Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem

D5.3 mF2C solution demonstration and field trial results (validation IT-1)

Project Number	**730929**
Start Date	**01/01/2017**
Duration	**36 months**
Topic	**ICT-06-2016 - Cloud Computing**

Work Package	WP5, mF2C - PoC Integration and Demonstration Strategy
Due Date:	M18
Submission Date:	25/06/18
Version:	1.9
Status	Draft
Author(s):	Andrea Bartoli, Denis Guilhot, Alejandro Lampropulos, Breogán Costa (WOS), Cristovao Cordeiro (SixSq), Matic Cankar, Sašo Stanovik (XLAB), Antonio Salis, Roberto Bulla (ENG), Xavi Masip, Eva Marín (UPC)
Reviewer(s)	Admela Jukan (TUBS) Ana María Juan Ferrer (ATOS)

Keywords
PoC, integration, demonstration, testbeds, iteration 1 results

Project co-funded by the European Commission within the H2020 Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

This document is issues within the frame and for the purpose of the mF2C project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	*15/05/2018*	*First draft circulated.*	*Breogán Costa, Andrea Bartoli (WOS)*
0.2	*22/05/2018*	*Second draft circulated.*	*Andrea Bartoli (WOS), Eva Marin, Xavi Masip (UPC)*
0.3	*08/06/2018*	*Contributions regarding the IT-1 components' integration.*	*Cristovao Cordeiro (SIXSQ)*
0.4	*08/06/2018*	*Contributions regarding Use Case 3.*	*Antonio Salis, Roberto Bulla (ENG)*
0.5	*07/06/2018*	*Inputs XLAB.*	*Matic Cankar (XLAB)*
0.6	*12/06/2018*	*Inputs UPC.*	*Eva Marin, Xavi Masip (UPC)*
1.0	*14/06/2018*	*Consolidated version.*	*Andrea Bartoli, Denis Guilhot, Alejandro Lampropulos (WOS)*
1.1	*17/06/2018*	*Internal review (QA1).*	*Ana Maria Juan Ferrer (ATOS)*
1.2	*18/06/2018*	*Review and integrations.*	*Antonio Salis, Roberto Bulla (ENG)*
1.3	*20/06/2018*	*Corrections.*	*Sašo Stanovnik (XLAB)*
1.4	*21/06/2018*	*Corrections after QA1.*	*Denis Guilhot, Alejandro Lampropoulos (WOS)*
1.5	*21/06/2018*	*Corrections after QA1.*	*Eva Marín, Xavi Masip (UPC)*
1.6	*22/06/2018*	*Corrections and consolidation after QA1.*	*Denis Guilhot (WOS)*
1.7	*24/06/2018*	*Internal review (QA2).*	*Ana Maria Juan Ferrer (ATOS), Admela Jukan (TUBS)*
1.8	*25/06/2018*	*Corrections after QA2.*	*Denis Guilhot (WOS), Matic Cankar (XLAB)*
1.9	*25/06/2018*	*Final review and Quality Check*	*Lara López (ATOS)*

Table of contents

Version History 3
List of figures 6
List of tables 7
Executive Summary 8
1. Introduction 9
1.1 Purpose 9
1.2 Structure of the document 9
1.3 Glossary of Acronyms 9
2. PoC and Application description 11
2.1 mF2C PoC implementation for IT-1 11
2.2 Emergency Situation Management in Smart Cities (ESM) 12
2.3 Smart Boat Service (SBS) 15
2.4 Smart Fog-Hub Service (SFHS) 16
3. Demonstration strategy for IT-1 20
3.1. PoC deployment strategy 20
3.2 Pilot deployment 20
3.3 Tested functionalities and limitations for IT-1 21
4. Use Case #1: Emergency Situation Management in Smart Cities (ESM) 25
4.1. Complete Architecture Use Case 25
4.1.1. Current Hardware and base Software for IT-1 27
4.1.2. Current Software Responsibilities 30
4.1.3. Tasks to be executed by the mF2C Agent 31
4.2. First Iteration Use Case (IT-1) 31
4.2.1 Current implementation of the IT-1 components 31
4.3. Data Flow Diagrams 33
4.3.1. IoT sensors to Monitoring Software (no mF2C) 33
4.3.2. IoT sensors to Monitoring Software (with mF2C) 35
4.3.3. Emergency Resolution Service 38
4.4. Experimental set-up 39
4.5. Results 40
4.6. KPI measurement 40
4.7. Business Prospective 42
5. Use Case #2: Smart Boat Service (SBS) 43
5.1. Complete Architecture Use Case 43
5.1.1. Current Hardware 44

