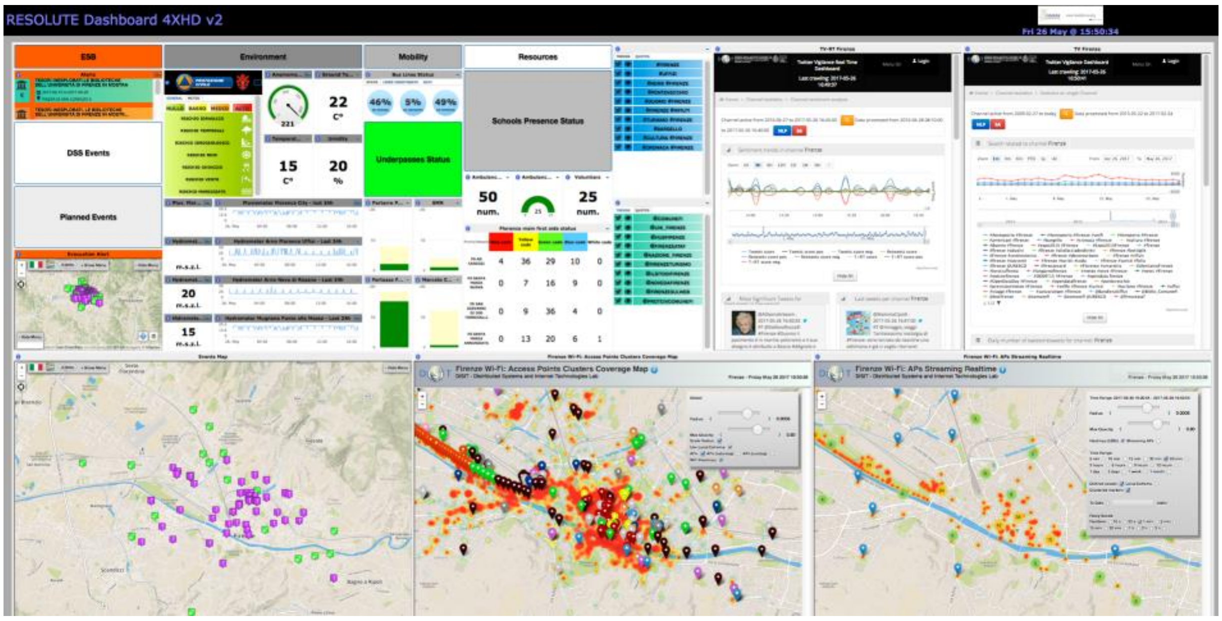


Figure 37. Dashboard UI

The Dashboard has been implemented following the D5.2 recommendations. The Dashboard is going to be used by different users such as Civil protection, Mobility dept., Urban Police, etc. thus the information included in the panel comes from the result of a harmonization of the requirements applied in the D5.2.

The widgets have been grouped vertically, according to the domain they refer to:

- messages coming from the different DSS and published in the ESB (ESB column)
- indicators referred to the most important environmental aspects (Environment column)
- indicators referred to mobility aspects (Mobility column)
- indicators referred to resources availability in the system (Resources column)
- Social media section

[10] www.replicate-project.eu

- Service map
- Territory usage clustering
- Real time people concentration

The layout strategy is organised from the Real Time (left) info to the slow dynamics info (right)

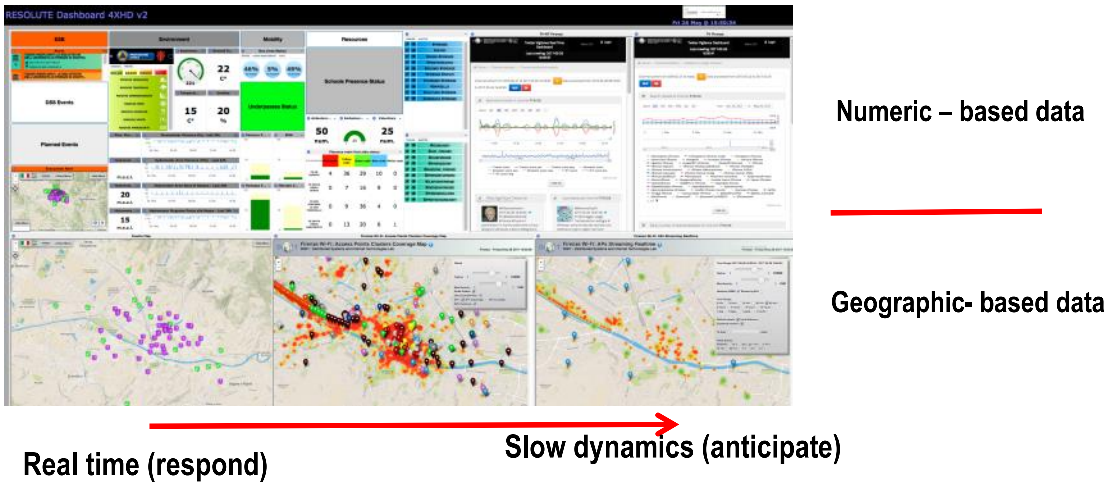


Figure 38. Layout strategy of Dashboard

The resulting Dashboard, thus, can be used in different situations while reducing confusion and mental workload thanks to the spatial organization of the signals. In particular numeric parameters has been separated by the geographical data. Weithin the numeric section the information has been organised in columns as described in the subsequent sectons.

6.1.1 ESB column

In this column the info read form the ESB are shown. Such info are basically: Alerts (e.g. car accident), events generated by the connected DSS (e.g., Mobility supervisor), the events already planned in the city and that are on-going, and the messages arrived from the eDSS where an acceptance request is sent to the ESB and displayed by the Dashboard. In particular, through the account of the Mayor (see. D5.2) it is possible to accept the evacuation route proposed by the eDSS and send it back to it through the ESB the message.

6.1.2 Environmental column

In this column a number of relevant indicators has been grouped, which are useful to assess the status of the environment in the city. The extreme events related to the climate change, such as storm, flash flooding, heat wave, etc. can be easily monitored and put in relation to each other when they are displayed together. In fact, such information is generated by different authorities, and a unique view able to combine them together enhances the situational awareness. The indicators selected are: river levels, wind speed, humidity, civil protection alerts, and temperature.

6.1.3 Mobility column

In the mobility column the information regarding the status of relevant mobility services and infrastructure are reported. In particular, the status of the public transport system (bus), the status of the underpasses (open/closed), and the status of the parking's availability. The traffic status is directly accessible through the Service map widget, while the mobility events are include di the ESB column (alert or DSS).

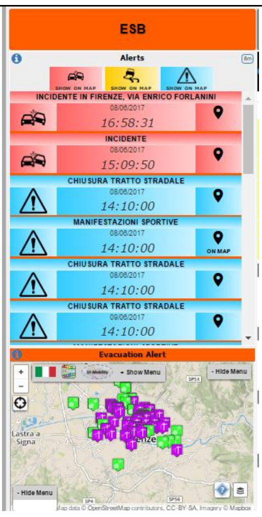


Figure 39. ESB column

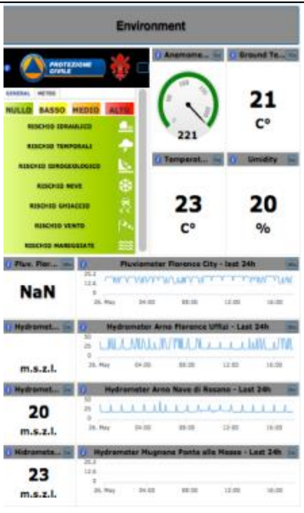


Figure 40. Environmental column

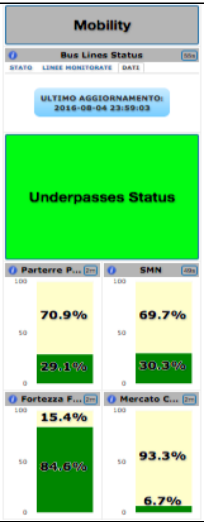


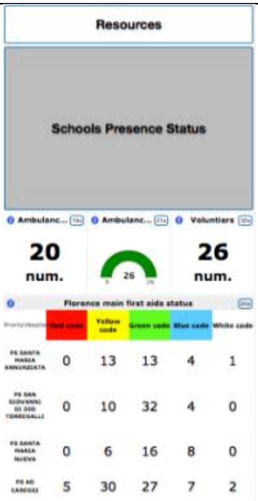
Figure 41. Mobility column

6.1.4 Resources column

This section includes the resources that can be monitored in the city. The availability of resources and their efficient allocation are at the core of the resilience strategy and adaptive capacity estimation. The resources considered useful for taking decisions are: children’s presence at school day by day, number of ambulance units available, number of civil protection volunteers available, the triage status for the hospitals in the city in real time.

6.1.5 Twitter Vigilance

Additionally to the work presented in “Section 5.6.2 - Twitter Dataset creation of the deliverable” D4.2, this section includes the information coming from the Twitter Vigilance tool. It is a separated application but it has been widgetised for the RESOLUTE project in order to have a view of the social media trends in real time integrated in the Dashboard. There are three kinds of information displayed: most frequent tags and users cited real time trends, and daily-based statistical analysis.



[illegible]	Red code	Yellow code	Green code	Blue code	White code
PS SANTA MARIA ANNUNZIATA	0	13	13	4	1
PS SAN GIOVANNI DI DIO TORREGALLI	0	10	32	4	0
PS SANTA MARIA NUOVA	0	6	16	8	0
PS AO CAREGGI	5	30	27	7	2

Figure 42. Resources column

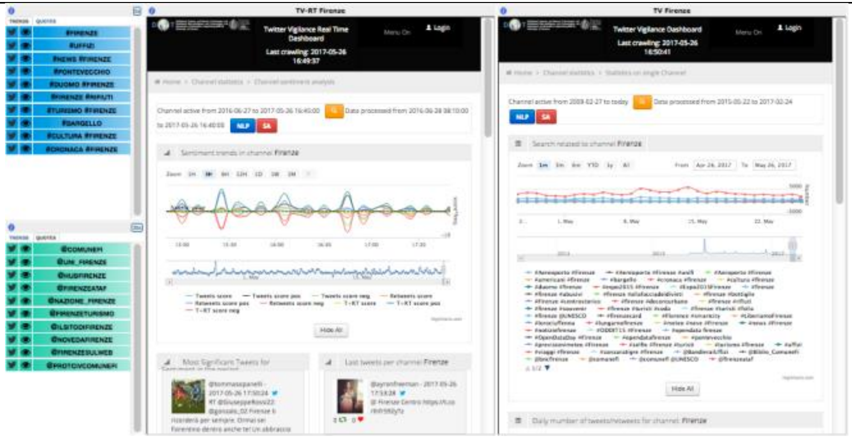

TV-RT Firenze
Twitter Vigilance Real Time Dashboard
Last crawling: 2017-05-26 16:49:37
TV Firenze
Twitter Vigilance Dashboard
Last crawling: 2017-05-26 16:50:41
Login
#FIRENZE
@COMUNEFI
Sentiment trends in channel Firenze
Most Significant Tweets for
Last tweets per channel Firenze
Search related to channel Firenze
Hide All
Daily number of tweets/retweets for channel Firenze

Figure 43. Twitter Vigilance

The Twitter Vigilance Real Time platform (http://disit.org/rttv/) has been designed by the DISIT Lab of the University of Florence as a multipurpose comprehensive tool, providing different tasks and metrics suitable for Twitter data analysis, with the aim of early monitoring critical events and different contexts regarding resilience aspects, producing also alerts, notifications and several kinds of actions. Its modular architecture is based on a distributed crawler, which performs data gathering and extraction by using Twitter public APIs. The data acquisition approach is based on the concept of Twitter Vigilance Channel, consisting of a set of simple and complex search queries, which can be defined by a registered user by combining keywords, hashtags, user IDs, citations, etc., in a structured logical syntax, according to the search syntax of Twitter. Subsequently, collected tweets are processed by the back-office processes, which implement statistical analysis (number of tweets, retweets, number of unique users etc.), natural language processing (NLP, extracting different Part-of-speech items such as nouns, adjectives, verbs, hashtags and mentions) and sentiment analysis (in terms of a positive and negative polarity sentiment score of extracted text items), as well as general data indexing. The metrics resulting from by the back-office processes (low-level metrics) are computed in real time and stored in a dedicated database. This approach allows making them accessible to the front-end graphic user interface, which graphically show their temporal trend and allows users to download processed data for further analysis.

Users can also define custom high-level metrics (by exploiting the above mentioned low-level metrics, dynamically combining them by using mathematical operators and expressions, as well as other user defined high-level metrics). It is also possible to define temporal trends of high-level and low-level metrics, computed by summing the metrics on defined time intervals chosen by the user, allowing also to specify temporal repetition of the computation. In a similar way as for low-level metrics, the computation of high-level metrics and trends is carried out by scheduled processes, which are periodically computed by the back office.

Furthermore, besides the definition of high-level metrics and trends, users can define custom thresholds (which can be represented by constants as well as by other metrics) for defined metrics. Dedicated back end processes continuously monitor low-level and high-level metrics, trends and corresponding thresholds in order to detect when a threshold is exceeded:. When this occurs, a firing event is produced by the system through a specifically designed notificator, which can perform several kinds of operations, such as: automatically sending alerts through email (to custom lists of recipient users), SMS, and also activating the emergency call center by posting tweets (Twitter bot), and also performing actions, for instance sending API rest calls to other tools dedicated to produce recommendations and/or decision support systems.

6.1.6 Service map

This widget includes all the information provided by the Service Map web application. For instance, it is possible to display the real time data of traffic sensors, the bus stops status, healthcare services, accommodation services, etc. It is possible to obtain info of ca. 20K different services in the city, organised in 20 categories and 512 sub-categories.

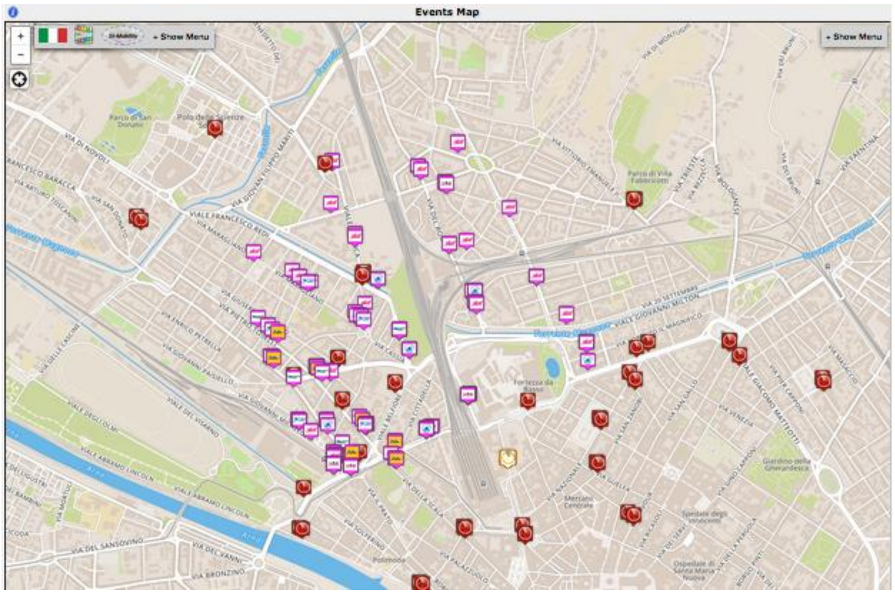
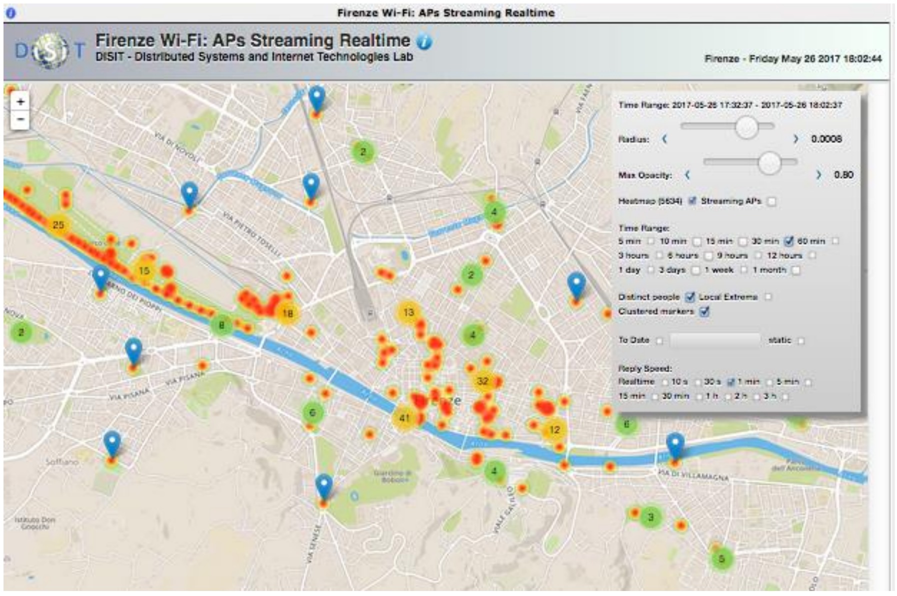


Figure 44 - Service map

6.1.7 Real time people concentration

This widget represents the presence of the people in real time in a specific area of the city. The data displayed is the result of continuous a geo-clustering of the data coming from the city's open WiFi. This widget includes all the features of the APs Streaming Realtime web application and allows several levels of filtering according to the user needs.


Firenze Wi-Fi: APs Streaming Realtime
Firenze Wi-Fi: APs Streaming Realtime
DISIT - Distributed Systems and Internet Technologies Lab

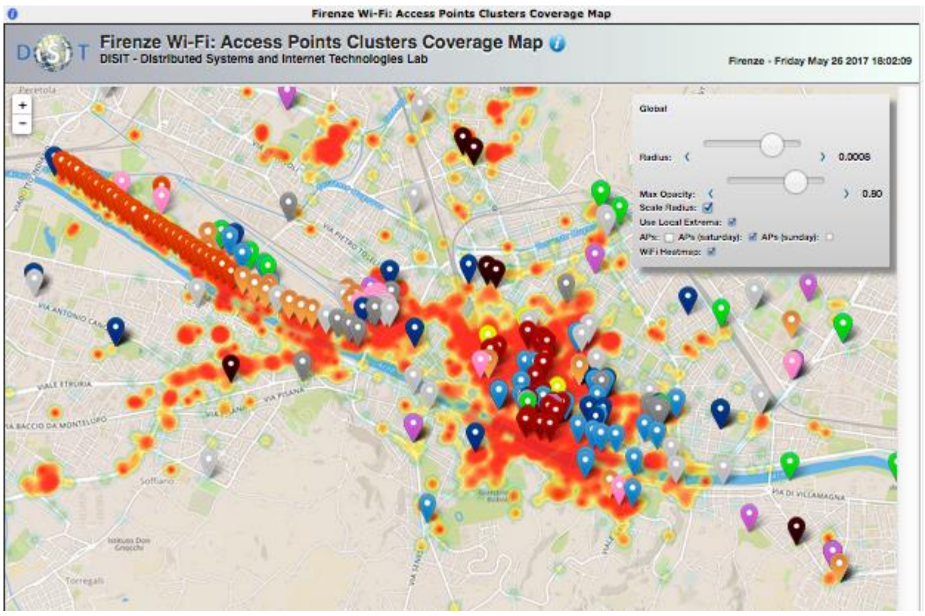
Firenze - Friday May 26 2017 18:02:44

Time Range: 2017-05-26 17:32:37 - 2017-05-26 18:02:37
Radius: 0.0008
Max Opacity: 0.80
Heatmap (5634) ✓ Streaming APs ☐
Time Range:
5 min ☐ 10 min ☐ 15 min ☐ 30 min ✓ 60 min ☐
3 hours ☐ 6 hours ☐ 9 hours ☐ 12 hours ☐
1 day ☐ 3 days ☐ 1 week ☐ 1 month ☐
Distinct people ✓ Local Extrema ☐
Clustered markers ✓
To Date ☐ static ☐
Reply Speed:
Realtime ☐ 10 s ☐ 30 s ✓ 1 min ☐ 5 min ☐
15 min ☐ 30 min ☐ 1 h ☐ 2 h ☐ 3 h ☐

Figure 45 - Rea time people presence

6.1.8 Territory usage

This widget displays on the map the access points with a colour that presents a specific kind of access dynamics. Access points with the same colours exhibit the same access dynamics. Since each access represents a presence of the person in a specific area of the territory, we can deduce that such a measure represents how the territory is used by the people along the day. With this perspective, it is possible to identify which areas of the city have a pick of presence in the same time window. Such information can support the operators in optimising the patrols distribution.



Firenze Wi-Fi: Access Points Clusters Coverage Map

Firenze Wi-Fi: Access Points Clusters Coverage Map
DISIT - Distributed Systems and Internet Technologies Lab

Firenze - Friday May 26 2017 18:02:09

Global
Radius: 0.0006
Max Opacity: 0.80
Scale Radius: [✓]
Use Local Extrema: [✓]
APs: [ ] APs (saturday): [✓] APs (sunday): [ ]
WiFi Heatmap: [✓]

Figure 46 - Territory usage

7 RESILIENCE DS TOOL

The Resilience Decision Support (**Resilience DS)** tool is a collaborative tool extending the FRAM model for several aspects and integrating it with a decision support grounded on data and experts' assessments. Such a tool allows modelling a sociotechnical system and generating formal models for continuously assessing the CI's resilience and conditions. In this section, we explain the Resilience DS's architecture. We report how the single parts of the software were made and how they interact with each other. The tool is presented in the following section, detailing its interface and the features.