5.1.2. Current software responsibilities 47
5.1.3. Tasks to be executed by the mF2C agent 48
5.2. First Iteration Use Case (IT-1) 49
5.3. Data Flow Diagrams 50
5.4. Experimentations Set Up 53
5.5. Results 54
5.6. Business Prospective 55
6. Use Case #3: Smart Fog-Hub Service (SFHS) 56
6.1. Complete Architecture Use Case 56
6.2. First Iteration Use Case (IT-1) 58
6.2.1. Current Hardware and base Software for IT-1 60
6.2.2. Current Software Responsibilities 62
6.2.3. Use Case Services 63
6.2.4. Tasks to be executed by the mF2C Agent 63
6.3. Data Flow Diagrams 64
6.3.1. App setup, privacy terms, topics selection 64
6.3.2. Object monitoring and tracking with proximity notification 66
6.3.3. Airport events notification 67
6.3.4. Behaviour analysis and forecasting 68
6.3.5. Dashboard 69
6.4. Experimentations Set Up 69
6.5. Results 70
6.6. Business Prospective 70
7. Combined mF2C functionalities 71
7.1. User Registration to mF2C system 71
7.1.1. Architecture 72
7.1.2. Experimental set-up 72
7.1.3. Results 72
7.2. User (and resources) participation in mF2C 74
7.2.1. Architecture 75
7.2.2. Experimental set-up 75
7.2.3. Results 76
7.3. Failure of a mF2C area leader 76
7.3.1. Architecture 77
7.3.2. Experimental set-up 77
7.3.3. Results 77

8. Roadmap towards IT-2 mF2C PoC 78
8.1. Lessons learned from IT-1 mF2C design and implementation usage 78
8.2. mF2C PoC extensions for IT-2 79
8.3. Demonstration strategy update for IT-2 80
9. Conclusions (and Next Steps) 81
References 82

List of figures

Figure 1 Components layout and applicability overview for IT-1 11
Figure 2 Infrastructure monitoring 13
Figure 3 UPC Testbed 14
Figure 4 Infrastructure of use case 1 14
Figure 5 Smart Boat Application screenshots 16
Figure 6 Objects tracking 17
Figure 7 Test lab with objects tracking 18
Figure 8 Heatmap to represent most visited places in the field 19
Figure 9 ESM Use Case for IT-1 26
Figure 10 UC1 normal scenario workflow 34
Figure 11 UC1 threshold alarm with no fog workflow 34
Figure 12 UC1 Jammer alarm without fog workflow 35
Figure 13 UC1 normal scenario with fog workflow 36
Figure 14 UC1 threshold alarm with fog workflow 37
Figure 15 UC1 jammer alarm with fog workflow 38
Figure 16 UC1 emergency resolution service workflow 39
Figure 17 Photos of the testbed deployed at UPC 40
Figure 18 UC1 KPI measurements experiment 41
Figure 19 Smart Boat - Network availability scenarios 44
Figure 20 UC2 basic component overview 47
Figure 21 UC2 application - logical/software diagram 48
Figure 22 UC2 architecture in layers 50
Figure 23 Data flow overview 51
Figure 24 Door Sensor event data flow diagram 52
Figure 25 Sensor histogram preparation data flow diagram 53
Figure 26 Smart Boat sensor screenshots 54
Figure 27 Calculated/optimal network coverage map for 3G/4G 55
Figure 28 UC3 final system architecture 56
Figure 29 UC3 dataflow between different layers 58
Figure 30 UC3 IT-1 system architecture 59
Figure 31 Android App screenshots (splash screen, Privacy Policy Terms, Indoor map) 65
Figure 32 Android App screenshots 65
Figure 33 Point of Interest (POI) proximity handling 66
Figure 34 Topic message notification 67

Figure 35 Big data analysis, predictive analysis (IT-2) 68
Figure 36 Dashboard (partially implemented in IT-1 and to be completed in IT-2) 69
Figure 37 Combined mF2C Functionalities Demo Architecture 72
Figure 38 Registering a new user 73
Figure 39 mF2C dashboard 73
Figure 40 Service Catalogue 73
Figure 41 Registering a new service 74
Figure 42 Discovery + Authentication 75
Figure 43 Experimental Set-up for Discovery + Authentication 75
Figure 44 mF2C front end activity when discovering a new device 76
Figure 45 Architecture for leader failure 77
Figure 46 Leader failure in the frontend 78
Figure 47 Leader failure, the RaspberryPI becomes the leader 78

List of tables

Table 1. Acronyms 10
Table 2. Components validated in each of the demonstrations 24
Table 3. List of devices for the demonstration of UC1 in IT-1 29
Table 4. Current implementation of the IT-1 components 32
Table 5. Delay in use case 1 41
Table 6. Current hardware for Smart Boat use case 46
Table 7. Current hardware and base software in UC3 61

Executive Summary

This deliverable is the logical continuation of *D5.1 mF2C reference architecture (Integration IT-1)* submitted in project month 16. D5.1 deals with the interaction between the mF2C components in the first iteration of the project in order to provide the first integrated version of the project reference architecture.

The present deliverable describes the deployment of this first proof-of-concept of the mF2C architecture in three real-life use cases simulated in three testbeds developed specifically for this project. Three scenarios are imagined:

- Emergency Situation Management in Smart City, in which an alarm is simulated and the intervention of the emergency services managed in an optimised fashion.
- Smart Boat, in which monitoring of a fleet and control system for boat owners and users are simulated.
- Smart Fog-Hub Service, which proposes a proximity marketing and topics adviser system for travellers in crowded places such as airports, railway stations or shopping centres.

This deliverable reports on the challenges faced during the demonstration deployments as well as the assumptions and modifications that resulted necessary in order to perform these real-life tests. The tasks realised allowed deeper understanding of the mF2C architecture and allowed the identification of current issues, as well as the formulation of corresponding solutions to be implemented in the second iteration of the project.

The first preliminary results obtained are also presented, which demonstrate the relevance of the proposed architecture in the use cases selected and illustrate the benefits of using the technology developed in the mF2C project in each situation. For instance, reduction of the services' latency is demonstrated in both the Emergency Situation Management and Smart Fog-Hub services whilst reliability is demonstrated in the Emergency Situation Management and Smart Boat use cases. Finally, an increase in coverage is demonstrated for the Smart Boat use case.

1. Introduction

1.1 Purpose

The objective of this deliverable is to describe the design, implementation and evaluation of the use case, scenarios that will allow us to validate the mF2C management system proposed in the first iteration of the project. The three selected scenarios are presented, as well as the deployment of the mF2C architecture on each of the testbeds, and the preliminary results are detailed. The assumptions and evolutions envisaged for the second iteration of the project will also be addressed. A second version of this deliverable will be released at the end of the project, that will present the process and results for the implementation of the complete agent developed during the second half of the project.

The deliverable *D2.6 mF2C Architecture (IT-1)* presented the initial architecture for the first implementation (IT-1), while each of the architecture blocks were described in more detail in deliverables *D3.3 Design of the mF2C Controller Block IT-1* and *D4.3 Design of the mF2C Platform Manager block components and microagents* IT-1. *D5.1 mF2C reference architecture (Integration IT-1)* reported on the modifications of functionalities supported by the mF2C blocks for IT-1. This document builds on all of those previous deliverables.

1.2 Structure of the document

This deliverable is structured in 9 sections:

Section 1. The current section introduces this document and presents the scope that will be described in the following sections.

Section 2. describes the proof-of-concept that has been developed during IT-1 of the project and introduces the three use cases that will be used to demonstrate the functionality and benefits of the mF2C system architecture in real-life scenarios.

In section 3., the strategy adopted for the architecture development during IT-1 and the reasoning behind this strategy are described. The pilot deployment is described and the functionalities that will be tested in each demonstration detailed. Finally, the limitations are mentioned.

Sections 4, 5, and 6. detail the three proposed use cases, the testbeds they use, the structures that are being used for this first iteration, the data flow diagrams, the experimentation set-ups and the preliminary results, as well as introduce briefly business considerations for each one of the scenarios, that will be developed in detail in Work Package 6.

Section 7. deals with the combined mF2C functionalities which is to say those mF2C functionalities that are demonstrated independently of the execution of an mF2C service in IT-1.

Section 8. addresses the road map towards the second iteration and describes the improvements that must be applied to the architecture as well as the new functionalities that were not integrated in this first iteration but that have been demonstrated as necessary in order to improve the architecture.