7.1 Tool Architecture

The Resilience DS is a client-server web application with the aim of supporting resilience analysts in modelling complex socio-technical systems, such as a UTS. The first step was to create a hierarchic schema of models and the models' instances. The models are equal to FRAM models and are composed of functions and aspects. An instance, instead, is linked to the model and contains its specifications, such as the functions' variability, and who carries out the functions. Consistent with FRAM, the function's variability can regard the precision and the time. The function's handler can be a human, an organization or a technology. The instance contains the necessary information to execute the model on SmartDS. A future development is to transfer the required data directly to the ResilienceDS, so that it can also be used in other simulation tools. However, we have created this tool in a way that can be possible to extend the model information without deleting the work done or the saved models and it can be done without much effort.

With respect to the modelling, a user can:

- Create, load, import, modify and delete a model.
- Create functions and for each function provide: name, description and colour.
- Create aspects, between functions or for only one function. For each aspect, it is possible to associate: a label, a source, a target and a type that can be one of the 6 vertices of the FRAM's hexagon: Input (**I**), Precondition (**P**), Resource (**R**), Control (**C**), Time (**T**) or Output (**O**).
- Create groups of functions.

For the instances, the tool gives the possibility to provide for each functions:

- The variability concerning the time (Too early, in time, too slow, not at all) and precision (Precise, Acceptable, Imprecise).
- The Function's type, precisely who carries out the operations (Human (**H**), Organization (**O**) or Technological (**T**)).

For the user management, the provided operations are:

- Differentiation among users' typology.
- Login/logout of a registered user.
- Registration of a user from the client side.
- Advanced operation for an administrator
    - Users' permissions management
    - Registration in the system of new users.

The system handles the interface with a relational Database and with an RDF store. The former is needed to store the models, their functions and their aspects. The latter is used to retrieve the data necessary for linking the functions to the respective data from the Smart City. For the RDF store, methods have been created but they have not been implemented, yet.

Depending on the user's permissions, the user interface allows the user to perform the actions described above. So, each user typology has a list of operations that can be done, while the server side manages the operations on a model and provides the interface between client and MySQL. The server retrieves all the information stored in the DB, about the models and the users.

The final result is composed of a client-server application. The client side lets the users to do the operations on a model, and define it and its instances. The server is composed by two modules. The first one, the FRAM module, is related to the model and instance management. For that, there are two interfaces. One is for the creation, saving, loading, importation, exportation, modification and deletion of a model. The other is for saving, loading, modifying and deleting instances. In the FRAM module, there is yet another interface for linking the model with the SQL database and the RDF store. The interface to the SQL DB is necessary for saving the model, the instances and their data. The interface to the RDF store is necessary to the retrieve the data from the Smart City, the sensor or other simulation tools.

The second module serves for user handling. Through the DB it is possible to save and retrieve users' information and make different functionalities accessible to them, based on the typology.

Each module is divided into three parts: one interface for connecting client and server, a core with the classes for managing the models and another interface for translating the information requested and saving it in the DB. More details are explained in the next sections. Figure 47 shows the system schema and the interactions among the different parts.

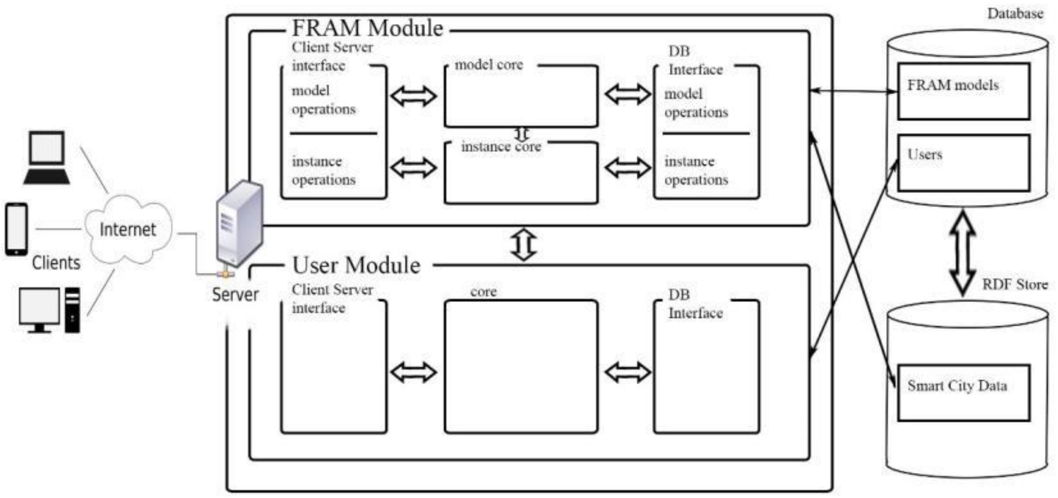


Figure 47. Resilience DS Architecture

Regarding the program languages, we have used java for the server side, whereas the client was developed with JavaScript. For the latter, the D3 library and the jQuery framework have been used.

7.2 Database Structure

This section describes the approach, which in the Resilience DS serves to generate and store a FRAM's model. To start, we have considered the XML format produced by the FRAM Model Visualizer (FMV), presented in [27]. The FMV application is used to create and visualize FRAM models. It stores all of the data in a text file with a '.xfmv' file extension, using a standard XML format. The data files can be viewed using a text editor or imported by any program that reads XML data. The XML structure has 4 hierarchical levels: The 'root' level sits within an <FM> tag.

1. The second level contains the following seven sections: **<Functions>**, **<Controls>**, **<Inputs>**, **<Outputs>**, **<Preconditions>**, **<Resources>**, **<Times>**. An **<Aspects>** section may be present but is not used by the FMV, as this level was created by an earlier FMV application.
2. Each of the seven sections above can contain any number of tags that represent the elements of that section. For example, Functions in the FMV model are nested within the <Functions> tags, each having its own **<Function>** tag at the third level. The remaining six sections are for the FMV model's Aspects. Each <Input> tag is nested within the <Inputs> tag, and so on.
3. The fourth level contains data specific to the elements within each section.

For each Function, there are 4 data fields:

<IDNr> An index number for the Function. Sequential, beginning with zero.

<FunctionType> The type of Function as determined by the application. Valid values are: 0 (Foreground), 1 (Foreground Variable), 2 (Background).

<IDName> The name of the Function as entered.

<Description> The description of the Function as entered.

The <Function> tag has 8 attributes used to store data specific to the visualisation of the FRAM model:

fnStyle: Indicates the model rendering style: 0 (Traditional), 1 (Modern).

Style: The selected pre-set colour of the Function: white, blue, green, grey, red, yellow, purple, custom.

Color: If the style is 'custom' then this is used to store the custom colour value.

fnType: The variability type. Valid values are: 0 (undefined), 1 (Technological), 2 (Human), 3 (Organisational).

x, y: These are the x, y coordinates for positioning the Function within the model space.

Tp: The potential output variability with respect to time: 0 (Too early), 1 (On time), 2 (Too late), 3 (Not at all).

Pp: The potential output variability with respect to precision: 0 (Precise), 1 (Acceptable), 2 (Imprecise).

For each of the other sections that contain Aspect information there are three data fields:

<IDNr> An index number. Sequential within each section, starting at 1.

<IDName> The name as entered.

<FunctionIDNr> The index number (IDNr) of the Function that this Aspect is a child of, for example, an Output with <FunctionIDNr> = 0 will be an output of the Function that has <IDNr> = 0.

The association is based on the attribute IDName, so if another Function has a different Aspect with the same <IDName>, the two will be shown as linked in the FMV.

In Figure 48 example created with FMV is depicted, which we use for showing the internal structure. The corresponding FMV file is provided in Annex IV.

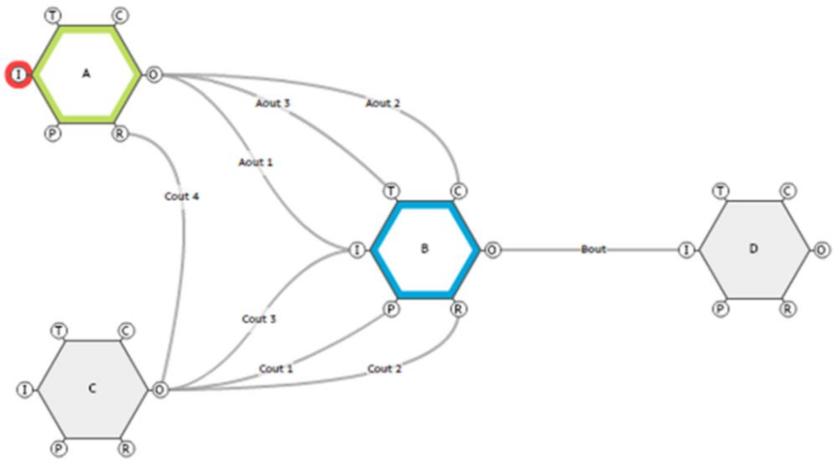


Figure 48. FMV Example with 4 functions: 2 foreground (A, B) and 2 Background (C, D).

What we expect from the Resilience DS is at least the same features of the FMV, especially for editing a model. Furthermore, we want to distinguish the users and give them the possibility to share the model between each other. So, starting from the structure above, to save a FRAM model, we have identified 5 tables, corresponding to the main classes, that are:

- **Model**: Represent an instantiation of the FRAM model. The fields are:
    - **id:(int)** the univocal model identifier. An integer value starting from one.
    - **type:(varchar)** the type of visibility/share of the model. Possible values are:
        - **public:** All the users can see, share and use as template the model.
        - **private:** The model is shared in a group and all the users who belong the group can view, share and modify it.
        - **protected:** The model is shared in a group and all the users who belong the group can view the model but not share or modify. Only the creator can make changes or share it with other.
    - **objective:(text)** the model's goal.
    - **description:(text)** a text variable that reports a description of the model's scope.
    - **idUser:(int)** the identifier of the user that has created the model.
    - **data_create:(time)** the data creation of the model.
    - **data_last_change:(time)** the data of the last change in the model.
    - **size:(int)** the number of functions present in the model.
- **Function:** are the FRAM's functions\actions which compose the model (The hexagons). The fields of this table are:
    - **id:(int)** the univocal function's identifier. A progressive value that start from 1.
    - **idModel:(int)** the univocal identifier of the model that contains the action.
    - **Name:(text)** the function's name.
    - **Description:(text)** the description of the action.
    - **Position:(integer, integer)** the x-coordinate and y-coordinate of the hexagon in the frame.

   - **Color:(text)** the color of the function's hexagon. This value can be a predefined style, for example: red, blue, green etc. or a rgb value. The default value is black (#000000)
   - **function_type:(int)** like FMV this value reports the role of the function in the current model. This value is set when a model is imported from FMV, and actually is not updated. The values are:
     - **0: foreground**. A function that is part of the study focus, which in practice means if the variability of the function may have consequences for the outcome of the event or process being examined.
     - **1: background**, something that is used from foreground function. It also used for drain function or downstream, the function that receives an Input, and it represents a further process or continuation of the event. So this type of function can provide the inputs for other blocks, or receives the output of others without doing anything.

Unlike FMV, which uses a third value for describe function in foreground that has a variation, we use only 2 options for 'function_type'. The way that we represent a variable function is explained above, with the introduction of the class 'model_instance'.

- **Aspect:** in the literature, the term 'aspect' represents one of the five inputs of a FRAM block. For us, this class represents the five entries: input, prerequisite, resource, time and control. Furthermore, with this term, we also considered the remaining pin of the hexagon, the 'output'. This is because, in FRAM, an output is also an input of another function (except for a drain function). This simplifies the DB management and avoids duplications. Below, the variables of an aspect are reported:
   - **id:(int)** the identification value for the aspect.
   - **idModel:(int)** the model where the aspect is defined.
   - **label:(text)** the label to show on the edge.
- **FRAMGroup:** these are the components introduced in the Resilience DS in order to handle hierarchy and models with a huge number of functions. A group is substantially a function of functions whose behaviour depends on its components. For that reason, the attributes are essentially the same as those of a function.
- **Model_Instance:** is the class that represents an instantiation of the FRAM model. In each instantiation, it is possible to study the behaviour of a model due to functional variability.
   - **id:(int)** the univocal identifier of a model instantiation.
   - **idModel:(int)** the univocal identifier of a model.
   - **idUser:(int)** the univocal identifier of the user who creates the instantiation. This user is not necessarily the model's creator.
   - **description:(text)** the instance's description.
   - **data_create:(time)** the data of creation of the model.
   - **data_last_change:(time)** the data of last change in the model.

Now, we are going to show how these classes are connected. In Figure 49 is reported the connection between Model and Function. Each model can have different functions, as opposed every function appertains at one model.

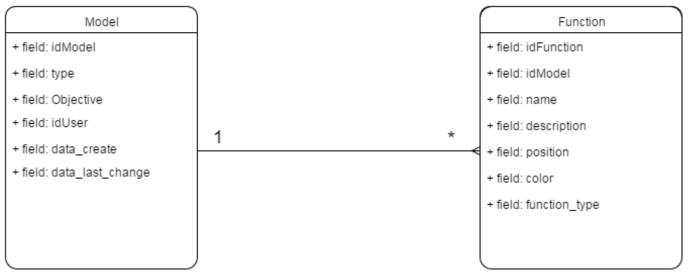


Figure 49. Relation between the classes Model and Function.

In Figure 50 the connection between the tables is reported: Function and Aspect. Every FRAM function can have more aspects and outputs, not only one for entry. On the other hand, each aspect can be shared between more functions; in a FRAM representation it is possible to have 2 or more hexagons with the same input. Moreover, each output is an input of another function (except the case of drain). So, we have introduced a table, **Functions_Aspects**, for those connections. The functions' attributes are:

- **idFunction:(int)** the identifier of the function. That together with idAspect forms the key of the table.
- **idAspect:(int)** the identifier of the aspect. That together with idFunction forms the key of the table.
- **type:(text)** the aspect type in the referenced function. The value can be one of the 6 vertexes of the hexagon: **Input, Precondition, Resource, Time, Control** or **Output**.

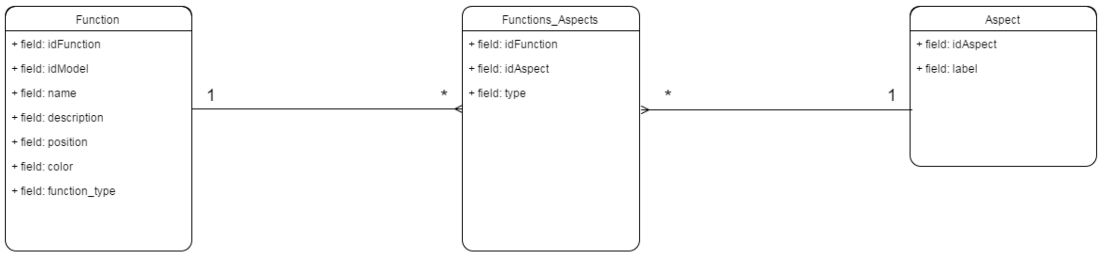


Figure 50. Relation between Function and Aspect.

The last thing that we have to consider is the instance of a model. A model can have different instances, opposed to the instance that can appertain only to a specific model. As said before in each instance we can set a variation value for the function, so every instance can have different settings for the model's functions. Consequently, a function can have different variations depending on the instantiation studied. For that reason, we have created another table, **Models_Functions**. In this table, we have:

- **idInstance:(integer)** the identifier of the instance. That together with idFunction forms the key of the table.
- **idFunction:(integer)** the identifier of the function. That together with idInstance forms the key of the table.
- **time_precision:(integer)** like FMV we report the expected variation in time's precision. The possible values are:
  - **-1:** No variation expected.
  - **0:** Function finishes too early.
  - **1:** Function finishes on time.
  - **2:** Function finishes too late
  - **3**: Function not at all finished. It could be not terminated, maybe for an interruption.
- **potential_precision:(integer)** as in FMV this value is the variation in the output. It can be:
  - **-1:** No variation expected
  - **0:** The function's output is precise, exactly what we expect from that.
  - **1:** The function's output is acceptable, not so far from the expectation.

- **2:** The function's output is imprecise, not what we expect.

- **function_maker:(integer)** who or which do the function. Like FMV, the set of values are:
    - **0:** undefined.
    - **1:** function performed by a technological part.
    - **2:** is a Human who carries out the function.
    - **3:** an Organisational group conducts the function.

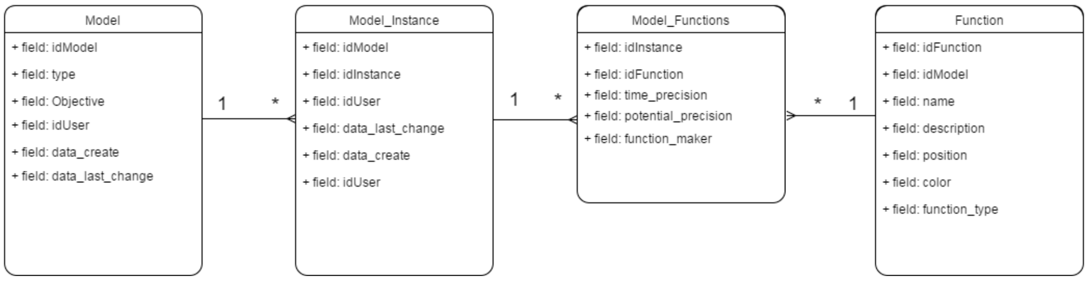


Figure 51. Model instances data management.

At last, we see that a group contains functions, and it can also contain other groups or be included in one group. A single function can be included in only one group. For that, we have introduced two tables: **FRAMGroup_Function** and **FRAMGroup_Hierarchy**. The first is used to link the groups with their functions. This table is formed of two columns, one for the group's id and one for the function's id. The second table, analogously to the first, serves to link the group with the groups in it. This table contains the identifier of the group father and the identifier of the group in it.

In the last figure we have reported the schema of the complete DB, with all the tables and the connections explained before.

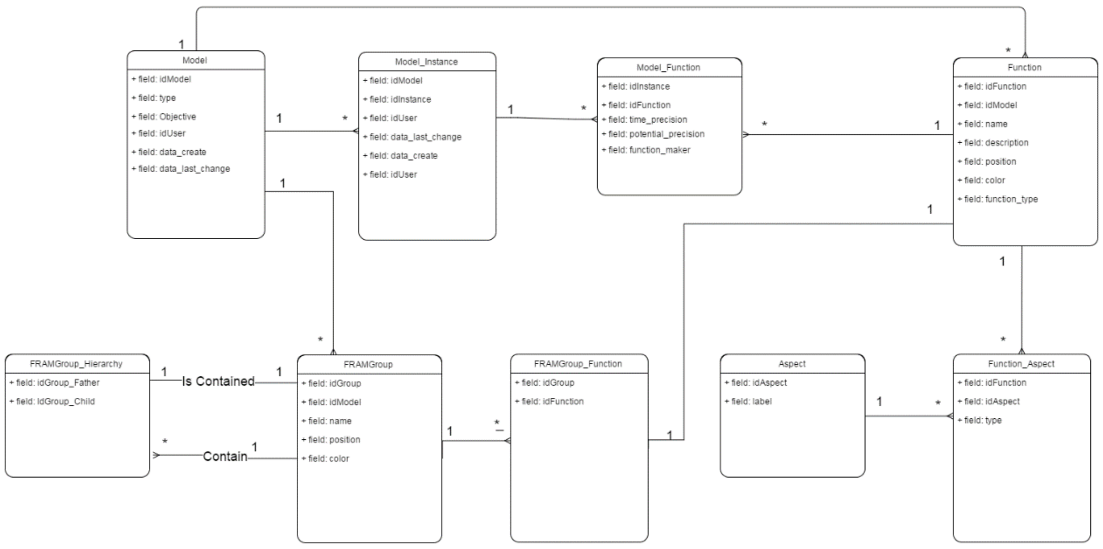


Figure 52. Resolute_DB complete database structure.

7.3 User Typologies

The users' typologies defined are the following:

- Simple user / Guest (Without registration)
- Advanced User

- Decision Maker (DM)
- Administrator

The access to the models management is enabled only for the registered users. With the registration is possible to give different functionalities to different type of users. A new registered user has the Advanced User privilege. The actions enabled for each user type are reported in the next paragraph.

Functionalities for the models: All the typologies of users can visualize the temporary models and those saved in the DB. The simple user (Guest) can't do any other operation. The advanced user can access to models available and modify. The DM behaviour is to create new models, delete existent models, modify them, and provide the weights for the models. The administrator can do the same operations as the decision maker. A list is reported in Table 7. User's Functionalities for manage the models.

Table 7. User's Functionalities for manage the models.

Functionalities	**Guest**	**Advanced**	**DM**	**Administrator**
Visualize Model	**V**	**V**	**V**	**V**
Create New Model			**V**	**V**
Save New Model			**V**	**V**
Modify a Model		**V**	**V**	**V**
Delete a model			**V**	**V**

Functionalities for model's instances. All the typologies of users can visualize the instances present temporarily in the DB. The simple user (Guest), as for the model, cannot do any other operation besides seeing the models. The advanced user can modify the available instances. The DM can see all the instances, create new one, modify the existent and has the possibility to set the weights and the template for the functions to export to SmartDS. Currently, the weights and template are not included, yet. The administrator can do the same action of the decision maker.

Table 8. Possible user's actions on the instances.

Functionalities	**Guest**	**Advanced**	**DM**	**Administrator**
Visualize Instance	**V**	**V**	**V**	**V**
Create New Instance			**V**	**V**
Save New Instance			**V**	**V**
Modify an Instance		**V**	**V**	**V**
Delete an Instance			**V**	**V**
Set weights for the Functions			**V**	**V**
Set templates for the Functions			**V**	**V**

Application Functionalities. The administrator is the unique user that has the permissions to see and modify all the profiles and related permissions.

Table 9. Application functionalities for the typologies of users.

Functionalities	**Guest**	**Advanced**	**DM**	**Administrator**
Manage users' permission				**V**

Following are reported the user case diagram for each typology.