Finally, section 9. will present the conclusions reached during the work presented in this deliverable.

1.3 Glossary of Acronyms

Acronym	Definition
API	Application programming interface
CA	Certificate Authority
CAU	Control Area Unit
CIMI	Cloud Infrastructure Management Interface
CSR	Certificate Sign Request
DoA	Description of the Action

DoS	Denial of Service
I2C	Inter-Integrated Circuit
IoT	Internet of Things
IT-#	Iteration 1 or Iteration 2
JVM	Java Virtual Machine
LoRa	Long range LPWAN
LPWAN	Low-Power Wide-Area Network
MAC	Media Access Control
NUC	Next Unit of Computing
OPEX	Operating Expenses
PCs	Personal Computers
PoC	Proof of Concept
POI	Points of Interest
QoS	Quality of Service
REST	Representational State Transfer
SDK	Software development kit
SDR	Software Defined Radio
Timeout status	When a device, for example, a sensor, is not sending any data or status information.
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VM	Virtual Machine

Table 1. Acronyms

2. PoC and Application description

2.1 mF2C PoC implementation for IT-1

For IT-1, the main goal was the implementation of a preliminary version of the mF2C system which could be used to validate the architecture, functionalities and workflows defined in deliverables 2.6, 3.3 and 4.3 [1,2,4].

To achieve a working prototype which could fit the needs of the use cases by the end of IT-1, compromises had to be reached whereby some components and functionalities have been either postponed to IT-2 or simplified to provide only the indispensable purposes. Also, because of the hands-on experience and evolving technical awareness around fog-to-cloud computing, some components have been declared obsolete and new ones have been included in the architecture.

One of the newly added components is the Control Area Unit client (CAU client) which is used to interact with the mF2C Control Area Unit (CAU) and Certificate Authority (CA) middleware, during the mF2C agent authentication workflow.

Figure 1 below provides an overview of the status of applicability of the mF2C components for IT-1, including new components and highlighting those that have been either completely excluded or postponed for IT-1.

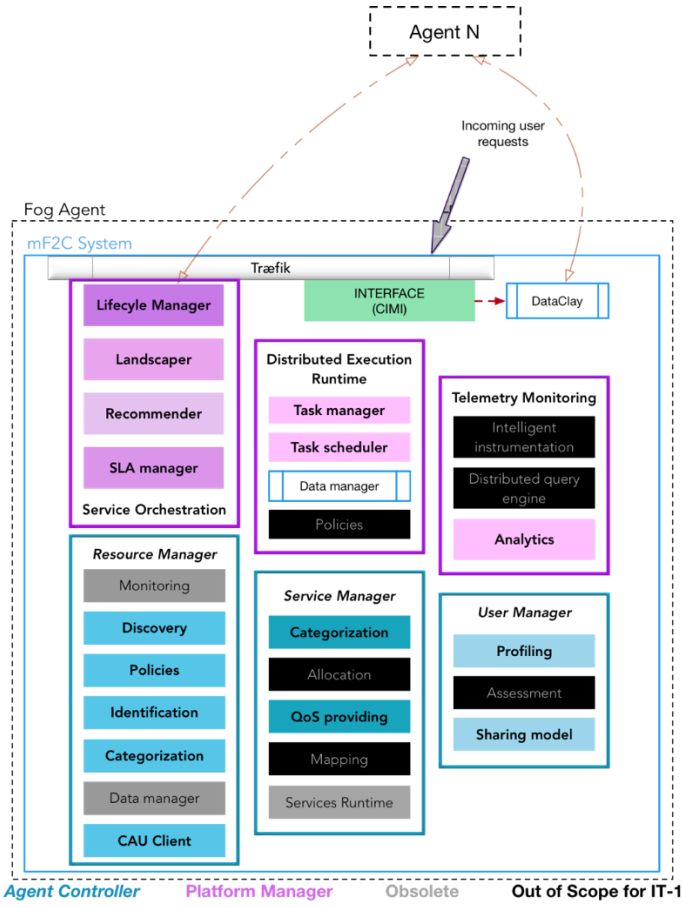

Figure 1 Components layout and applicability overview for IT-1

By prioritizing collaborative development and functionality, a modular approach has been adopted, based on the Platform Manager and Agent Controller components proposed in the initial mF2C architecture described in D2.6. This modular design allowed the developers to work in parallel on their respective components, in a standalone manner, avoiding the usual critical dependencies and programming conflicts that originate when building monolithic applications.