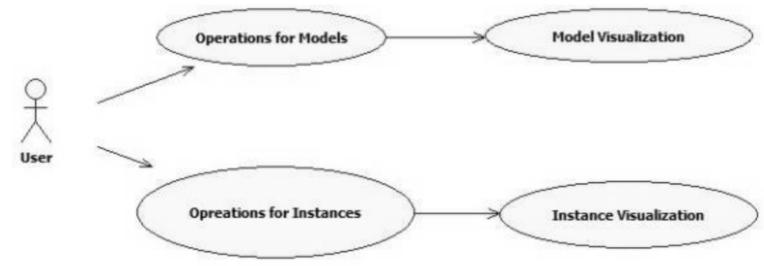


Figure 53. Guests functionalities

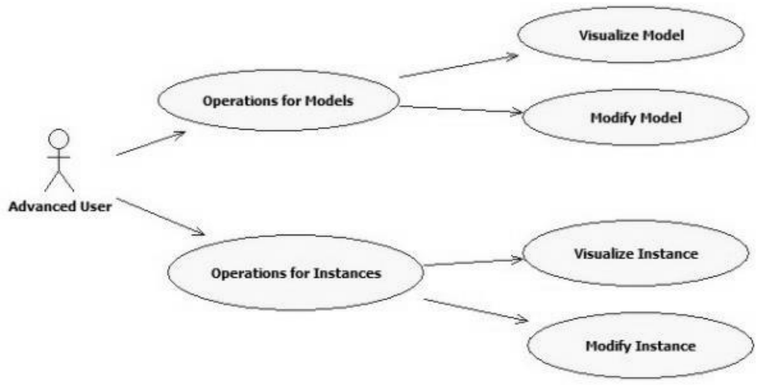


Figure 54. Advanced user's functionalities.

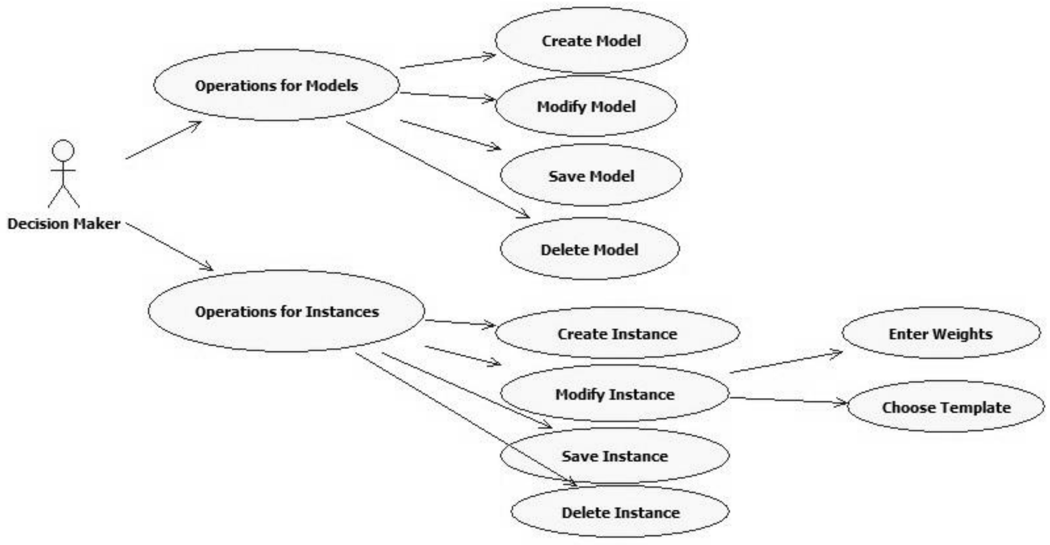


Figure 55. Decision Maker available actions.

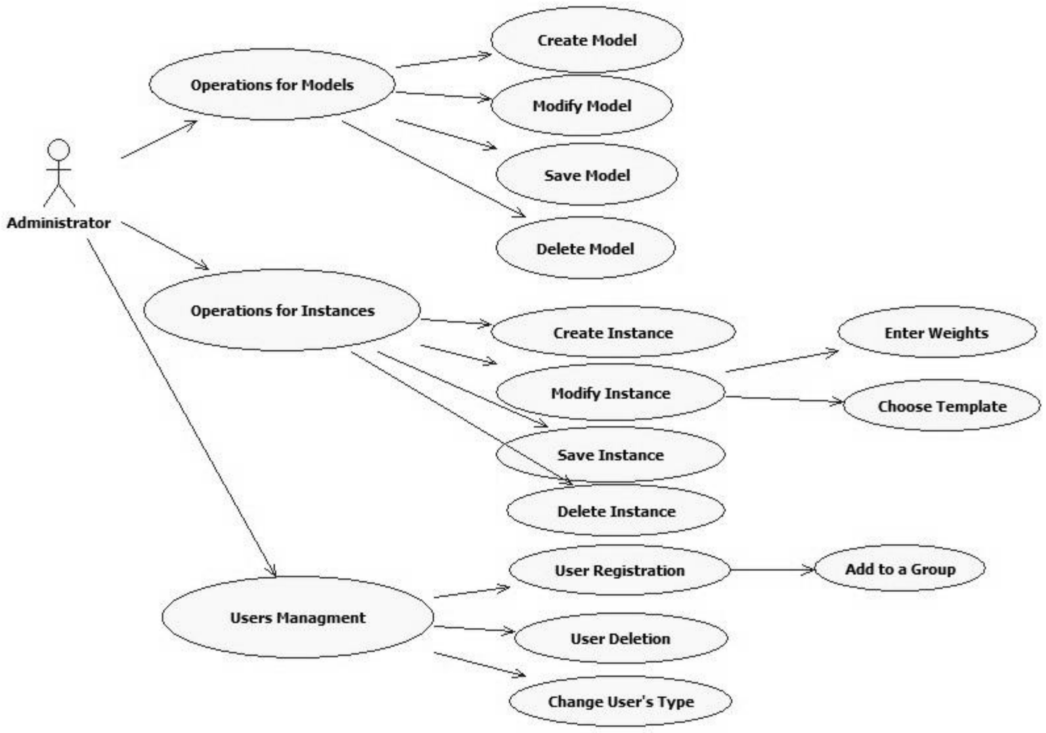


Figure 56. Administrator's functions. The same of DM, plus the users managment.

7.4 Server Side

The server has been realized in JAVA, using the **REST** paradigm (REpresentational State Transfer) for the definition of a uniform interface and a distinction between client and server. Its purpose is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability. The REST architecture defines the principle for data exchange between client and server where the resources are identified uniquely with **URI** (Uniform Resource Identifier). This paradigm uses the HTTP methods: GET, POST, DELETE and PUT requests, for make the simple CRUD operations: Create, Retrieve, Update and Delete. REST, respect to SOAP, permits to use the HTTP operations to define the operations over the resources, without the use of methods or dedicate interface for information access. Unlike SOAP-based web services, there is no "official" standard for RESTful web APIs. This is because REST is an architectural style, while SOAP is a protocol. Even though REST is not a standard per se, most RESTful implementations make use of standards such as HTTP, URI, JSON, and XML. For exchange data we have used the XML for encoding the resources.

The principal classes defined are six: **Models**, **FRAMModel**, **ModelInstances**, **FRAMModelInstance**, **ModelExtraOperation**, **ModelExtraOperations**. The first two are needed for make direct operations over the FRAM models, like creation, loading, editing and deleting. **ModeInstances** and **FRAMModelInstance** do the same operations as the previous but for the model's instances. The last two defines the other possible operations for models and instances. These functions are: save, set and get the identifier and set and get the description.

Furthermore, we have defined other seven resources: **Function**, **Aspect**, **FunctionAspect**, **Group**, **FunctionInstance, ExportFRAMModelResource** and last, **FMVModel**. As possible to deduce from the name, these are all classes for manage the components of a model. **Function**, **Aspect** and **Group** are respectively for the operations over functions, aspects and groups of functions. Specifically, **Group** is used to create a hierarchy on the model and contains the list of functions and groups in. In the **Aspect** are saved only the essential information: label, id and ModelId, the others are handled by **FunctionAspect**. Consequently, an aspect could link different functions by different entry (an aspect is usually the function's output and could be an input, a precondition, a time, a control or a time of one or more different functions), this resource is needed for retrieving the source, the target function and the entry type (one among input, precondition, output, etc.). **FunctionInstance** is used for retrieve the data about a specific function's instance: the time variability, the precision variability and the function's handler.

For exporting a Model to SmartDS, we have decided to create a different class, **ExportFRAMModelResource**. This is used for retrieving the model's information from the client and creating the SmartDS's tree structure to export.

The last class, FMVModel, is needed to interact with models created in the FMV tool (FMV s.d.). With this, it is possible to import a FMV file (.xfmv) into the ResilienceDS.

The REST calls for the model Operations are reported in Annex V.

For this request are needed two parameters: {model_id} and {output_id}. The first is the target's model, which contains the functions to study. The second, 'output_id', is the identifier of the output's aspect that we want to analyse the behaviour. We report here an example to explain how the export works.

Consider the FRAM's model in Figure 57. FRAM export example with three functions, and a cycle. In this model, we have a short representation of an events chain for the management of a help call by a rescue organization. There are 3 functions: One for handling requests for help ('Rescue calls handling' in green), one for managing the persons and the vehicles of the rescue force ('Handling team' in blue) and the last one is the function that activates the team to send ('Team activation' in black).

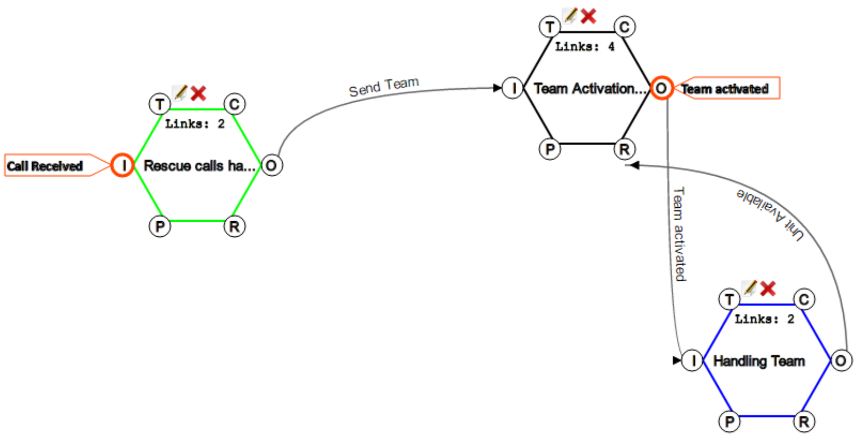
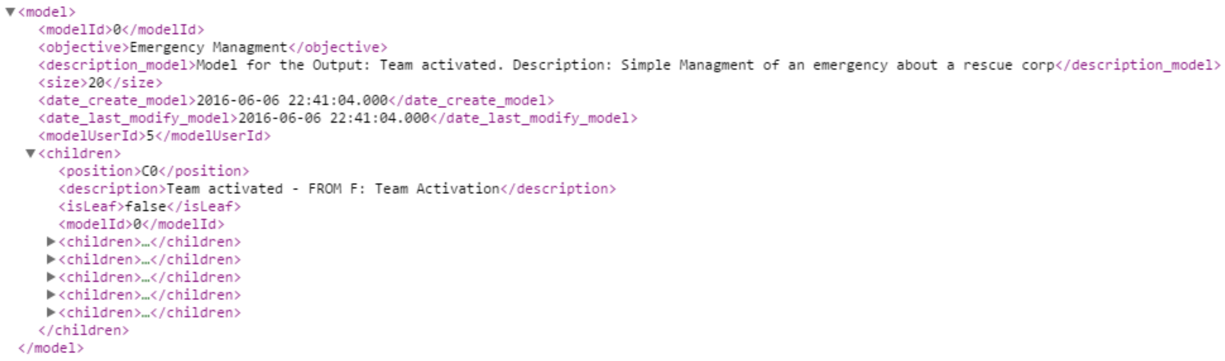


Figure 57. FRAM export example with three functions, and a cycle.

When a call is received, the green function listens to the request, evaluates, decides which type of operation is needed and sends a team. Basically, this is a function of an operations room. The black function activates the forces based on the available units provided as resources from the blue one. When the team is ready it's activated and it starts the rescue operation. When a team is at work, the functions that manage the team have to be updated, in the model this is the job of the 'Team Activated' edge between the black and blue hexagons.

For this model, is useful to export the output's aspect 'Team activated' to study its behaviour, especially for analysing when it is possible to do that. Utilizing the REST's call above, we send the necessary information to the server to build the SmartDS's model. Following there is the correspondent XML for the model's aspect studied and coherent with the Analytical Hierarchical Process behind the tool.



```
▼<model>
   <modelId>0</modelId>
   <objective>Emergency Managment</objective>
   <description_model>Model for the Output: Team activated. Description: Simple Managment of an emergency about a rescue corp</description_model>
   <size>20</size>
   <date_create_model>2016-06-06 22:41:04.000</date_create_model>
   <date_last_modify_model>2016-06-06 22:41:04.000</date_last_modify_model>
   <modelUserId>5</modelUserId>
  ▼<children>
     <position>C0</position>
     <description>Team activated - FROM F: Team Activation</description>
     <isLeaf>false</isLeaf>
     <modelId>0</modelId>
    ▶<children>…</children>
    ▶<children>…</children>
    ▶<children>…</children>
    ▶<children>…</children>
    ▶<children>…</children>
   </children>
 </model>
```

Figure 58. XML exported, relative to the Rescue management FRAM's Model.

The server creates this structure in the following manner. Starting from the goal, it retrieves the function that has this aspect has output. For this block, finds its input's aspects: inputs, preconditions, resources, times and controls. These aspects are in turn associated to their function and studied as the goal. This process ends when an aspect is reached that was generated by a background function (in the study case the input 'Call Received' of 'Rescue calls handling' block) or when a function has no inputs. But, as shown in the example, there can be cycles in the structure (the blue and black blocks call each other). These cycles occur when one function depends on another one in different time stamps or steps. The problem is that a cycle is translated into an infinite branch of the tree. To prevent this, along the path to the leaf, when a propagation is observed that could generate a cycle, the search is stopped. The next function in the path is added as a leaf and is marked with a '-1' to indicate that is the result from a previous step.

The final tree for the example reported is shown in the figure below.

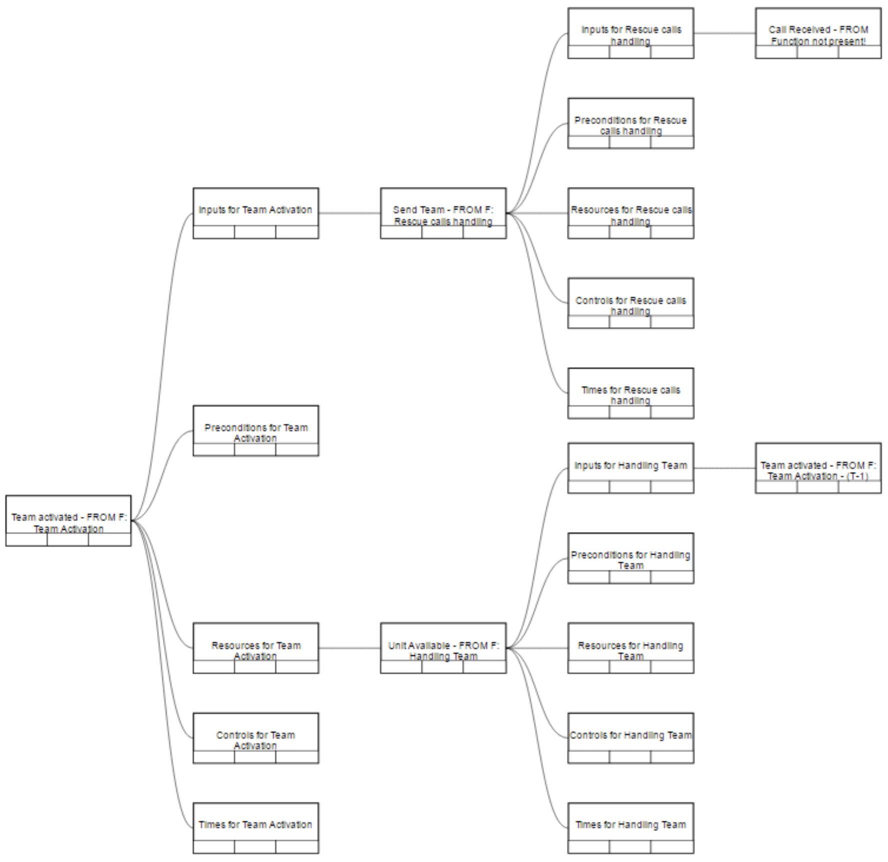

Inputs for Rescue calls handling
Call Received - FROM Function not present!
Preconditions for Rescue calls handling
Inputs for Team Activation
Send Team - FROM F: Rescue calls handling
Resources for Rescue calls handling
Controls for Rescue calls handling
Times for Rescue calls handling
Preconditions for Team Activation
Inputs for Handling Team
Team activated - FROM F: Team Activation - (T-1)
Team activated - FROM F: Team Activation
Preconditions for Handling Team
Resources for Team Activation
Unit Available - FROM F: Handling Team
Resources for Handling Team
Controls for Team Activation
Controls for Handling Team
Times for Team Activation
Times for Handling Team

Figure 59. SmartDS's tree for the Rescue Example.

Now we are going to see how is structured the server. The server side is essentially divided into two modules:

- FRAM module
- User module

The FRAM module manages the models and the instances. The main operations are:

- Creation of a model / model's instance.
- Modify of models / models' instances.
- Save of a model / model's instance.
- Load of a model / model's instance.
- Import of a FMV model.
- Export of a model to SmartDS.

This module interfaces with the relational database to save and retrieve models. Also, even if not exploited, the interface methods to the RDF store are present. In Figure 60 the complete schema of ResoluteDS server side is reported.

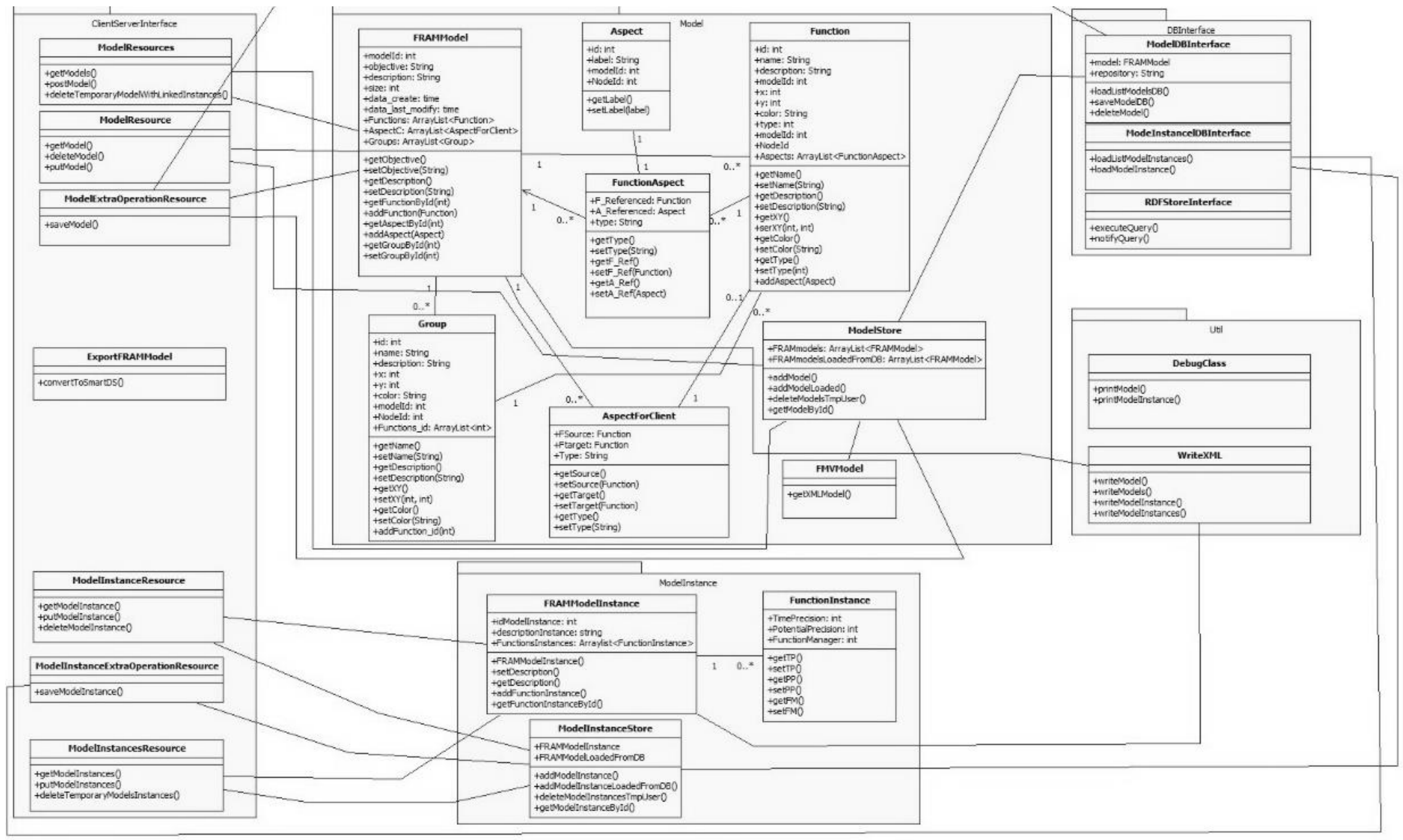


Figure 60. ResoluteDS Server schema

There are 5 packages, logically divided by operation's type done: **ClientServerInterface**, **Model**, **ModelInstance**, **DBInterface** and **Util**.

In the first the classes for interfacing client and server are present. These pick up the REST requests and subsequently call the appropriate functions in the packages **Model** and **ModelInstance**. In these packages, there are all the possible operations over the FRAM models and instances. The classes for makes these operations are: **ModelExtraOperationResource**, **ModelInstanceExtraOperationResource**, **ModelResource**, **ModelInstancesResource**.

In the package **FRAMModel** there are the classes for load on server the models, a new model, or to modify an existent one. The main classes are **FRAMModel**, **FMVModel**, **Function**, **Aspect**, **AspectForClient**, **FunctionAspect**, **Group** and **ModelStore**. The class **FRAMModel** contains the model's identifier (id), the name, a description and a variable size for the functions number in. Furthermore, the list of Functions, the list of Aspects and the list of Groups are present . To keep track of the model's history the creation date is present, and also the date of the last modificatoin. The **FMVModel** is needed for translating a model created into FMV. The class **Function** contains the variables: id, name, description, type for background or foreground (set only when a FMV model is loaded) and the model-ID. For the information about the rendering these variables are present: x, y and colour. In this resource is also present a list of Aspects that come from, or goe to that function. The class **Aspect** contains the ID, the label and the model to which it belongs. The **AspectForClient** resource is, however, an extension of the previous one and it also contains the specific information about the aspect: the source function, the target function and the type, that is the role that the aspect plays in the relation. This class simplifies the management of the model's component with the client. **FunctionAspect** serves to connect the functions with the aspects. It contains the function's ID, the aspect's id and the aspect's type for that function. **Group** is like **Function** but for the groups. It has the attributes: *[id, name, description, colour, x, y, model-ID]*. Moreover, there is the list of functions and groups' identifiers contained. Finally, the class **ModelStore** has been introduced for temporarily storing the model on the server. Here are two array lists: FRAMmodels and FRAMmodelsLoadedFromDB. The first is for the new models created by the user and not saved yet. The second is for the models loaded from the DB. With the method in this class, it is possible to load any model by its id, modify it and eventually save.

Going on, in the package **ModelInstance** there are the classes and the methods for managing the instances, like for Model. The main classes are: **FRAMModelInstance** and **FunctionInstance**. These two functions extend the structure of **FRAMModel** and **Function** with other data. The first introduces the description of the instance, while the second holds more information about the functions: time precision, potential precision and the actor who performs that function. The last one has to be improved by entering other information like: the template for translating the model to SmartDS, the weight for the aspects, and the URL for the RDF's data associated.

The package **DBInterface** provides the classes for interfacing with the relational database and with the RDF store: **ModelDBInterface**, **ModelInstanceDBInterface** and **RDFStoreInterface**. The first contains the methods for saving and retrieving the model's data from MySQL. The second contains the same method as the first but for the instances. Otherwise, RDFStoreInterface has two methods; executeQuery() and NotifyEndQuery(), respectively for executing a SPARQL query in the RDFStore, and for notify the memorization of the function's result. Those two methods are implemented but not used, yet.

In the last package, **Util**, there are the classes: **DebugClass** and **WriteXML**. The first contains the methods printModel() and printModelInstance() to print, on the server, the data of a model or of a specific instance. The second implements the methods for writing the XML to send to the client.

Sequence Diagram for the models. Here are some sequence diagrams for explaining how the server side manages some operations. The first diagrams are for the operations in the package Model, used for executing the main functions on the FRAM models. The sequence in Figure 61 represents the steps for loading a model from the DB by an authorized user. When a user asks for a specific model, the REST request is managed by the method getModel in the class ModelResource. The last one calls the method loadModel of ModelDBInterface. Now the FRAMModel is created with all its data and it is saved on ModelStore for manage the next request directly on the server without the execution of queries on the DB.

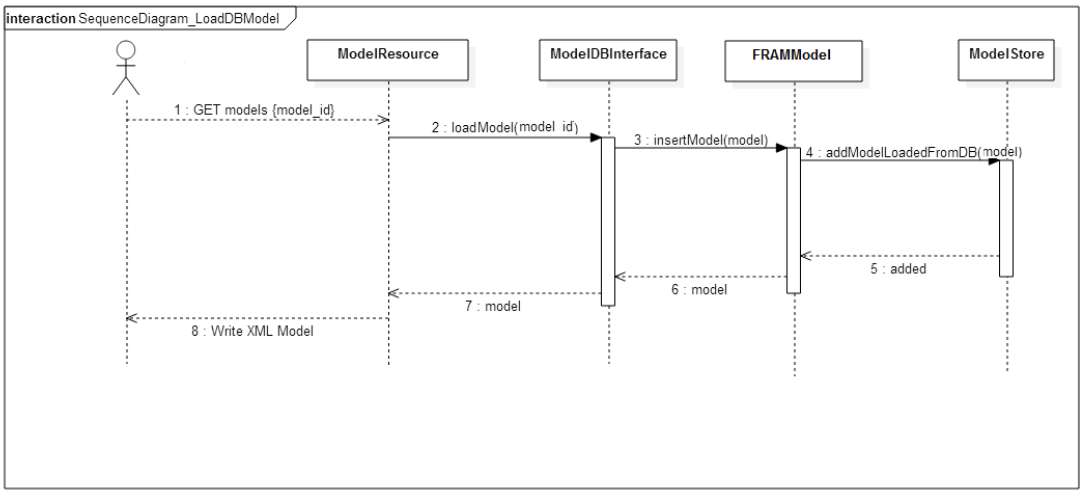


Figure 61. Sequence Diagram for loading a model from the relational DB.

The operation for saving a model in the DB is reported in Figure 62. When the user sends the request for saving the model, the call is managed by the server via the operation postOperation in the class ModelExtraOperationResource. This propagates the request to the class ModelDBInterface with the method saveModel().

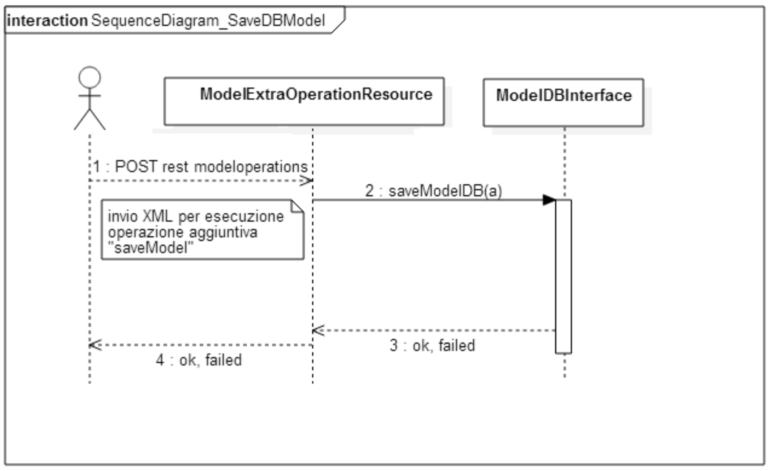

interaction SequenceDiagram_SaveDBModel
ModelExtraOperationResource
ModelDBInterface
1 : POST rest modeloperations
2 : saveModelDB(a)
invio XML per esecuzione operazione aggiuntiva "saveModel"
3 : ok, failed
4 : ok, failed

Figure 62. Sequence Diagram for Save the model in the DB.

For the models' instances, the sequence diagrams are similar to the previous one but managed by the class ModelInstances.

The UML for the User module is reported in Figure 63. The class **ManageUserInterface** serves as interface between the client application and the module for handling the users on the server side. It provides to the outside the functions login(), logout(), and register(), respectively used to authenticate a user (guest, advanced, decision maker or administrator), disconnect a user, or register a new one. Registering is possible only for an administrator. The administrator can choose the priority level (advanced, decision maker or administrator) for the new users. The **User** class holds the information about the user connected or to register and a vector of permits to access at the models. The vector contains boolean elements that identify the singular permission (true if the user can make that operation, or false).

Element 0.	Model creation.
Element 1.	Modify of a model present in the system.
Element 2.	Modify of an own model created.
Element 3.	Deleting a model from the system.
Element 4.	Deleting an own model.
Element 5.	Save a model.
Element 6.	Loading a Model from a server.
Element 7.	Loading an own Model created.

ManageUserDB has the methods for loading the user's data during the authentication, or for saving the data for a new registration. The methods that do this are respectively loadUser() and saveUser(). These functions are automatically called by the User class once entered the user's data: name, surname, nickname, password and email. This is possible thanks to the methods setName(), setSurname(), setNickname(), setPassword() and setEmail().

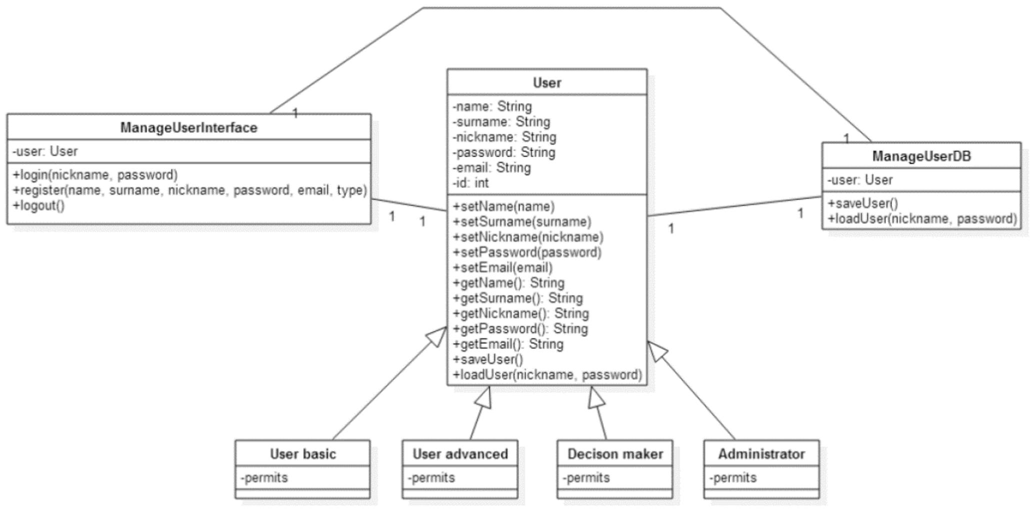


Figure 63. UML Diagram for the User Module.

In the Figure 64 the sequence diagram is represented for the registration of a new user in the system. The user has to choose an available nickname (not already present), then enter the requested data in the registration form and click 'Register'. With the click event the function register() is called, thus passing the user's parameters: name, nickname, password, email and type=2 that set the user as advanced. After this, the constructor of the superclass **UserAdvanced** is called to create the advanced user with their permissions. Finally the user is saved in the DB with the function saveUser() in the class ManageUserDB.

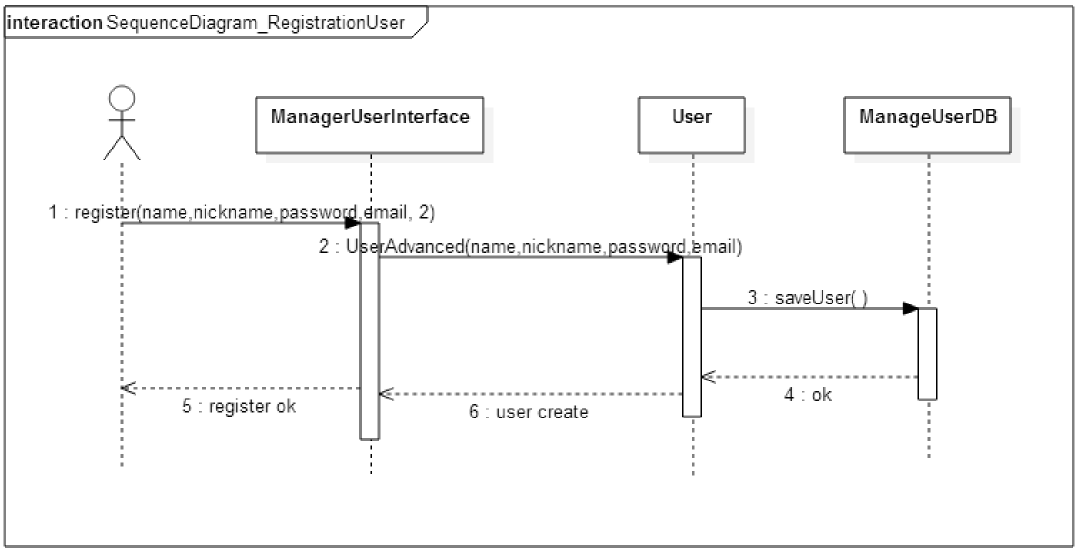

interaction SequenceDiagram_RegistrationUser
ManagerUserInterface
User
ManageUserDB
1 : register(name,nickname,password,email, 2)
2 : UserAdvanced(name,nickname,password,email)
3 : saveUser( )
4 : ok
5 : register ok
6 : user create

Figure 64. Sequence Diagram for register a new user.

In the next figure (Figure 65) the sequence diagram is shown for the steps that an administrator has to do to register a new user. This operation is divided in two steps: in the first, the administrator authenticates himself, in the second he enters all the data for the new user to register and the user's type. The client calls the function login() of **ManageUserInterface** to extract from the DB the user's information, if present. If the user is not present in the system, the returned value identifies the type of error and the registration interface is disabled. If the user is present, once obtained the information, a new element *Administrator* is created with the data of the new user authenticated. This is needed for maintaining in the session the user's information to keep it easily available and editable. Once that an administrator has entered all the data for the user to register, a click on the Register button calls the method register() with the entered data. Finally, through the class **User**, the **ManageDB**'s method saveUser() is called to save the new registered user in the DB.

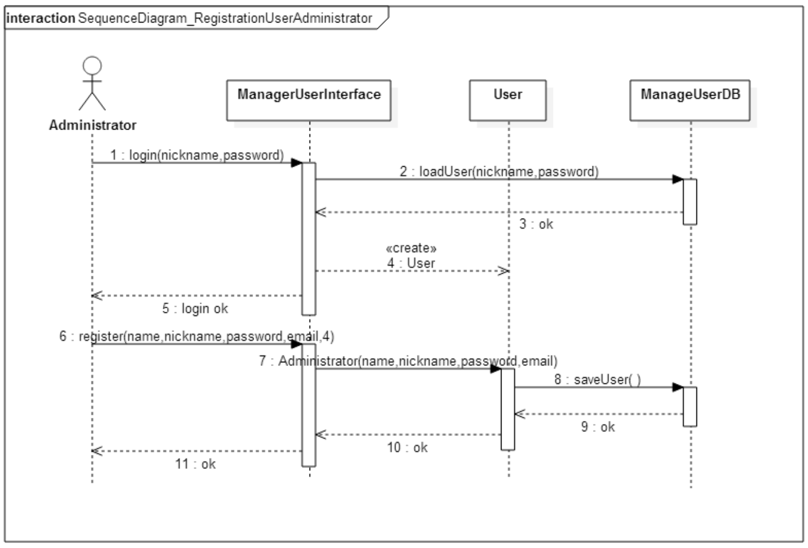

interaction SequenceDiagram_RegistrationUserAdministrator
Administrator
ManagerUserInterface
User
ManageUserDB
1 : login(nickname,password)
2 : loadUser(nickname,password)
3 : ok
«create»
4 : User
5 : login ok
6 : register(name,nickname,password,email,4)
7 : Administrator(name,nickname,password,email)
8 : saveUser( )
9 : ok
10 : ok
11 : ok

Figure 65. Registration of a new user by the administrator.

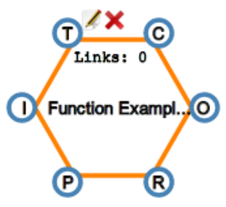
7.5 Client Side

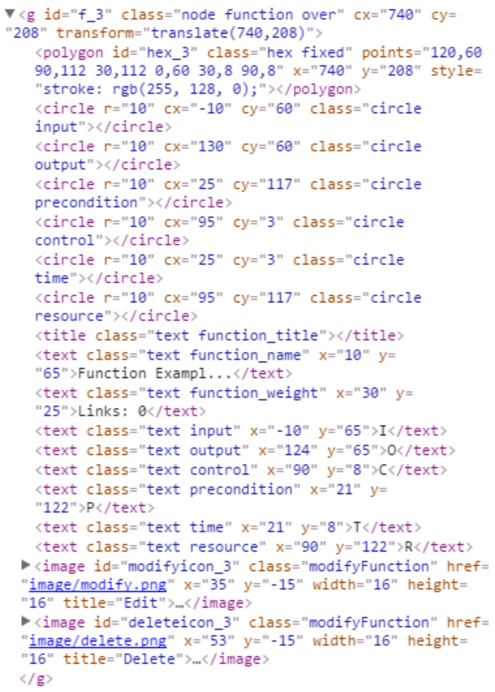
The client side manages the information about the user and the representation of FRAM models provided by the Server. It also handles the requests for the operations that a user can do on the server.

User accesses into Resilience DS with the login window as registered or as a guest. Based on the user typologies there are different functionalities that he can do. After the login, the main menu is displayed, with the features available based on the user's privilege. For the administrator, that has the maximum level of priorities, is possible to:

- Create new models and new instances.
- Visualize models and instances, also of other users. This is the only feature available for all usertypes.
- Save models and instances.
- Load models and instances.
- Delete models and instances.
- Handle users' permissions.

Creation of new models and instances. When a user requests to create a new model, the tool provides a window for inserting the model's name and description. After that a new frame is created with an editable starting function. All the functions\actions can be added that are needed to manage the C.I. system. Each of this actions is represented by the classical FRAM hexagon, with name, description and colour editable. Contrary to the FMV, each function is identified by an id, so it is possible to give the same name to different blocks.



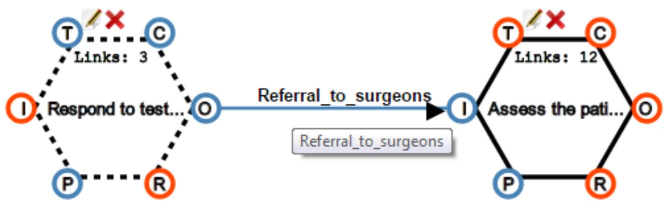


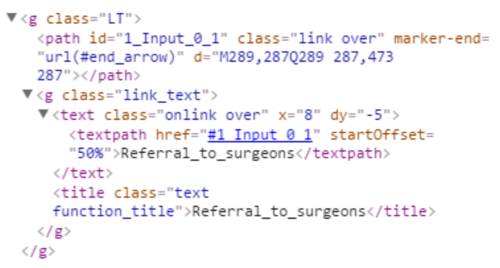
```
▼<g id="f_3" class="node function over" cx="740" cy=
"208" transform="translate(740,208)">
    <polygon id="hex_3" class="hex fixed" points="120,60
    90,112 30,112 0,60 30,8 90,8" x="740" y="208" style=
    "stroke: rgb(255, 128, 0);"></polygon>
    <circle r="10" cx="-10" cy="60" class="circle
    input"></circle>
    <circle r="10" cx="130" cy="60" class="circle
    output"></circle>
    <circle r="10" cx="25" cy="117" class="circle
    precondition"></circle>
    <circle r="10" cx="95" cy="3" class="circle
    control"></circle>
    <circle r="10" cx="25" cy="3" class="circle
    time"></circle>
    <circle r="10" cx="95" cy="117" class="circle
    resource"></circle>
    <title class="text function_title"></title>
    <text class="text function_name" x="10" y=
    "65">Function Exampl...</text>
    <text class="text function_weight" x="30" y=
    "25">Links: 0</text>
    <text class="text input" x="-10" y="65">I</text>
    <text class="text output" x="124" y="65">O</text>
    <text class="text control" x="90" y="8">C</text>
    <text class="text precondition" x="21" y=
    "122">P</text>
    <text class="text time" x="21" y="8">T</text>
    <text class="text resource" x="90" y="122">R</text>
  ▶<image id="modifyicon_3" class="modifyFunction" href=
  "image/modify.png" x="35" y="-15" width="16" height=
  "16" title="Edit">…</image>
  ▶<image id="deleteicon_3" class="modifyFunction" href=
  "image/delete.png" x="53" y="-15" width="16" height=
  "16" title="Delete">…</image>
  </g>
```

Figure 66. Function's SVG structure.

For each action it is also possible to specify the aspects, with the name and the type. We have five types of aspects input: Input (**I**), Precondition (**P**), Resource (**R**), Control (**C**) and Time (**T**). There is also one Output (**O**) that can link together 2 or more functions. It is possible not to provide both aspect's source and target, but only one of these. In that case, the relative function's circle of the aspect is highlighted and the label for the missed link is provided. Links can be entered in the function's edit menu or directly by tracing the link from two hexagons.

For coherence with the FRAM's theory, some controls are provided. It is not possible to add two aspects for linking two inputs (input, precondition, time, control or resource) or two outputs. The link is graphically built with a quadratic Bezier's curve by a svg:path.