To maximize portability and facilitate the components' integration onto a final application deployment, Docker was the chosen platform for building, distributing, running and testing the components. Also, thanks to the compliance with the Docker Compose tool, the full deployment of the IT-1 mF2C system was orchestrated through a single YAML file (Compose file) in which all the components were referenced, configured and automatically linked with each other, under an isolated local network.

The components' integration was achieved by selecting all components that rely on inputs from either mF2C users or neighbouring internal components and have them provide an HTTP-based RESTful API. These are either visible only from within their isolated local network or from outside the mF2C agent, depending on whether the API ports had been published to the hosting device at the time of deployment (through Docker Compose). Finally, the mF2C components' integration was completed by complying with a set of predefined development conventions that fixed:

- the REST APIs' ports to be used by every component
- the service name to be given to the component at the time of deployment
- the components' API visibility (interval vs. external for inter-agent and user interactions)
- the code versioning and distribution strategies (through GitHub and Docker Hub).

The final steps for the validation of the PoC implemented in IT-1 were the integration tests, which were conducted against each one of the workflows. The latter are presented in deliverables *D3.3 Design of the mF2C Controller Block IT-1* and *D4.3 Design of the mF2C Platform Manager block components and microagents (IT-1)*. They have been included in the final IT-1 implementation, as described in deliverable *D5.1* [5]. This validated PoC was integrated on all three use cases described below.

2.2 Emergency Situation Management in Smart Cities (ESM)

As the world's population grows in the next few years, the urban areas will have to deal with the strain of hugely growing needs and demands. The use of the latest technological advances in the digital and communication fields will help resolve issues in a way that has been coined as "Smart City". This first use case caters to this field of innovation by demonstrating an alarm manager for emergency situation management in smart cities. In critical situations, the response time of the emergency vehicles that assist people, such as police, ambulances and fire engines is of outmost importance as a short amount of time can make a big difference. Through fast intervention, the severity of an injury can be reduced, with the corresponding impact on wellbeing and healthcare costs, sometimes even avoiding deaths, as 66% of those happen in the first twenty minutes in the case of car crashes [6]. For this reason, a big effort has been dedicated to this topic. For instance, Qin et al. [7] report on control strategies of traffic signal timing transition for emergency vehicle pre-emption so that the approaching emergency Vehicle can cross the intersection safely at its operating speed instead of having to slow down at each intersection [8]. Also, companies like GTT have released the commercial Opticom system solution that deals with priority control of traffic lights for emergency vehicles [9]. Actually, numerous patents have been applied for over the years for such systems, some of them have even been granted. [10,11].

The main service presented here will be a decision-making system that, according to an inclination sensor that monitors smart infrastructures, will declare whether a specific situation is normal or represents an emergency. In the latter case, Emergency Situation Management in a Smart City context will be provided by processing information and triggering the intervention of the relevant emergency services. If a sensor reports a value higher than an alarm threshold, the alarm manager will report an emergency situation to the software that will trigger its alert methods. Furthermore, in order to improve the solution's security, the alarm manager is able to detect whether the communication is active or is affected by interferences, whether accidental or intentional. If a lack of communication is detected, the Jammer detector (identifier of interference signals) is automatically powered up and configured to detect interferences in the channel used by the Long range LPWAN (LoRa) network. The

Jammer detector communicates with the Gateway via Ethernet (wired), in order to prevent the data transmission between themselves from being affected.