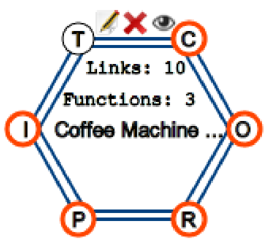


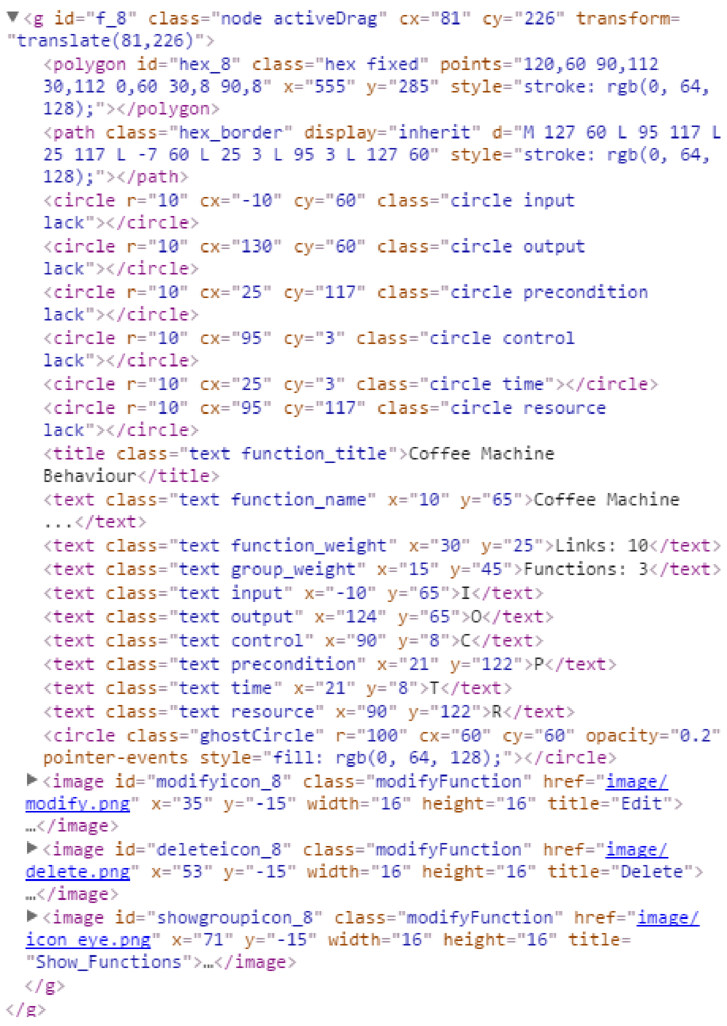


```
▼<g class="LT">
   <path id="1_Input_0_1" class="link over" marker-end=
   "url(#end_arrow)" d="M289,287Q289 287,473
   287"></path>
  ▼<g class="link_text">
    ▼<text class="onlink over" x="8" dy="-5">
       <textpath href="#1_Input_0_1" startOffset=
       "50%">Referral_to_surgeons</textpath>
     </text>
     <title class="text
     function_title">Referral_to_surgeons</title>
   </g>
 </g>
```

Figure 67. SVG code for an arch. The orange inputs of the functions represent missed aspects (Aspects with only the target or the source).

A user can also create the group element, useful to group together similar functions. This cluster is like an action element in terms of attributes, in fact it has a name, a description and a colour. A group can contain as many functions as the user wants and also other groups, but not vice versa. A function can be part of only one group, not two different groups.





```
▼<g id="f_8" class="node activeDrag" cx="81" cy="226" transform=
"translate(81,226)">
    <polygon id="hex_8" class="hex fixed" points="120,60 90,112
    30,112 0,60 30,8 90,8" x="555" y="285" style="stroke: rgb(0, 64,
    128);"></polygon>
    <path class="hex_border" display="inherit" d="M 127 60 L 95 117 L
    25 117 L -7 60 L 25 3 L 95 3 L 127 60" style="stroke: rgb(0, 64,
    128);"></path>
    <circle r="10" cx="-10" cy="60" class="circle input
    lack"></circle>
    <circle r="10" cx="130" cy="60" class="circle output
    lack"></circle>
    <circle r="10" cx="25" cy="117" class="circle precondition
    lack"></circle>
    <circle r="10" cx="95" cy="3" class="circle control
    lack"></circle>
    <circle r="10" cx="25" cy="3" class="circle time"></circle>
    <circle r="10" cx="95" cy="117" class="circle resource
    lack"></circle>
    <title class="text function_title">Coffee Machine
    Behaviour</title>
    <text class="text function_name" x="10" y="65">Coffee Machine
    ...</text>
    <text class="text function_weight" x="30" y="25">Links: 10</text>
    <text class="text group_weight" x="15" y="45">Functions: 3</text>
    <text class="text input" x="-10" y="65">I</text>
    <text class="text output" x="124" y="65">O</text>
    <text class="text control" x="90" y="8">C</text>
    <text class="text precondition" x="21" y="122">P</text>
    <text class="text time" x="21" y="8">T</text>
    <text class="text resource" x="90" y="122">R</text>
    <circle class="ghostCircle" r="100" cx="60" cy="60" opacity="0.2"
    pointer-events style="fill: rgb(0, 64, 128);"></circle>
  ▶<image id="modifyicon_8" class="modifyFunction" href="image/
  modify.png" x="35" y="-15" width="16" height="16" title="Edit">
  …</image>
  ▶<image id="deleteicon_8" class="modifyFunction" href="image/
  delete.png" x="53" y="-15" width="16" height="16" title="Delete">
  …</image>
  ▶<image id="showgroupicon_8" class="modifyFunction" href="image/
  icon_eye.png" x="71" y="-15" width="16" height="16" title=
  "Show_Functions">…</image>
  </g>
</g>
```

Figure 68. SVG code for a group.

Practically the groups are like macro functions which allow for setting aspects. This option has some constraint, in fact each aspect has to be connected to the group's functions. For this reason, it is not possible to have aspects for an empty group. All the aspects that go from the broad to the group's functions are grouped and shown with a single arch. Instead, the link between functions in the group is hidden. For usability, a function has been implemented for showing (on click) the group's components in the graph. The eye over the group is the feature's trigger.

The instance creation is like the previous. Resilience DS provides a window for enter the objective and then loads the model to analyse. The model is not editable from the instance, it's possible to move the elements but just temporarily until the next load. However, it is possible to set the variability of each function. Groups don't have their variability, but they reflect their components.

Visualization of models and instances. This is the only option present for all kinds of users. This option permits to display a FRAM model among those saved in the DB.

Save of models and instances. The models could be saved at any moment, even during the work. The models and the instances are saved in the DB and temporarily on the server for retrieval without any queries.

Loading and reediting of models and instances. This menu item permits to choose a model or an instance from the list and load it for editing the structure. The user can do one of the following operations on a loaded model.

- Add or delete functions.
- Add or delete groups.
- Add or remove functions and groups in a group.
- Modify the parameters of a function: name, description or colour.
- Add or delete aspects from a model.
- Modify an existent aspect: the label, the source, the target or the type.

For an instance, during the modification is possible to:

- Change the function's handler.
- Change the function's time precision.
- Change the function's potential precision.

Deleting models and instances. Only a user with the respective privilege can delete saved models. Once that a model is deleted it cannot be recovered because all the relative data are deleted from the DB. All the linked functions, aspects and instances to the model deleted are also deleted. Additionally, a single instance can be deleted.

Manage User's permission. The administrators can manage the users' permissions and have the possibility to modify their user type variable.

Application. The client side application has been developed in JavaScript with the **D3** graphical library and the **jQuery** framework.

The D3 library, available at (D3 2016) allows binding arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, it is possible to use D3 to generate an HTML table from an array of numbers. It is also possible to use the same data to create an interactive SVG bar chart with smooth transitions and interaction. D3 is not a monolithic framework that seeks to provide every conceivable feature. Instead, D3 solves the crux of the problem: efficient manipulation of documents based on data. This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as HTML, SVG, and CSS. With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviours for interaction and animation. D3's functional style allows code reuse through a diverse collection of components and plugins.

jQuery, (jQuery 2016), is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

The client application is structured in levels, shown in Figure 69. At main level, we have the system folder and the .html, .jsp files for the user interface. The application folder is composed of:

- **CSS**: contains the style sheet (.css files) for the graphical management of the interface's elements.
- **Fonts**: provides the fonts utilized.
- **Image**: is the folder where all the images used in the application are saved.
- **Js**: contains the JavaScript necessary for the interface to behave correctly.
- **Lib**: this folder rallies all the external libraries (d3, jquery) necessary for the tool.
- **META-INF / WEB-INF**: are intern folders of the system.

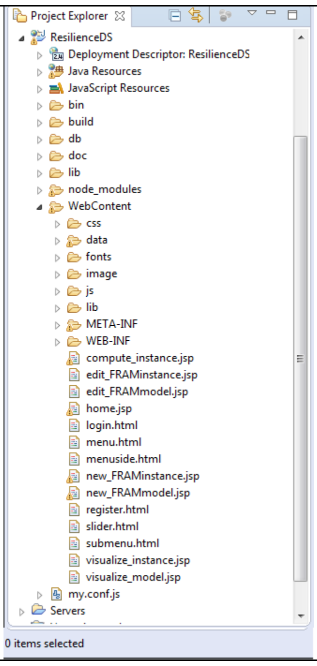
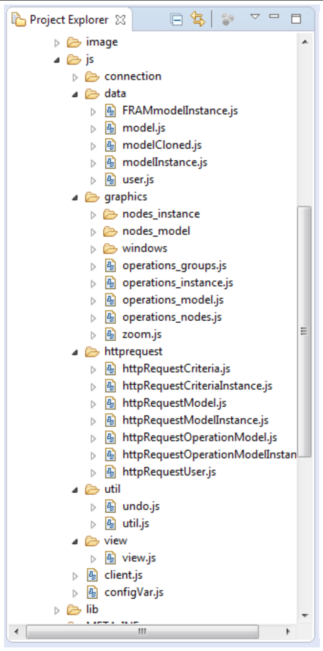
	

Figure 69. ResoluteDS project's files.	**Figure 70. ResilienceDS, JavaScript files that provides all the tool's functions.**

```
Project Explorer
ResilienceDS
  Deployment Descriptor: ResilienceDS
  Java Resources
  JavaScript Resources
  bin
  build
  db
  doc
  lib
  node_modules
  WebContent
    css
    data
    fonts
    image
    js
    lib
    META-INF
    WEB-INF
    compute_instance.jsp
    edit_FRAMinstance.jsp
    edit_FRAMmodel.jsp
    home.jsp
    login.html
    menu.html
    menuside.html
    new_FRAMinstance.jsp
    new_FRAMmodel.jsp
    register.html
    slider.html
    submenu.html
    visualize_instance.jsp
    visualize_model.jsp
  my.conf.js
Servers
0 items selected
```

```
Project Explorer
image
js
  connection
  data
    FRAMmodelInstance.js
    model.js
    modelCloned.js
    modelInstance.js
    user.js
  graphics
    nodes_instance
    nodes_model
    windows
    operations_groups.js
    operations_instance.js
    operations_model.js
    operations_nodes.js
    zoom.js
  httprequest
    httpRequestCriteria.js
    httpRequestCriteriaInstance.js
    httpRequestModel.js
    httpRequestModelInstance.js
    httpRequestOperationModel.js
    httpRequestOperationModelInstan
    httpRequestUser.js
  util
    undo.js
    util.js
  view
    view.js
  client.js
  configVar.js
lib
```

The components of the **js** folder are shown in Figure 70. There isthe **data** folder containing the data structure for the models and the instances. The **graphics** folder has an important role for managing the graphical interface for the structure visualized. All the operations over the functions, aspects and groups are handled by operations_model.js, operations_node.js, etc. **httprequest** manages the REST requests and responses. For the utility functions (like the undo and the XML creation) and for the graphical operations there are respectively, the **util** and the **view** folders. In the file **configVar.js** there are the configuration variables for the application. Last, there is **client.js** which is dedicated to making the REST requests to the server.

7.6 Resilience DS UI

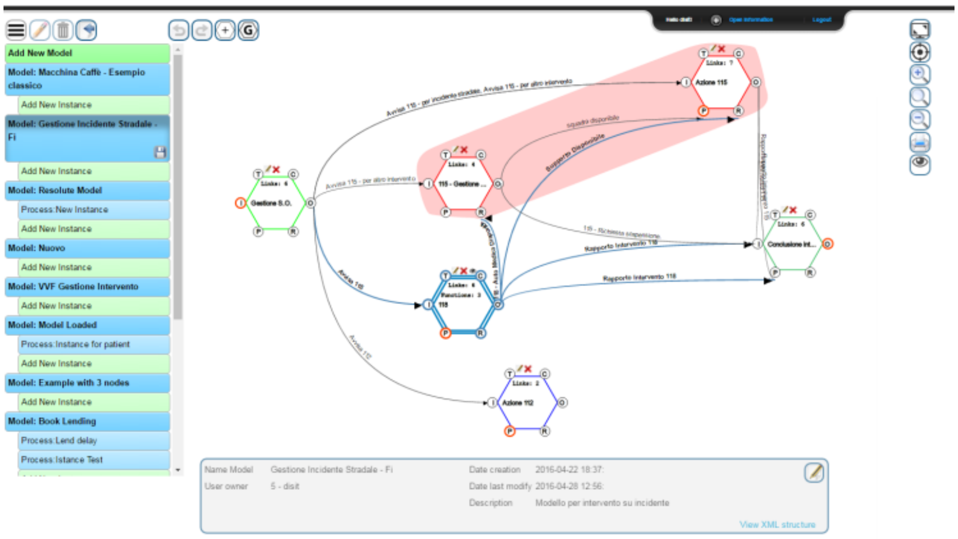
The state of the art of FRAM-based tools is well represented by FRAM Model Visualizer (FMV) [27]. The FMV is a FRAM editor that supports the FRAM model design. The FMV tool is only focused on realizing pdf documents generating the tables used to provide the descriptions of the functions and the aspects. The FMV tool does not support any kind of computations or consistency check, and remains at descriptive level without providing computational aspects. Other tools, such as Cambrensis [25], adopt similar approaches by adding some operational aspects with limited capabilities as described in the following. The Resilience DS tool presents a number of improvements with respect to the classical FRAM model and FMV tool such as:

- Increasing model expressiveness by allowing the hierarchical modelling of FRAM functions, introducing the Macro FRAM Function, **MFF** (described in the following).
- Introducing a number of rule based formal checks to assess the completeness, consistency and the complexity of Functions, and MFF; They can be used to guide the designer and the experts to create complete and consistent modelling and dedicating more attention to more complex aspects.
- modelling the outputs of a FRAM function as a Bayesian Decision Support process. For example, exploiting System Thinking (http://smartds.disit.org) approach.
- Connecting the FRAM function output estimation on the basis of a connection to real data from the smart city and critical infrastructure data (statistical and/or real time data as well).

This set of improvements has transformed the FRAM model to an operational paradigm allowing the operational execution of the model in real time according to the changes occurring in the city on the basis of the events.

7.6.1 Resilience DS Features.

In Figure 71, the Resilience DS tool is presented showing the editor for FRAM and Macro FRAM Functions (**MFF**).



Name Model: Gestione Incidente Stradale - Fi
User owner: 5 - disit
Date creation: 2016-04-22 18:37:
Date last modify: 2016-04-28 12:56:
Description: Modello per intervento su incidente

Figure 71. Resilience DS: FRAM and Macro FRAM Functional Tool editor. A simple example with 8 functions and 2 groups.

The Resilience DS allows creating a FRAM Model providing:

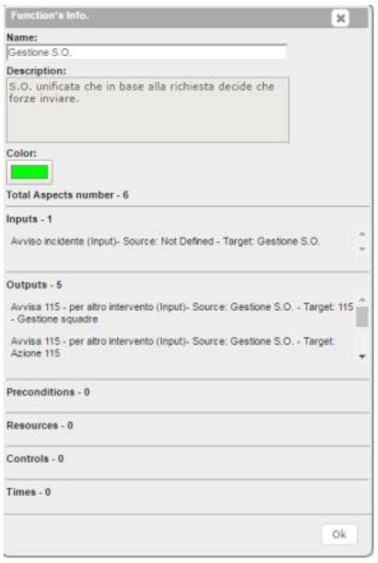
- a name and a description, and
- a combination of

    - FRAM Functions as hexagons;
    - MACRO FRAM Functions, MFF, as hexagons with double line border (modelling internal subgraphs of FRAM Functions as hexagons). Each Functions as hexagons presents 6 Aspects: **I**, **P**, **R**, **O**, **T**, **C**;
- a set of links connecting vertex of the hexagons.

According to the FRAM model:

- Function Type: who performs the function, it can be a person ('Human'), a group of persons ('Organization') or otherwise a 'Technological' part.
- Potential Output variability with regard to Time: the possible values are 'Too Early', 'In Time', 'Too Late' or even 'Not at All' when it's not sure that the function can complete in the expected time.
- Potential Output variability with regard to Precision, which can be: 'Precise', 'Acceptable' or 'Imprecise'. Several Outputs can be defined, and can be produced by a FRAM Function.

To study the behaviour of a model with respect to its variability, the Resilience DS provides the instances. When an instance is created, it is always possible to change the model, without loss of information, except for the function eventually deleted. Thus, the Resilience DS allows creating a hierarchical FRAM model.
A Resilience DS hexagon (Function or Macro Function) has attributes, like: name, a description and colour.


Function's Info.

Name:
Gestione S.O.

Description:
S.O. unificata che in base alla richiesta decide che forze inviare.

Color:
🟩

Total Aspects number - 6

Inputs - 1

Avviso incidente (Input)- Source: Not Defined - Target: Gestione S.O.

Outputs - 5

Avvisa 115 - per altro intervento (Input)- Source: Gestione S.O. - Target: 115 - Gestione squadre

Avvisa 115 - per altro intervento (Input)- Source: Gestione S.O. - Target: Azione 115

Preconditions - 0

Resources - 0

Controls - 0

Times - 0

Ok

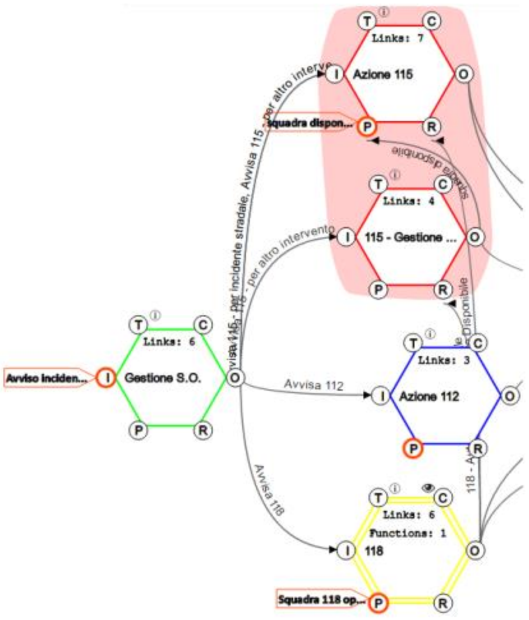
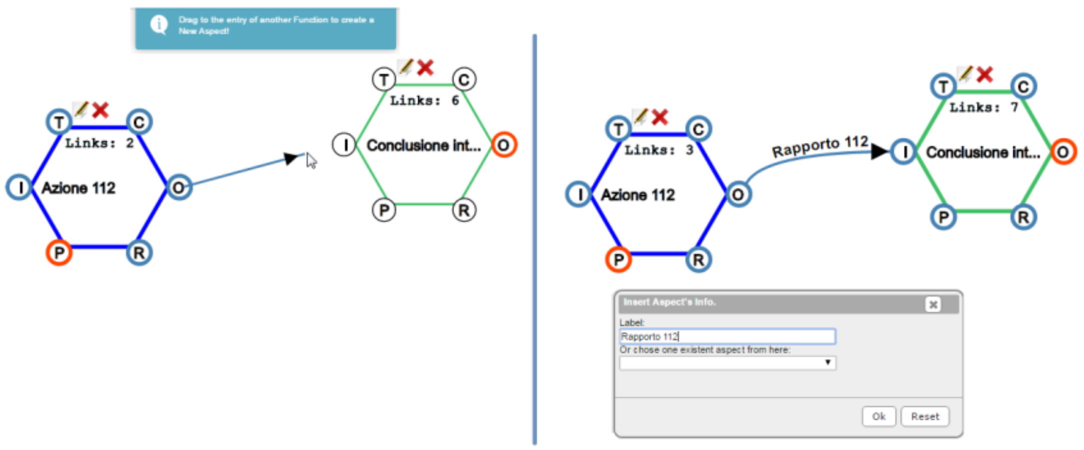


Figure 72. Function in Resilience DS. The Window reports the function's info (for the green one): name, description, colour and the aspects divided by typology. The aspects with source or target undefined are highlighted and a label is shown on click.

Once created, the user can delete or edit it (with the pencil icon or by double clicking the function). The Figure 72 shows the information panel, that is the same for the editing. The links can be created on the edit panel of a function, or by dragging the link from the circle that corresponds to the aspect selected, to the entryway of another function.


Drag to the entry of another Function to create a New Aspect!

Insert Aspect's Info.

Label: Rapporto 112

Or chose one existent aspect from here:

Figure 73. The Resilience DS creation of an aspect.

The Aspects are visualized on the screen and reported on the function's edit page, where it is also possible to modify or delete them. In FRAM, each link has always a source and a target Function aspect. The only links that may provide a missing terminal are those that have to be closed with the external information. Naturally, it is not possible to create a connection between two inputs (Like from a Precondition to a Resource) but only from an Output to one input, or vice versa. Therefore, it is possible that a function's Output goes into different functions or in the same but in various entries. The Resilience DS tool controls the model's completeness and consistency, and avoids user's distractions or other wrong behaviours, like duplication of aspects. Each function has a different identifier; also the aspects have an ID. This, fact is a major difference with respect to FMV where the links are univocally identified by their names. In our case, a link is identified by a unique number, and considering that the same link can be shared among many Functions (and Aspect), in the visualization is managed with the quadruple:

{*ID, source Function, target Functions, target Aspect*}.

The last one is the entryway's type (Aspect) of the Function. The FRAM Function vertex output is the only source for any other of the five Function's inputs (Aspects) to other hexagons. So, it is not discriminant for linking (the source Function is always the Output Aspect).

In order to improve FRAM modelling expressivity and capability of managing complexity by using FRAM graphs, a Macro FRAM Function (**MFF)**, concept has been created. The MFF is used to reorganize connected Functions, for example those that interest the same C.I., or which are addressed by the same organization. In Figure 74 is shown how to create a group and what it looks like. In the example, representing the management of an emergency call by the 115 (the Italian fire-fighters, VVF), the 2 functions of 115 can be grouped and handled together. The correspondent MFF, 'VVF management', is created.