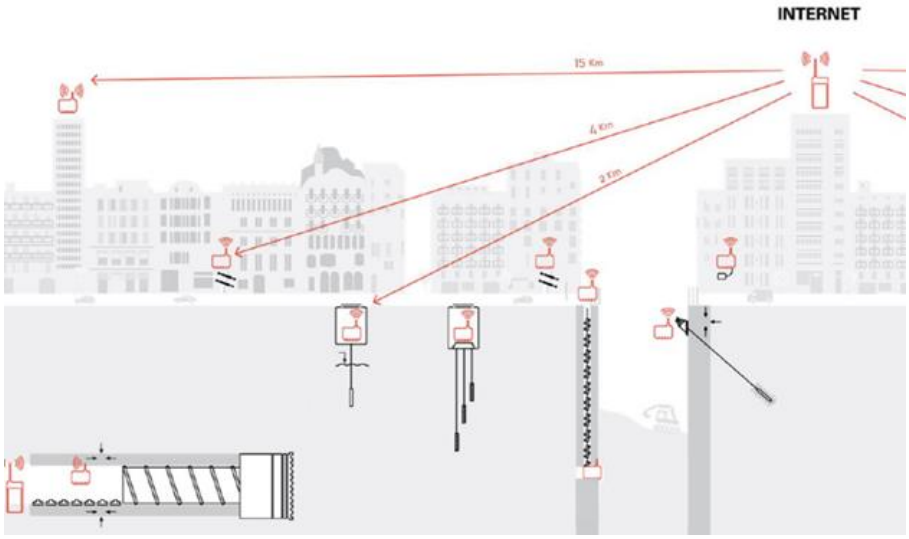

Figure 2 Infrastructure monitoring

A testbed has been developed by UPC that incorporates Worldsensing's technology, which will be used to validate the emergency situation management in smart city use case.

The emergency service is emulated in the testbed shown in Figure 3, where 3 buildings are shown (red, pink and blue boxes):

- The building being monitored (red building in the image below)
- A hospital, with the ambulance inside (blue building in the image below)
- A fire station, with the fire engine inside (coral building in the image below).

Additionally, other elements are illustrated, such as:

- Roads and streets
- Traffic lights.

In this scenario, we will consider two types of actions:

- An action produced by a Jamming attack that interrupts the connection between the inclinometer and the data reception centre
- An action produced by a seismic movement or any other situation that causes a collapse of the building
    - Which at the same time triggers new actions that form part of the emergency service such as the computation of the paths from the hospital and the fire station to the collapsing building, as well as the establishment of a green corridor (traffic lights) along these paths

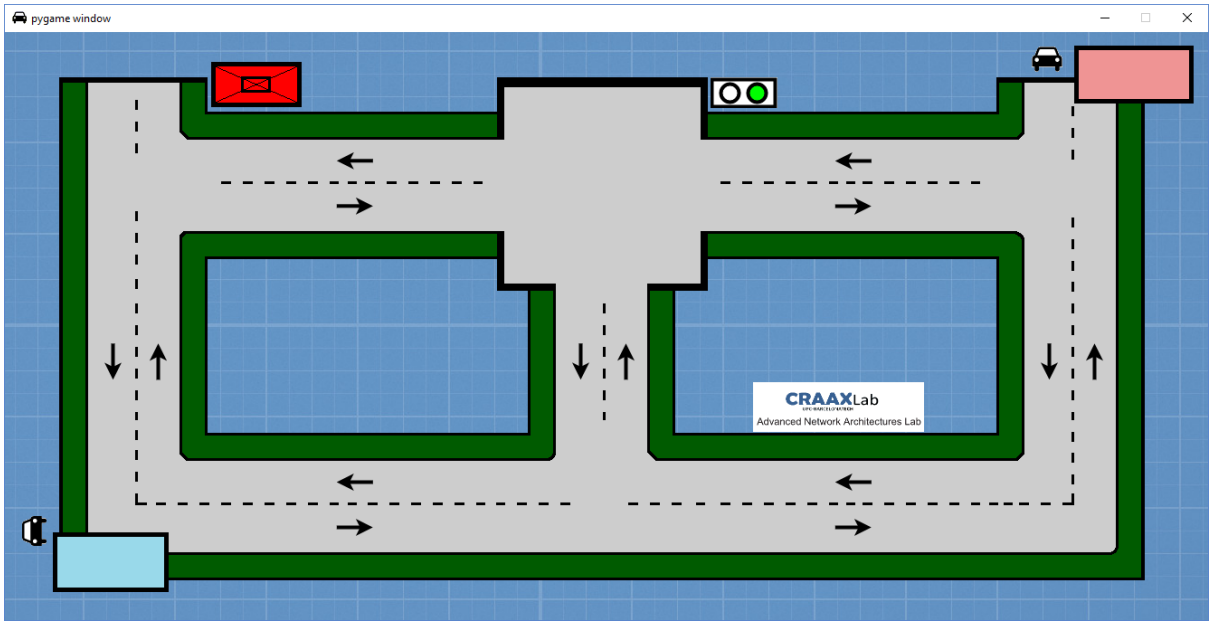

Figure 3 UPC Testbed

The two areas of action, shown in Figure 4, represent two separated zones of a Smart City.