The Functions can be added just dragging and dropping them into the group. An MFF is like a mini-model or a macro-function that can be split in sub components. The only constraint for maintaining consistency in the model is that this object doesn't have its own aspects. So when we try to add a new aspect, the tool asks to choose a component of the group. If the cluster is empty, the program returns an error message. It is possible to add links between the functions of a MFF. These are only internally displayed. Whereas, aspects that come from a function outside to another one inside the group are grouped by the Aspect selected, and shown like entering the group. Once created the user can edit and delete MFFs. Figure 75 shows the steps to add the two 115's functions in the MMF. The visualization is constantly updated.

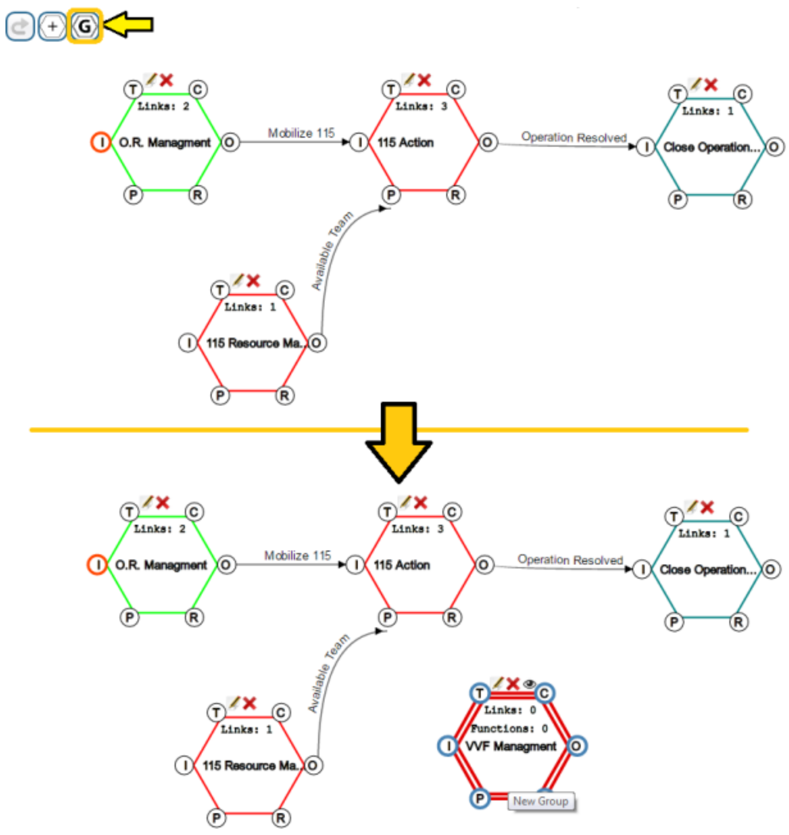


Figure 74. Group Creation. In the example is created the group 'VVF Management' that handle the fire brigade actions.

The attributes of MFF are: name, description and colour. Inside an MFF it is possible to have functions but also other groups as well. This implements a multi-level hierarchy in the model. As a constraint, an MFF can belong to at most one other MFF and it cannot be inside two different on the same level.

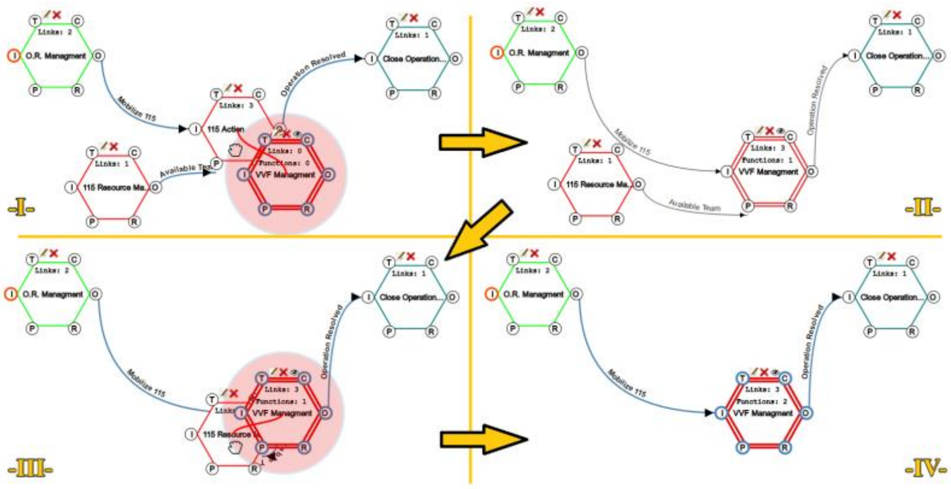
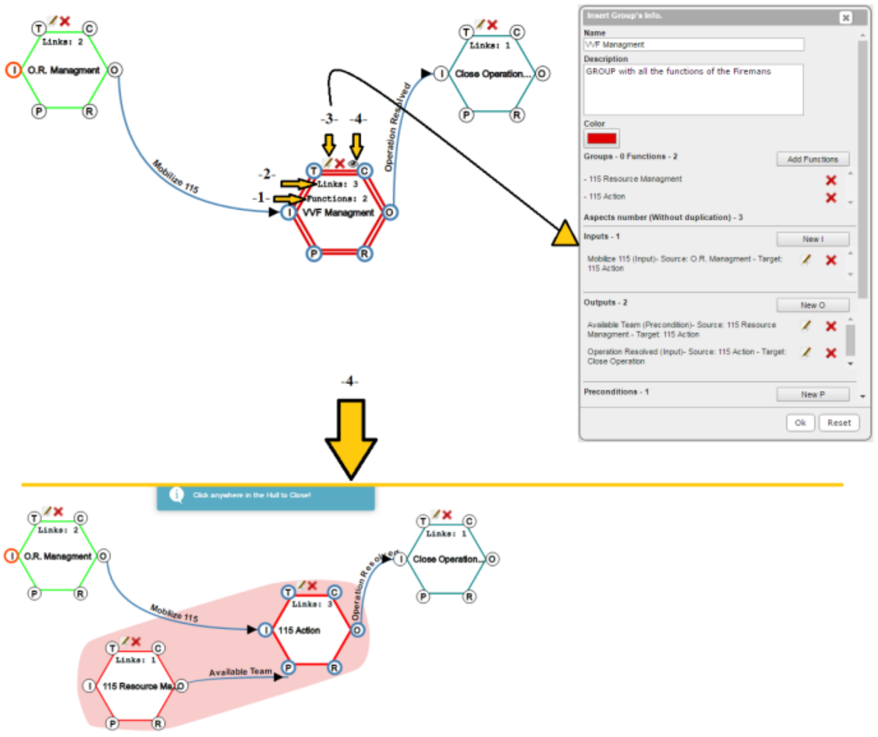


Figure 75. Add functions into a group. In the first frame (-I- on the top-left) is dragged the function '115 Actions' into the group. During the move for each group is showed a hull around the group and when a functions enter the hull is linked with the group to

The Figure 76 shows all the options and measures enabled for the MMF, most of them are similar to the Function. In the hexagon the number of functions and links of the group are reported. On top of it, there are the options to edit and delete it and also to show the functions inside. The last option creates a hull containing the actions and the groups of the MMF and shows the existent links. From the edit panel, it is also possible to add a function, or an aspect coherently with the FRAM constraints.

Last, the delete option allows to delete the MMF with all its contents, or to delete only the group and keep the functions inside.



Insert Group's Info.

Name
VVF Managment

Description
GROUP with all the functions of the Firemans

Color

Groups - 0 Functions - 2 | Add Functions
- 115 Resource Managment
- 115 Action

Aspects number (Without duplication) - 3

Inputs - 1 | New I
Mobilize 115 (Input)- Source: O.R. Managment - Target: 115 Action

Outputs - 2 | New O
Available Team (Precondition)- Source: 115 Resource Managment - Target: 115 Action
Operation Resolved (Input)- Source: 115 Action - Target: Close Operation

Preconditions - 1 | New P

Ok | Reset

Figure 76. Group's Options. -1- and -2- reports respectively the number of links and functions for the group. -3- are the edit and delete options. Clicking on edit, the editing window is displayed and reports all the info of the group: name, description, colour, f

The result is a simplified view, but at any time it is possible to show the functions inside a MFF/group without deleting it, just clicking on the eye icon over the relative hexagon. Above, we have reported an example of the utility of this feature. Let us consider the FRAM model created in the European Resilience Management Guidelines (Bellini, Gaitainidou e Fereira 2016). The model has been imported into the Resilience DS, as showed in Figure 77. All the properties of the functions and aspects, like; name, description, colour, as well as positions are the same of FMV. The only things changed are the identifiers of the elements. This model can be very concentrated and messy.

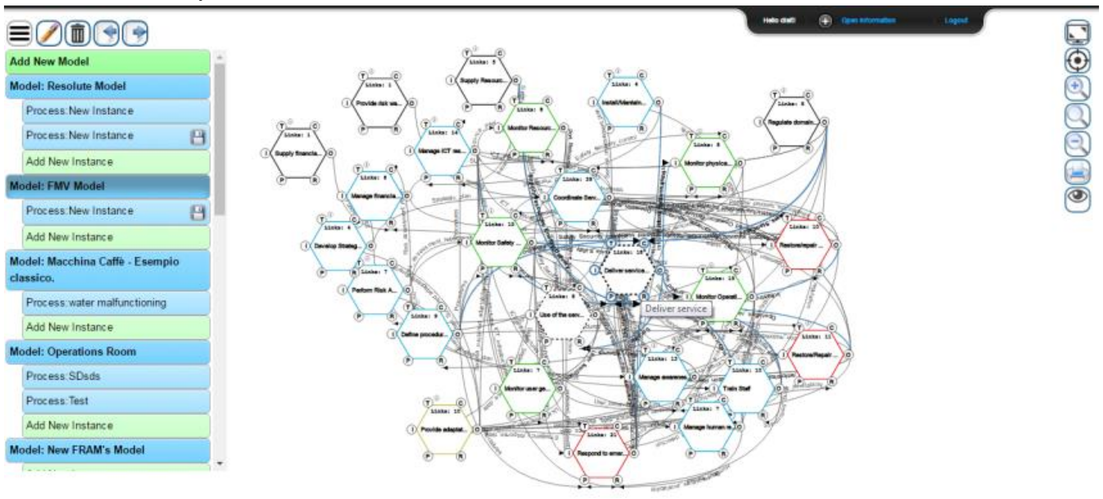


Figure 77. ERMG model. This is the model imported from FMV as it is. The function's properties are maintained; name, colour, description and also the position in the graph.

With hierarchy is possible to reduce the number of functions showed and consequently the links, as demonstrated in Figure 78.

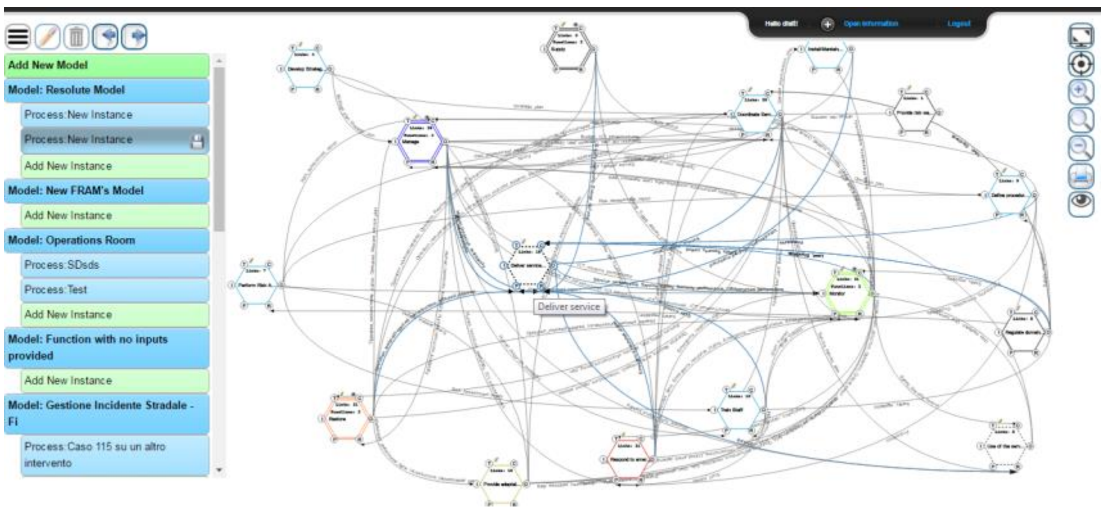
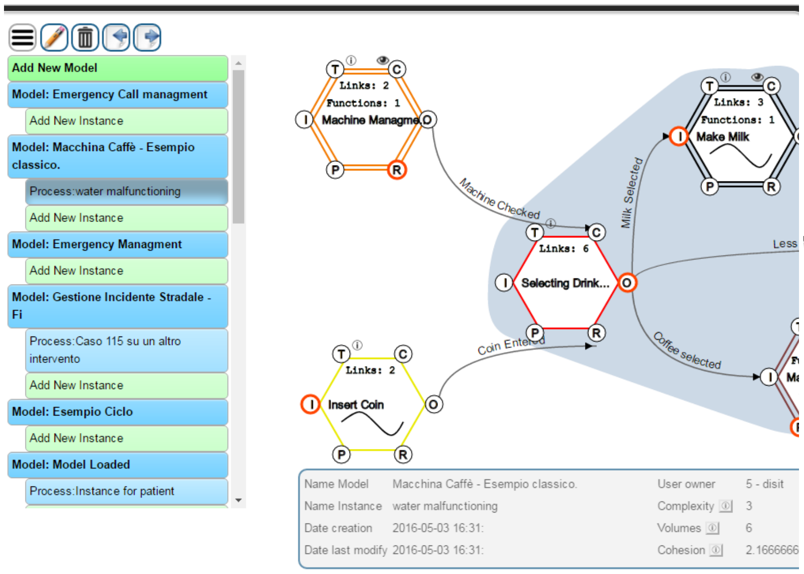


Figure 78. ERMG model with groups.

The models created are saved on the server and can be edited by the creator and shared among the users. For each Resilience DS model, it is possible to create one or more instances by providing real values and parameters. An instance is a real implementation of the model where it is possible to figure out behaviours, errors and to analyse resonance among the functions. Each function can be edited for setting the type of variability.



Name Model	Macchina Caffè - Esempio classico.	User owner	5 - disit
Name Instance	water malfunctioning	Complexity	3
Date creation	2016-05-03 16:31:	Volumes	6
Date last modify	2016-05-03 16:31:	Cohesion	2.1666666

Figure 79. How an Instance looks like in ResilienceDS. The wave on a function indicates that is effected by variability.

Figure 79 contains an instance example for a coffee machine. The instantiation refers to a problem in the water supply that affects the entire model, especially the coffee preparation.

The Resilience DS model extends the FRAM classical model. Each output process can be operationalized by a computable function based on the function's inputs. In the Resilience DS, each process is modelled by means of the SmartDS.disit.org process, which adopts an extended System Thinking Approach [26].

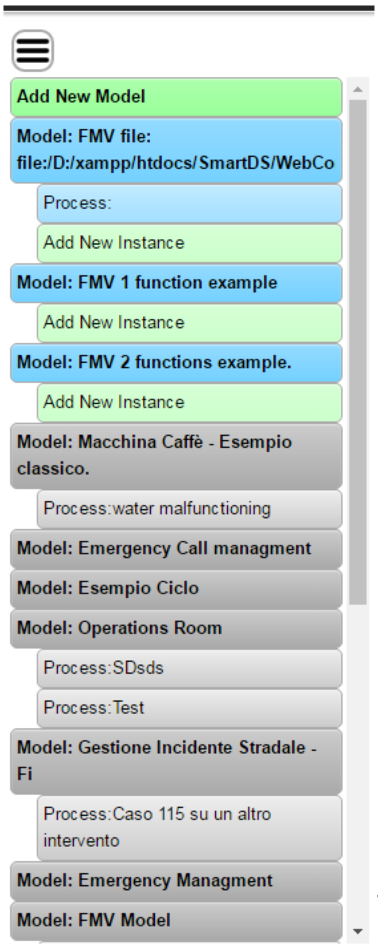

Add New Model
Model: FMV file: file:/D:/xampp/htdocs/SmartDS/WebCo
Process:
Add New Instance
Model: FMV 1 function example
Add New Instance
Model: FMV 2 functions example.
Add New Instance
Model: Macchina Caffè - Esempio classico.
Process:water malfunctioning
Model: Emergency Call managment
Model: Esempio Ciclo
Model: Operations Room
Process:SDsds
Process:Test
Model: Gestione Incidente Stradale - Fi
Process:Caso 115 su un altro intervento
Model: Emergency Managment
Model: FMV Model

Figure 80 List of models for the user logged in.

The Resilience DS provides many functions and options. When a user logs in the tool, it offers the list of model beforehand created and saved, as showed in Figure 80. In this list the instances created and the other users' projects are also present. The first element allows creating a new model.
When a model is selected, it is shown with all its actions and aspects on the main frame. At the top of the functions, there are the options for editing, deleting and, only for the group, for showing the elements within.

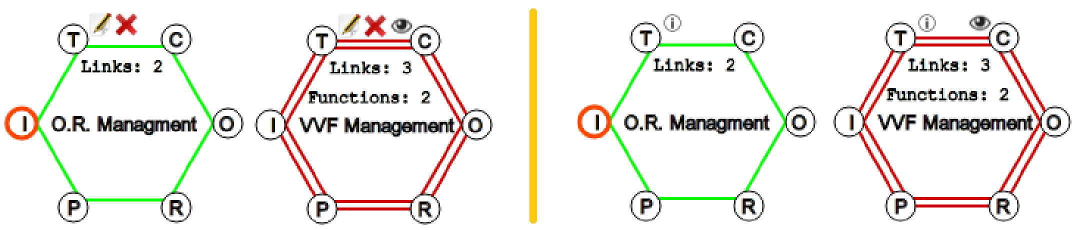
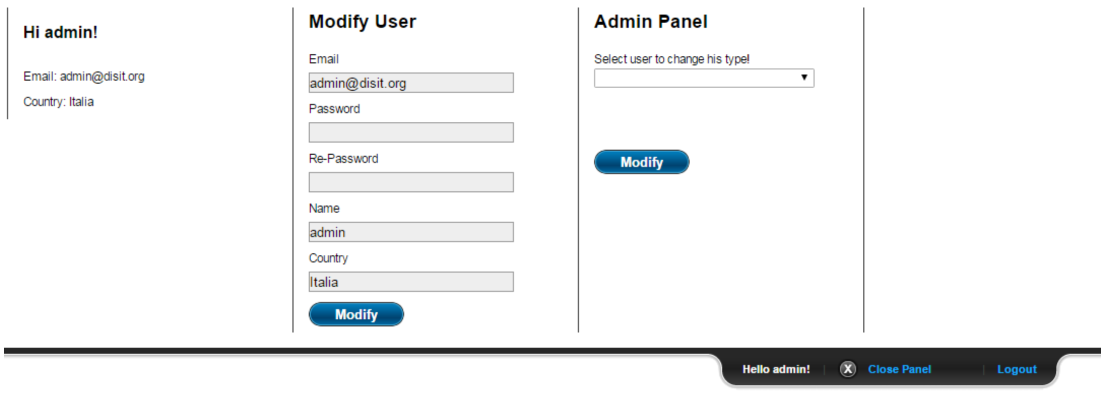


Figure 81. Options for functions and groups. For the actions we have, starting from the left: edit and delete. For the groups: edit, delete and show. On the right are reported the elements' options during the visualization: info for functions and info and show fo

There are also mouse triggers to speed up the user modelling experience, like: focus a function connection with mouse over, focus the extreme of an aspect with mouse over the label, edit functions and groups with double click and creating aspect with just the drag from the hexagon vertices.



Hi admin!

Email: admin@disit.org

Country: Italia

Modify User

Email: admin@disit.org

Password:

Re-Password:

Name: admin

Country: Italia

Modify

Admin Panel

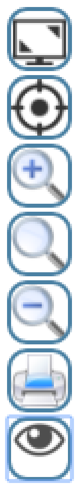
Select user to change his type!

Modify

Hello admin! Close Panel Logout

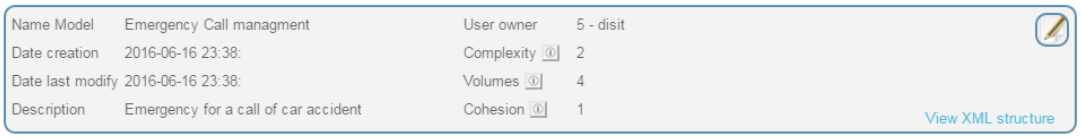
Figure 82. User's panel. Over the logout option there is the admin panel with the information about the user in the left frame. In the centre is possible to modify own credential, like email, name etc. The third section is only available for the admins and permit

The tool allows for the management of multiple users. With that is possible to register new users, login/logout with the own account, or access as a guest for view available models (see Figure 82).
The Resilience DS tool provides the options for: view in full screen the graph, centre the model at the origin, zoom in, out and reset the visualization (this is also possible scrolling the mouse's wheel), print the current model and show/hide the aspects for focus only over the functions. In Figure 83 is shown the right menu.
In the bottom menu are reported the model's information: name, description, user owner, creation and last modify date. From there is possible to edit the model info and view the XML structure (The information sent from the server to the client when the model is requested). There are also 3 indexes for the graph:



- **Volume**: is the total number of functions in the model.
- **Cohesion**: is how much the model is connected with others. It is obtained by the formula: $\boldsymbol{Ch} = \frac{Na}{Nf}$. Where: $\boldsymbol{Na}$ is the number of aspects and $\boldsymbol{Nf}$ the number of functions. In the instance reported above we have the cohesion as: $\boldsymbol{Ch} = \frac{13(different\ aspects)}{6} = 2{,}16667$
- **Complexity**: The complexity for the model is the max number of functions for a group. Exactly: $\boldsymbol{O} = \max_{i=0\dots N_g} G_i$. Where: $\boldsymbol{N_g}$ is the number of group and $\boldsymbol{G_i}$ the number of functions in the i-th group. In the example we have a MMF that contains 3 functions, the one exploded.

Figure 83 Right Menu for the visualization.



Name Model	Emergency Call managment	User owner	5 - disit
Date creation	2016-06-16 23:38:	Complexity	2
Date last modify	2016-06-16 23:38:	Volumes	4
Description	Emergency for a call of car accident	Cohesion	1

View XML structure

Figure 84. Bottom menu with all the information about a model loaded.

When a model is requested to be shown, from the top menu the following is possible: edit the model, delete the model and import a new FMV model to modify or test. While editing a model, a top menu is displayed that allows the user to undo and forward the basic operations (Precisely: add/remove/modify of functions, aspects and groups and add, remove functions in groups). Last, there is the export button, for exporting a function from the Resilience DS to SmartDS. This feature, actually provided only for the users of both the tools, shows the obtainable functions to export and then the output available as a goal. The exported graph is automatically imported into SmartDS. During modelling there are other options on this menu, useful for editing. With these, it is possible to undo, or redo some basic command like: create, modify and delete of functions, groups and aspects. These are supposedly useful for backtracking. The other two buttons allow creating new functions and MMF.

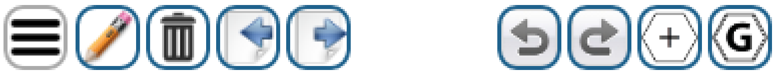


Figure 85. Menu for models management. The first group of icons, from left to right, are for: collapse the models list, edit a model/instance, delete a model/instance, import an FMV's project and export to SmartDS a function for measurement of probabilities. The second group of icons represent: undo/redo actions, add new function, add new group.

7.6.2 Analysis and comparison with other tools.

Here a comparison between Resilience DS and FMV is reported, regarding the modelling tool.

Table 10. Comparison table between main features of Resilience DS and FMV

FEATURES	Resilience DS	FMV
MODELING		
FRAM modelling	Y	Y
Functions and Aspects identified univocally	Y (1)	Y (6)
Modelling registration	N	Y
Export a Model for simulation	Y	N (7)
Provide instances for the models	Y	N
Count number of elements	Y	N
Provide graph indexes	Y (2)	N
Show missed aspect	Y (3)	Y
Hierarchy in the model	Y	N

GENERAL MANIPULATION		
Undo action	Y	N
Zooming the graph	Y	Y (8)
centring the graph	Y	N
Save and Load	Y (4)	Y (9)
Panning	Y	Y
Show/Hide the aspects	Y	N
Share	Y	N (10)
Show/Hide the functions in a group	Y	N
Full screen	Y	Y
Show aspect's label	Y	Y
Highlight selected function	Y (5)	Y
Switch to next function	N	Y
NON FUNCTIONAL		
Web tool	Y	N
Change Language	N	Y
Print the model	Y	Y
User differentiation	Y	N

1. In Resilience DS the functions and the aspects are identified by univocal ID integers,.
2. Three Indices are provided for the model Resilience DS: Volumes, Cohesion and Complexity.
3. Resilience DS like FMV highlight the entry of the missing links but also provides a label that display the aspect's name.
4. In Resilience DS the model is saved in the DB.
5. In Resilience DS is also possible to highlight the aspect just going with the mouse over the name.
6. In FMV the functions are identified by a univocal id, but the aspects with their name. This can create ambiguity during complex modelling if a name is already used.
7. In the FMV official version is not provided a way to export the model, but there is a version modified by CAMBRENSIS that export a model into iDEPEND.
8. In FMV is possible to zoom the editor only with the buttons and not with the mouse's wheel.
9. FMV save the model locally in the user's PC.
10. In FMV there is not any system included for directly sharing a newly created model . However, it is possible to share the file of the model created.

Concluding, we report the principal differences between the two tools presented in Table 11: FRAM with SimPy by Van Kleef and FRAM with iDEPEND by CAMBRENSIS [25][26], and Resilience DS with the SmartDS application [30].

Table 11. Resilience DS, Kleef (SimPy) and CAMBRENSIS (iDEPEND) differences.

	Resilience DS	**SimPy**	**iDEPEND**
Non Functional			
License	Open Source	There isn't an available tool	No Open Source
Web Based Tool (Programming language)	Y (Java + JavaScript + MySQL)	N (Python)	Y (PHP and JavaScript)

Available online	Y Best with Mozilla Firefox, Google Chrome.	N	Y Best with Mozilla Firefox, Google Chrome or later versions of Internet Explorer (version 10 or above).
People involved.	-Everyone (Experts in the field, Decision maker or even private)	-Experts in the field -Decision maker	-Everyone (Experts in the field, Decision maker or even private)
Space	< 100MB	SimPy < 500KB	?
User distinction	Y	N	Y (Based on the registration). Two typologies, one for Trial version (Limited for 2 models, and 30 entities). One commercial with no limit of models and entities plus other features.
		Modelling	
Type of simulation	- Modified version of Analytical Hierarchical Process (**AHP**)	-SimPy -Monte Carlo	-Bayesian Belief Nets (BBN) -Monte Carlo
CI system representation	Decision Tree	Block Diagram	Decision Tree
Nodes role	-The Root of the tree is the goal function's output. -The other criteria are the aspects of the functions	FRAM Functions	-The Root of the tree is the goal function's output -The other nodes are the aspects of the functions.
Branches role	Links between aspects. Each branch has a weight defined by the comparison matrix	Link between the functions	Links between aspects. Each branch has weight (By default: And, Or and custom).
Functionalities and application fields	-Decision Support -Dependency Modelling -Quantitative FRAM analysis	-Decision Support -Dependency Modelling -Qualitative FRAM analysis	-Decision Support -Dependency Modelling -Quantitative FRAM analysis
Follow Van Kleef's rules	N (1)	Y	Y
		General Model handling	
Visualization	2D	2D	2D
Locate on map the model	N	-	Y
Link external sources	Y	-	Y (With only the pay version)
Save/Load Model	Y	Y	Y
Share Model	Y	-	Y (Also follow a model)

Export model and results	Y	-	Y (JPG - XML – FRAM modified model)
Share Entity among models	N	-	Y
Group/Sort entity	Y	-	Y
FRAM Template	Y	-	Y
Move - Delete entity	Y	-	Y
Zoom and panning	Y	-	Y
Entities differentiation	Y	-	N
		Model Editing	
Visualize interrelationship	Y (2)	Y	Y
Compatibility with other tools	Y (3)	-	Y (The results can be exported in FMV)
Reports: Fail Mode, Three Sensitivity Model and	N	-	Y
Reports: Probabilities values	Y	-	Y
Manage Cycles	Y	-	-

1. The Resilience DS actually doesn't follow the Van Kleef's rules [31]. For example, with the current SmartDS architecture is not possible to have preconditions nodes that specify if a function work or not. However, it is currently possible refer to different time stamp and realize it with some effort.
2. It is possible to visualize the functions interdependencies by switching to Resilience DS. The others are not as clear.
3. The Resilience DS is compatible with SmartDS but there are other tools for improving the performances, like Km4City and DashBoard.

8 CONCLUSIONS

This deliverable presents the outcomes of the Task 5.2 that is the CRAMSS application. The main aim of the work was the implementation of a collaborative workspace, in which DSS operators of the UTS can share their outputs of or information about their work among each other, facilitating them thus in decision making. In this sense, several DSSs related to the UTS were developed as well as a user-friendly front-end interface that helps the UTS operators manage them easily. The implemented DSSs were developed so as to utilize the ESB (D4.4) for exchanging valuable information among them. Thus, the integration of all the separate DSSs was achieved. The development of the front-end interface was based on the recommendation described in D5.2 while the whole application was implemented considering the ERMG described in D3.5. Finally, the communication between the ESSMA (D5.4) and the CRAMSS was established allowing thus the direct communication of the UTS decision makers and the citizens.

The CRAMSS application supports the main objectives of the RESOLUTE project. The developed application increases the UTS resilience in every day conditions as well as during crises. Through its highly synergic approach the CRAMSS provides a mean for the exchange of valuable information among all stakeholders involved in UTS resilience management, facilitating thus the coordination of all actions and the early decision making. The CRAMSS system manages to address the following objectives of the RESOLUTE project:

- Obj.3.1: The CRAMSS system has been developed in such a way so as to be smoothly deployed in the pilot sites in Y3 covering the current infrastructure requirements and extending the future operator's capability's with beyond SoA technologies, as presented above.
- Obj 3.2: The CRAMSS system is aggregating big data sources through the "over the ESB" & "over the Data layer" data integration.
- Obj 3.3: Through the Twitter Vigilance tool, WIFI people behaviour tracking and the communication with the ESSMA, the CRAMSS is able to gather crowd sourced/sensed information (e.g. annotations, comments, news about the city, etc.) regarding the emergent events within the city.
- Obj 3.4: the system benefits of the semantic –based data integration and processing to enhance the information value managed by the system. The resilience based ontology approach organises the big data generated by the UTS and the city as a whole to exploit semantic relationships that can reveal hidden interdependencies.
- Obj 3.5: The system supports data mining and predictive modelling on human behaviour on the ground as well as analysis of data collected for the IoT sensors and traffic.
- Obj 3.6: Through the eDSS and the ESSMA the system is capable of identifying individuals or groups of individuals as voluntary rescuers or to-be-rescued, assigning the appropriate task to each taking into account multi-dimensional criteria, such as spatiotemporal context, users' profiles, etc.
- Obj 3.7: The system allows the continuous monitoring of the city and the happening events. The operators are informed through the Dashboard about new emergent events or the evolution of the already happening. Moreover the system is able to make suggestions (e.g. evacuation routes) in order to minimize the impact of the events.
- Obj 3.8: Through the communication with ESSMA the system can timely communicate with the crowd in order to inform and guide them properly about the happening events. Furthermore the CRAMSS is able to receive feedback from the citizens regarding the emergency.
- Obj 3.9: The system supports the decision making of the UTS stakeholders through its three interactive and user friendly UIs. Through the latter coherent and meaningful information are displayed to the operators.

- Obj 4: The system communicates with the game based training app (GBTA), which provides an entertaining way for the citizens to be familiar with ESSMA and the proper way of acting during emergencies. By assigning specific roles and tasks the GBTA evaluate its users helping thus to the training of the latter.

Following the outcomes of this deliverable the deployment of the CRAMSS system in the real pilot sites and its validation from real world operators have to be done.

APPENDICES

Annex I

Example Request:

```
GET --header 'Accept: application/json' 'http://192.168.55.75:8088/rest/v1/detectionunits/6f796196-5ca7-4546-ad84-
17ae79224e4b/measures?token=1234&fromDate=2016-06-07T15:00:00Z&toDate=2016-06-09T16:00:00Z'
```

Example Response:

```
RESPONSE CODE: 200
RESPONSE HEADER:
{
  "access-control-allow-origin": "*",
  "date": "Wed, 08 Jun 2016 16:49:02 GMT",
  "server": "Microsoft-HTTPAPI/2.0",
  "transfer-encoding": "chunked",
  "content-type": "application/json; charset=utf-8"
}
RESPONSE BODY:
{
  "Measures": [
    {
      "MeasureTimeUTC": "2016-06-08T15:45:00Z",
      "Sensors": [
        {
          "Data": [
            {
              "Group": null,
              "Key": "VOLUME",
              "Value": "20"
            },
            {
              "Group": null,
              "Key": "SPEED",
              "Value": "50"
            },
            {
              "Group": null,
              "Key": "OCCUPANCY",
              "Value": "0"
            },
            {
              "Group": null,
              "Key": "ACCURACY",
              "Value": "0"
            },
            {
              "Group": null,
              "Key": "GAP",
              "Value": "-1"
            },
            {
              "Group": null,
              "Key": "HEADWAY",
              "Value": "-1"
            },
            {
              "Group": null,
              "Key": "LENGTH",
              "Value": "-1"
            }
          ]
        },
        {
          "Data": [
            {
              "Group": null,
              "Key": "VOLUME",
              "Value": "30"
            },
            {
              "Group": null,
              "Key": "SPEED",
              "Value": "60"
            },
            {
              "Group": null,
              "Key": "OCCUPANCY",
              "Value": "0"
            },
            {
              "Group": null,
              "Key": "ACCURACY",
              "Value": "0"
            },
            {
              "Group": null,
              "Key": "GAP",
              "Value": "-1"
            },
            {
              "Group": null,
              "Key": "HEADWAY",
              "Value": "-1"
            },
            {
              "Group": null,
              "Key": "LENGTH",
              "Value": "-1"
            }
          ]
        }
      ]
    },
    {
      "MeasureTimeUTC": "2016-06-08T15:50:00Z",
      "Sensors": [
        {
          "Data": [
            {
              "Group": null,
              "Key": "VOLUME",
              "Value": "20"
            },
            {
              "Group": null,
              "Key": "SPEED",
              "Value": "50"
            },
            {
              "Group": null,
              "Key": "OCCUPANCY",
              "Value": "0"
            },
            {
              "Group": null,
              "Key": "ACCURACY",
              "Value": "0"
            },
          {
            "Group": null,
            "Key": "GAP",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "HEADWAY",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "LENGTH",
            "Value": "-1"
          }
        ]
      },
      {
        "Data": [
          {
            "Group": null,
            "Key": "VOLUME",
            "Value": "30"
          },
          {
            "Group": null,
            "Key": "SPEED",
            "Value": "60"
          },
          {
            "Group": null,
            "Key": "OCCUPANCY",
            "Value": "0"
          },
          {
            "Group": null,
            "Key": "ACCURACY",
            "Value": "0"
          },
          {
            "Group": null,
            "Key": "GAP",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "HEADWAY",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "LENGTH",
            "Value": "-1"
          }
        ]
      },
      {
        "Data": [
          {
            "Group": null,
            "Key": "VOLUME",
            "Value": "20"
          },
          {
            "Group": null,
            "Key": "SPEED",
            "Value": "50"
          },
          {
            "Group": null,
            "Key": "OCCUPANCY",
            "Value": "0"
          },
          {
            "Group": null,
            "Key": "ACCURACY",
            "Value": "0"
          },
          {
            "Group": null,
            "Key": "GAP",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "HEADWAY",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "LENGTH",
            "Value": "-1"
          }
        ]
      },
      {
        "Data": [
          {
            "Group": null,
            "Key": "VOLUME",
            "Value": "30"
          },
          {
            "Group": null,
            "Key": "SPEED",
            "Value": "60"
          },
          {
            "Group": null,
            "Key": "OCCUPANCY",
            "Value": "0"
          },
          {
            "Group": null,
            "Key": "ACCURACY",
            "Value": "0"
          },
          {
            "Group": null,
            "Key": "GAP",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "HEADWAY",
            "Value": "-1"
          },
          {
            "Group": null,
            "Key": "LENGTH",
            "Value": "-1"
          }
        ]
      }
```

```
   ]
  }
 ]
}
```

Annex II

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "timestamp": {
      "type": "integer"
    },
    "positions": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "ap": {
            "type": "string"
          },
          "lat": {
            "type": "string"
          },
          "lng": {
            "type": "string"
          },
          "id": {
            "type": "string"
          },
          "train": {
            "type": "string"
          },
          "dir": {
            "type": "string"
    }

        },

        "required": [

          "ap",

          "lat",

          "lng",

          "id"

        ]

      }

    }

  },

  "required": [

    "timestamp",

    "positions"

  ]

}
```

Annex III

Example Strategies:

```
<strategies>
        <timestamp>2016-02-13T17:51:00</timestamp>
        <strategy>
                <description>test</description>
                <priority>8</priority>
                <activationtype>automatic</activationtype>
                <activationrules>
                        <activationthreshold>50</activationthreshold>
                        <daysofweek>1234567<daysofweek>
                        <timesofday>
                                <timeofday>
                                        <starttime>07:00</starttime>
                                        <endtime>19:00</endtime>
                                </timeofday>
                        </timesofday>
                        <dates>
                                <date>
                                        <startdate>2017-01-01</startdate>
                                        <enddate>2017-31-12</enddate>
                                </date>
                        </dates>
                </activationrules>
                <currentmembership>47</currentmembership>
                <controlpoints>
                        <controlpoint>
                                <objecttype>DETECTOR</objecttype>
                                <description>Via Dante</description>
                                <measuretype>Volume</measuretype>
                                <functiontype>RAMP-UP</functiontype>
```

```
                                        <threshold1>1700</threshold1>
                                        <threshold2>2000</threshold2>
                                        <priority>3</priority>
                                        <observed>TRUE</observed>
                                        <currentmemebership>47</currentmemebership>
                                </controlpoint>
                        </controlpoints>
                        <actionpoints>
                                <actionpoint>
                                        <objecttype>VARIABLE_MESSAGE_SIGN</objecttype>
                                        <description>Via Dante</description>
                                        <command>MESSAGE</command>
                                        <commandparameters>
                                                <commandparameter>
                                                        <key>MESSAGE</key>
                                                        <value>Test message</value>
                                                </commandparameter>
                                        </commandparameters>
                                </actionpoint>
                        </actionpoints>
                </strategy>
</strategies>
```

Example Events:

```
<a:MisticEventDto>
  <a:arcId xmlns:b="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
        <b:int>3770</b:int>
  </a:arcId>
  <a:archived>1</a:archived>
  <a:bearing i:nil="true"/>
  <a:creatorUserId i:nil="true"/>
  <a:datex2Status>1</a:datex2Status>
  <a:directionName i:nil="true"/>
  <a:eventCode>EVT:20161220142145065</a:eventCode>
  <a:eventInstanceId>4082</a:eventInstanceId>
  <a:eventSubtypeId>9</a:eventSubtypeId>
  <a:eventTypeId>1</a:eventTypeId>
  <a:locationType>2</a:locationType>
  <a:mpkCountryCode i:nil="true"/>
  <a:mpkGeocode i:nil="true"/>
  <a:mpkHouseNumber i:nil="true"/>
  <a:mpkPlace i:nil="true"/>
  <a:mpkPostCode i:nil="true"/>
  <a:mpkRegion i:nil="true"/>
  <a:mpkStreetName i:nil="true"/>
  <a:notes i:nil="true"/>
  <a:placeName/>
  <a:predicted>0</a:predicted>
  <a:priority>0</a:priority>
  <a:roadCode i:nil="true"/>
  <a:roadName i:nil="true"/>
  <a:secondPlaceName i:nil="true"/>
  <a:severity>10</a:severity>
  <a:simulated>0</a:simulated>
  <a:situationId i:nil="true"/>
  <a:source>Da operatore</a:source>
  <a:srs>epsg:4326</a:srs>
  <a:startTime>2016-12-20T14:21:00</a:startTime>
  <a:status>C</a:status>
  <a:stopTime i:nil="true"/>
  <a:streetName>E45</a:streetName>
  <a:supplierCode i:nil="true"/>
  <a:tmcCid i:nil="true"/>
  <a:tmcDirection i:nil="true"/>
  <a:tmcPriDistance i:nil="true"/>
  <a:tmcPriPointLcd i:nil="true"/>
  <a:tmcRoadLcd i:nil="true"/>
  <a:tmcSecDistance i:nil="true"/>
  <a:tmcSecPointLcd i:nil="true"/>
  <a:tmcTabCd i:nil="true"/>
  <a:updateTime>2016-12-20T14:21:45.25</a:updateTime>
  <a:valid>1</a:valid>
  <a:validFrom>2016-12-20T14:21:45.25</a:validFrom>
  <a:validTo>2016-12-27T09:04:36.187</a:validTo>
  <a:validatorUserId i:nil="true"/>
  <a:visible>1</a:visible>
  <a:x>11.9788161278</a:x>
  <a:y>57.7155278043</a:y>
</a:MisticEventDto>
```

Annex IV

```xml
<?xml version="1.0" standalone="yes"?>
<FM Version="0,0,2,0">

 <Functions>
  <Function fnStyle="0" Tp="-1" Pp="-1" x="109.5" y="120" style="custom" color="12574559">
   <IDNr>0</IDNr>
   <FunctionType>0</FunctionType>
   <IDName>A</IDName>
   <Description>Description A</Description>
  </Function>
  <Function fnStyle="0" Tp="1" Pp="0" x="328" y="233" fnType="2" style="blue" color="108251">
   <IDNr>1</IDNr>
   <FunctionType>0</FunctionType>
   <IDName>B</IDName>
   <Description>Description B</Description>
  </Function>
  <Function fnStyle="0" Tp="-1" Pp="-1" x="540" y="233">
   <IDNr>2</IDNr>
   <FunctionType>2</FunctionType>
   <IDName>D</IDName>
   <Description>Description C</Description>
  </Function>
  <Function fnStyle="0" Tp="-1" Pp="-1" x="113.5499389648437" y="323.0499923706055">
   <IDNr>3</IDNr>
   <FunctionType>2</FunctionType>
   <IDName>C</IDName>
   <Description>Description C</Description>
  </Function>
 </Functions>

 <Aspects>
  <Aspect x="0.040" y="0.262" directionX="from" directionY="from">
   <Name>1|Bout|2</Name>
  </Aspect>
 </Aspects>

 <Controls>
  <Control>
   <IDNr>1</IDNr>
   <IDName>Aout 2</IDName>
   <FunctionIDNr>1</FunctionIDNr>
  </Control>
 </Controls>

 <Inputs>
  <Input>
   <IDNr>1</IDNr>
   <IDName>Ain</IDName>
   <FunctionIDNr>0</FunctionIDNr>
  </Input>
  <Input>
   <IDNr>2</IDNr>
   <IDName>Aout 1</IDName>
   <FunctionIDNr>1</FunctionIDNr>
  </Input>
  <Input>
   <IDNr>3</IDNr>
   <IDName>Bout</IDName>
   <FunctionIDNr>2</FunctionIDNr>
  </Input>
  <Input>
   <IDNr>4</IDNr>
   <IDName>Cout 3</IDName>
   <FunctionIDNr>1</FunctionIDNr>
  </Input>
 </Inputs>

 <Outputs>
  <Output>
   <IDNr>1</IDNr>
   <IDName>Aout 1</IDName>
   <FunctionIDNr>0</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>2</IDNr>
    <IDName>Bout</IDName>
    <FunctionIDNr>1</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>3</IDNr>
    <IDName>Cout 1</IDName>
    <FunctionIDNr>3</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>4</IDNr>
    <IDName>Cout 2</IDName>
    <FunctionIDNr>3</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>5</IDNr>
    <IDName>Cout 3</IDName>
    <FunctionIDNr>3</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>6</IDNr>
    <IDName>Aout 2</IDName>
    <FunctionIDNr>0</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>7</IDNr>
    <IDName>Aout 3</IDName>
    <FunctionIDNr>0</FunctionIDNr>
  </Output>
  <Output>
    <IDNr>8</IDNr>
    <IDName>Cout 4</IDName>
    <FunctionIDNr>3</FunctionIDNr>
  </Output>
 </Outputs>

 <Preconditions>
  <Precondition>
    <IDNr>1</IDNr>
    <IDName>Cout 1</IDName>
    <FunctionIDNr>1</FunctionIDNr>
  </Precondition>
 </Preconditions>

 <Resources>
  <Resource>
    <IDNr>1</IDNr>
    <IDName>Cout 2</IDName>
    <FunctionIDNr>1</FunctionIDNr>
  </Resource>
  <Resource>
    <IDNr>2</IDNr>
    <IDName>Cout 4</IDName>
    <FunctionIDNr>0</FunctionIDNr>
  </Resource>
 </Resources>

 <Times>
  <Time>
    <IDNr>1</IDNr>
    <IDName>Aout 3</IDName>
    <FunctionIDNr>1</FunctionIDNr>
  </Time>
 </Times>

</FM>
```

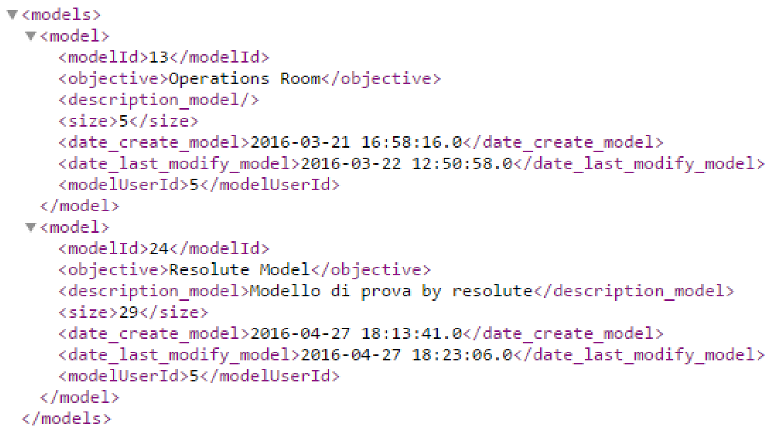
Annex V

REST Calls for models.

Request for obtain the list of models saved in the DB (REST request send to the Models resource).

GET rest / models /

The result of this call is reported below but it is also available entering in the browser the URI: http://smartds.disit.org:8080/fram/rest/models.



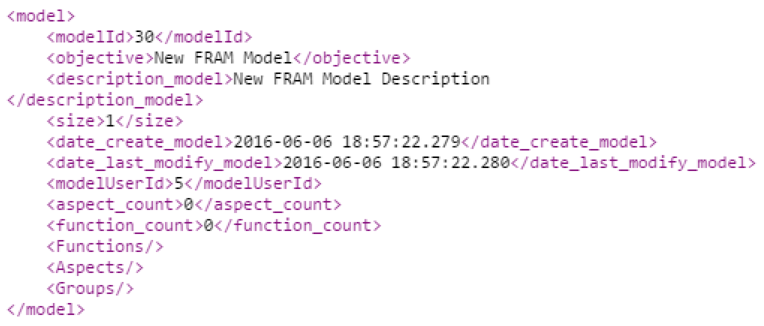
```
▼<models>
  ▼<model>
     <modelId>13</modelId>
     <objective>Operations Room</objective>
     <description_model/>
     <size>5</size>
     <date_create_model>2016-03-21 16:58:16.0</date_create_model>
     <date_last_modify_model>2016-03-22 12:50:58.0</date_last_modify_model>
     <modelUserId>5</modelUserId>
   </model>
  ▼<model>
     <modelId>24</modelId>
     <objective>Resolute Model</objective>
     <description_model>Modello di prova by resolute</description_model>
     <size>29</size>
     <date_create_model>2016-04-27 18:13:41.0</date_create_model>
     <date_last_modify_model>2016-04-27 18:23:06.0</date_last_modify_model>
     <modelUserId>5</modelUserId>
   </model>
 </models>
```

Figure 86. List of models present in the DB.

REST call for create a new model on server (Request send to the Models resource). The model created is not saved until the user's click on the save button.

POST rest / models /

The server's response is a XML with the data for the new model.



```
<model>
    <modelId>30</modelId>
    <objective>New FRAM Model</objective>
    <description_model>New FRAM Model Description
</description_model>
    <size>1</size>
    <date_create_model>2016-06-06 18:57:22.279</date_create_model>
    <date_last_modify_model>2016-06-06 18:57:22.280</date_last_modify_model>
    <modelUserId>5</modelUserId>
    <aspect_count>0</aspect_count>
    <function_count>0</function_count>
    <Functions/>
    <Aspects/>
    <Groups/>
</model>
```

Figure 87. Response to a model creation.

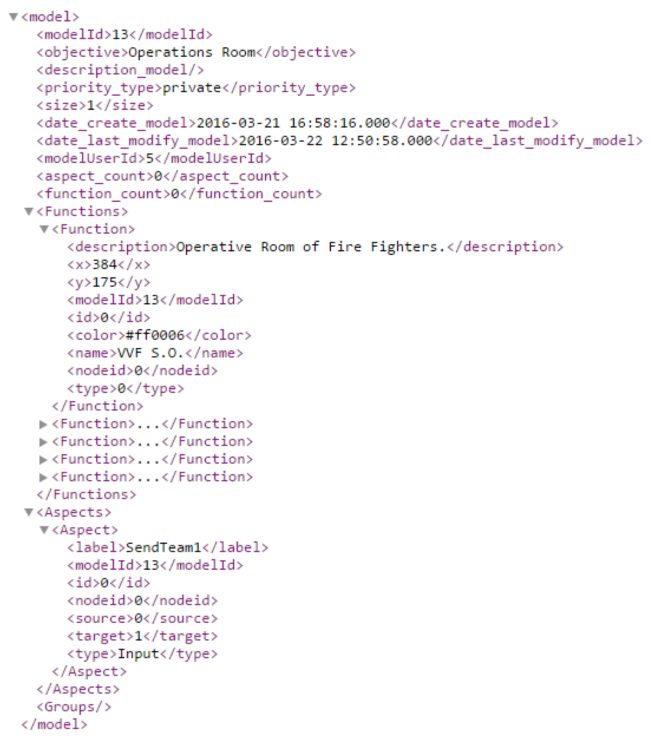
Call for modifying the information (name and description) of an existent model of the currently logged-in user. The resource that manages this operation is Models.

PUT rest / models / {model_id}

The next request is for get a specific model of the user. The model is identified by its id ({model_id}). Models is the resource that do this on the server side.

GET rest / models / {model_id}

Below there is the result of this request. It is possible to view this XML by entering in the browser the URI: http://smartds.disit.org:8080/fram/rest/models/{model_id} , with in place of {model_id} the identifier of an own model.



```
▼<model>
   <modelId>13</modelId>
   <objective>Operations Room</objective>
   <description_model/>
   <priority_type>private</priority_type>
   <size>1</size>
   <date_create_model>2016-03-21 16:58:16.000</date_create_model>
   <date_last_modify_model>2016-03-22 12:50:58.000</date_last_modify_model>
   <modelUserId>5</modelUserId>
   <aspect_count>0</aspect_count>
   <function_count>0</function_count>
 ▼<Functions>
   ▼<Function>
       <description>Operative Room of Fire Fighters.</description>
       <x>384</x>
       <y>175</y>
       <modelId>13</modelId>
       <id>0</id>
       <color>#ff0006</color>
       <name>VVF S.O.</name>
       <nodeid>0</nodeid>
       <type>0</type>
     </Function>
   ▶<Function>...</Function>
   ▶<Function>...</Function>
   ▶<Function>...</Function>
   ▶<Function>...</Function>
   </Functions>
 ▼<Aspects>
   ▼<Aspect>
       <label>SendTeam1</label>
       <modelId>13</modelId>
       <id>0</id>
       <nodeid>0</nodeid>
       <source>0</source>
       <target>1</target>
       <type>Input</type>
     </Aspect>
   </Aspects>
   <Groups/>
 </model>
```

Figure 88. Model returned from the server to the client.

REST call for save a model on the server. The resource that handles this request is Model.

POST rest / modeloperations /

REST call for delete a model on the server. This request is also managed by the class Model.

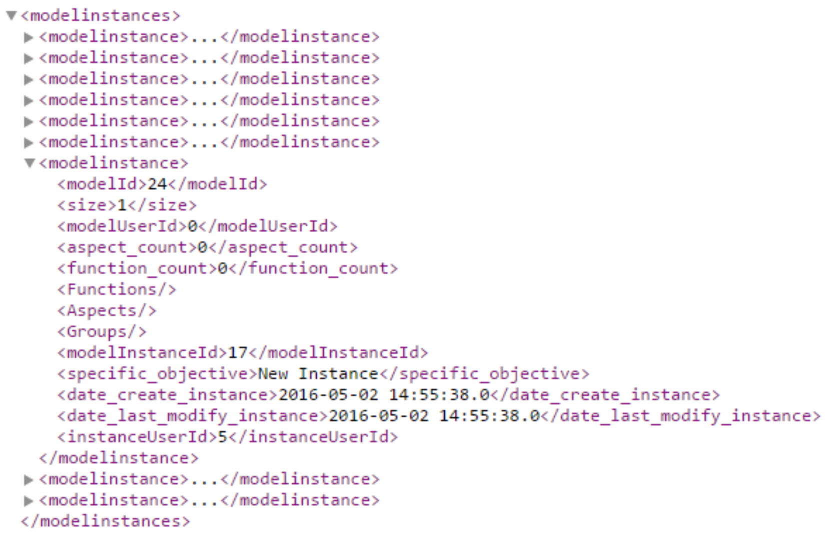
DELETE rest / models / {models_id}

REST's calls for operations over the instances.

The resource ModelInstances manages the request to retrieve the list of instances.

GET rest / modelinstances /

The result is reported below and it is accessible on the browser entering the URI: http://smartds.disit.org:8080/fram/rest/modelinstances.



```
▼<modelinstances>
  ▶<modelinstance>...</modelinstance>
  ▶<modelinstance>...</modelinstance>
  ▶<modelinstance>...</modelinstance>
  ▶<modelinstance>...</modelinstance>
  ▶<modelinstance>...</modelinstance>
  ▶<modelinstance>...</modelinstance>
  ▼<modelinstance>
     <modelId>24</modelId>
     <size>1</size>
     <modelUserId>0</modelUserId>
     <aspect_count>0</aspect_count>
     <function_count>0</function_count>
     <Functions/>
     <Aspects/>
     <Groups/>
     <modelInstanceId>17</modelInstanceId>
     <specific_objective>New Instance</specific_objective>
     <date_create_instance>2016-05-02 14:55:38.0</date_create_instance>
     <date_last_modify_instance>2016-05-02 14:55:38.0</date_last_modify_instance>
     <instanceUserId>5</instanceUserId>
   </modelinstance>
  ▶<modelinstance>...</modelinstance>
  ▶<modelinstance>...</modelinstance>
 </modelinstances>
```

Figure 89. List of the instances created.

Below the REST call for an instance creation. The resource that manage this request is ModelInstances and it creates a new instance on the current section but not in the DB.

POST rest / modelinstances /

REST call for modify the instance's information (The only available at the moment is the Name). Handled by ModelInstances.

PUT rest / modelinstances / {model_instance_id}

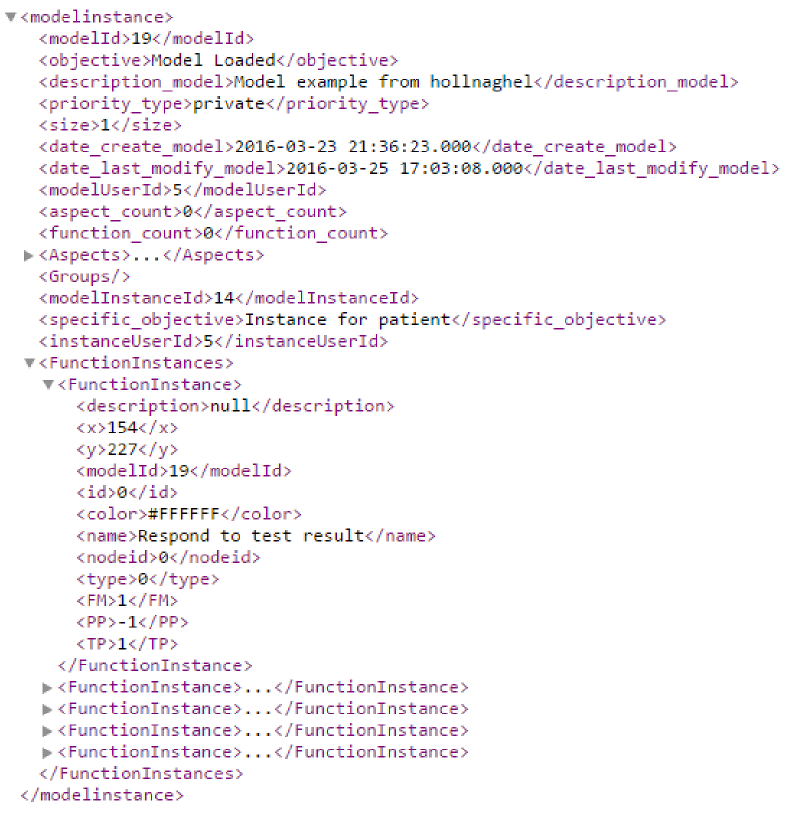
For delete an instance the call is similar to the previous and is managed by ModelInstances.

DELETE rest / modelinstance / {model_instance_id}

The class ModelInstances manage REST's call for get a specific instance.

GET rest / modelinstances / {model_instance_id}

Is possible to view the result below directly on the browser, with the URI: http://smartds.disit.org:8080/fram/rest/modelinstances/{model_instance_id} for an own instance.



```
▼<modelinstance>
   <modelId>19</modelId>
   <objective>Model Loaded</objective>
   <description_model>Model example from hollnaghel</description_model>
   <priority_type>private</priority_type>
   <size>1</size>
   <date_create_model>2016-03-23 21:36:23.000</date_create_model>
   <date_last_modify_model>2016-03-25 17:03:08.000</date_last_modify_model>
   <modelUserId>5</modelUserId>
   <aspect_count>0</aspect_count>
   <function_count>0</function_count>
 ▶<Aspects>...</Aspects>
   <Groups/>
   <modelInstanceId>14</modelInstanceId>
   <specific_objective>Instance for patient</specific_objective>
   <instanceUserId>5</instanceUserId>
 ▼<FunctionInstances>
   ▼<FunctionInstance>
       <description>null</description>
       <x>154</x>
       <y>227</y>
       <modelId>19</modelId>
       <id>0</id>
       <color>#FFFFFF</color>
       <name>Respond to test result</name>
       <nodeid>0</nodeid>
       <type>0</type>
       <FM>1</FM>
       <PP>-1</PP>
       <TP>1</TP>
     </FunctionInstance>
   ▶<FunctionInstance>...</FunctionInstance>
   ▶<FunctionInstance>...</FunctionInstance>
   ▶<FunctionInstance>...</FunctionInstance>
   ▶<FunctionInstance>...</FunctionInstance>
   </FunctionInstances>
 </modelinstance>
```

Figure 90. Response for a specific model instance.

Next, the REST calls for saving an instance. The ModelInstances resource manages the request.

POST rest / modelinstanceoperations /

For exporting a model, the REST's request is handled by the ExportFRAMModelResource resource. The call is:

POST rest / exportFRAMmodel / {model_id}?OutputNodeId={output_id}

REFERENCES

1. Braun, A.; Musse, S.R.; de Oliveira, L.P.L.; Bodmann, B.E.J. (2003). Modeling individual behaviors in crowd simulation. In: *16th International Conference on Computer Animation and Social Agents*, 2003, 143- 148.
2. Hamacher, H., Tjandra, S. (2002). Mathematical modelling of Evacuation Problems: A State of the Art. *Pedestrian and Evacuation Dynamics*, 227-266.
3. Rathi, A.K; Solanki, R.S. (1993). Simulation of Traffic Flow during Emergency Evacuations: a Microcomputer Based Modeling System.In: *Simulation Conference Proceedings*, (1993), 1250-1258.
4. Borrmann, A., Kneidl, A., Koester, G., Ruzika, S., Thiemann, M. (2012). Bidirectional Coupling of Macroscopic and Macroscopic Pedestrian Evacuation Models. *Safety Science*, 50(8), 1695-1703.
5. Reuben, B, Goldblatt, and Kevin Weinisch, Evacuation Planning, Human Factors, and Traffic Engineering,In: *Transportation Research News*, Issue 238, p.p. 13-17, May-June 2005.
6. Sheffi, Y., Mahmassani, L., Powell, W. (1982). A Transportaion Network Evacuation Model. *Transportation Research Part A: General*, 16(3), 209-218
7. Manley, M. (2012). EXITUS: An Agent-Based Evacuation Simulation Model for Heteroneous Populations. *Ph.D. Thesis, Utah State University.* (2012)
8. Aedo, I., Yu, S., Díaz, P., Acuña, P., Onorati, T. (2012). Personalized Alert Notifications and Evacuation Routes in Indoor Environments. *Sensors (Basel)*, 12(6), 7804–7827.
9. Duan, P., Zhang, H., Xiong, S., Zhou, S., Chen, Z., Yang, P. (2015). Personalized Route Planning System Based on Wardrop Equilibrium Model for Campus Evacuation. In: *2nd International Symposium on Dependable Computing and Internet of Things (DCIT)*,2015, 101-105.
10. Tsekourakis, I., Orlis, C., Ioannidis, D., Tzovaras, D., (2012). A Decision Support System for Real-Time Evacuation Management and Rescue Team Planning during Hazardous Events in Public Infrastructures, Mikulski J. (eds) Telematics in the Transport Environment. TST 2012. Communications in Computer and Information Science, 329, Springer, Berlin, Heidelber
11. J.L. Kennington and R.V. Helgasson, *Algorithms for Networks Programming*. John Wiley & Sons, 1980.
12. D. Bertsekas and P. Tseng, “Relax-IV: A Faster Version of the Relax Code for Solving Minimum Cost Flow Problems,”Technical Report P-2276, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Gambridge, Mass., 1994.
13. A.V. Goldberg, “An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm ”J. Algorithms, vol 22, pp. 1-29, 1997.
14. Lu, Q., George, B., and Shekhar, S, Capacity Constrained Routing Algorithms for Evacuations Planning: A summary of Results, SSTD’ 2005, pp. 291-307, 2005.
15. Sangho, K., George, B., and Shekhar, S. Evacuation Route Planning: Scalable Heuristics, ACMGIS ’07, Seattle, WA, 2007.
16. Hope, B., and Tardos, E., Polynomial time algorithms for some evacuation problems, *Society for Industrial and Applied Mathematics*, 1994.
17. Long Shi, Qiyuan Xie, Xudong Cheng, Long Chen, Yong Zhou, Ruifang Zhang, Developing a database for emergency evacuation model, *Building and Environment*, Volume 44, Issue 8, Pages 1724-1729, August 2009.
18. “SAVE ME Crowd Simulation,” Deliverable A5.1, 7th Framework Programme SST.2008.4.1.2. , Contract N.234027.
19. Angular JS: https://angularjs.org/
20. Bootstrap: http://getbootstrap.com/
21. WebSockets: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
22. C++ REST SDK: https://github.com/Microsoft/cpprestsdk
23. Leaflet: http://leafletjs.com/

24. Haversine formula: https://en.wikipedia.org/wiki/Haversine_formula
25. *iDEPEND EU - Dependency Modelling tool*. (n.d.). Retrieved 12 12, 2015, from http://idependeu.herokuapp.com/
26. CAMBRENSIS. (n.d.). *Systemic Interdependency Modelling*. Retrieved from http://www.cambrensis.org/wp-content/uploads/2012/05/Systemic-Interdependency-Modelling-GENSIM-0.1-docx.pdf
27. Bartolozzi, M., Bellini, P., Nesi, P., Pantaleo, G., & Santi, L. (2015). A Smart Decision Support System for Smart City. *Proc. of the 2015 IEEE International Conference on Smart City, SocialCom, SustainCom, DataCom 2015 and SC2 2015.* Chengdu, Sichuan, China
28. *Hill, R. (n.d.). Fram Model Visualizer FMV. Retrieved from the Functional Resonance Analysis Method.: http://functionalresonance.com/FMV/index.html*
29. Bellini, E., Gaitainidou, E., & Fereira, P. (2016). *European Resilience Managment Guidelines* – RESOLUTE D3.5 deliverable
30. E. Bellini, P. Nesi, G. Pantaleo and A. Venturi, "Funtional Resonance Analysis Method based-Decision Support tool for Urban Transport System Resilience Management", in Proc. of the 2nd IEEE International Smart Cities Conference (ISC2) 2016, Trento (Italy), 12-15 September 2016.
31. Van Kleef, V., & Stop, J. (2014). Reliable, resilient: Towards a dialectic synthesis. *46 th ESReDA Conference.* Torino, Italy.