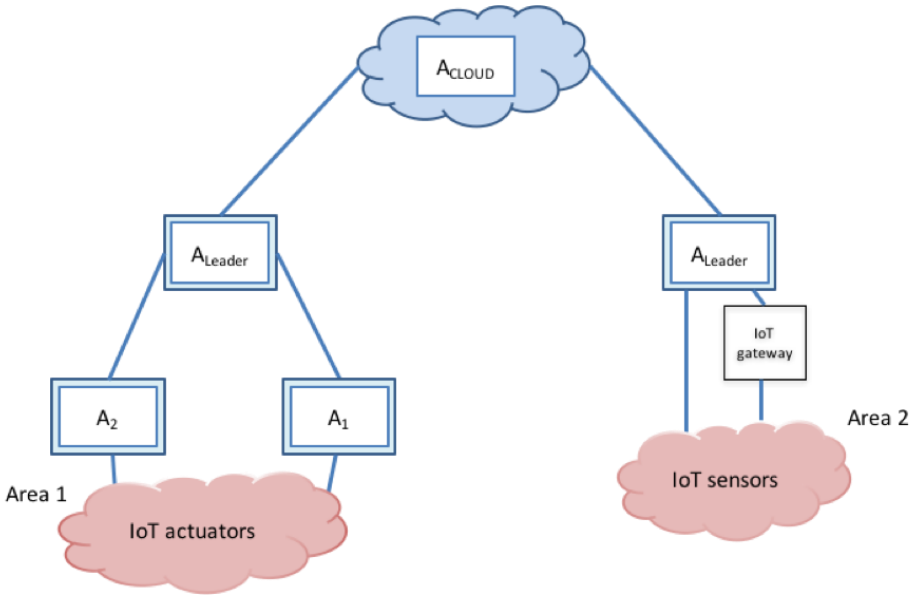

Figure 4 Infrastructure of use case 1

The part of the emergency service that responds to a trigger (actuators) set off by an alarm event is located in the first area and the elements in this area are:

- Leader
- Agent 1 → connected to a fire engine and to a traffic light.
- Agent 2 → connected to an ambulance

All three elements in the area can also be used as computational elements by the mF2C system. The event detection itself is located in the second area, where the activation and the triggering of the emergency will take place due to the simulation of an excessive inclination of a building produced by a seismic movement. It consists of a leader connected to a Jammer detector and other different IoTs connected through a gateway:

- Leader → connected to a gateway and the jammer detector. Both via Ethernet.

- The Gateway is also connected through LoRa to an inclinometer, which detects inclination and provides temperature.

The leader in this area is also a computational element in the mF2C system. The two areas are interconnected through the Cloud via the leaders.

Detailed results of the implementation of this use case are presented in Section 4, Use Case #1: Emergency Situation Management in Smart Cities (ESM).

2.3 Smart Boat Service (SBS)

This use case demonstrates a Smart Boat monitoring and control system for boat owners and users. The main service is to provide the owners and users with insight of the boats status over the fog or cloud – depending on the network possibilities.

The Smart Boat application caters to two types of users: the owners of the vessels, but also the guests, which are the sailors or skippers currently operating or visiting the boat. This main distinction is introduced to provide the ability to share the sensor data and boat overview information to a third person currently renting the boat or to friends and relatives, which might like to track your marine journey from the comfort of their homes. The interface of the owner or guest are similar with small differences: for instance, the owner has more permissions and control over the boat and can add or remove guest tokens, while the guest can only see the information he is allowed to. The overview of the boat consists of the following screens in the Android application:

- Sign in, user selection, settings
- List of boats
- List of sensors/alarms
- Specific sensor view
- Map and tracks
- Component status - which components are installed in the system.

The example screenshots of the application on Figure 5 illustrate the user's overview of the components and sensors.

Boat users require reliable hardware mounted on the boat providing them insights on the boat's current state, all the time, regardless of whether they are on the boat or not. This is due to the fact that boats are relatively seldom occupied as they may only serve free-time activities, while at other times an owner would like to have overview of their expensive asset. Boat rentals and hobby usage are becoming increasingly popular and serve a high-value market. All marine equipment, and especially upkeep and maintenance, has a relatively high cost, which is significantly larger than the cost of IoT equipment and services available for the boats. This opens up a very dynamic and expressive market for high-end high-value-add solutions that cloud, fog and IoT providers strive for in terms of business. The potential impact and target market is thus very large, especially with a solution that can ease and enhance the deployment and execution of services for the developers and the services' users alike.

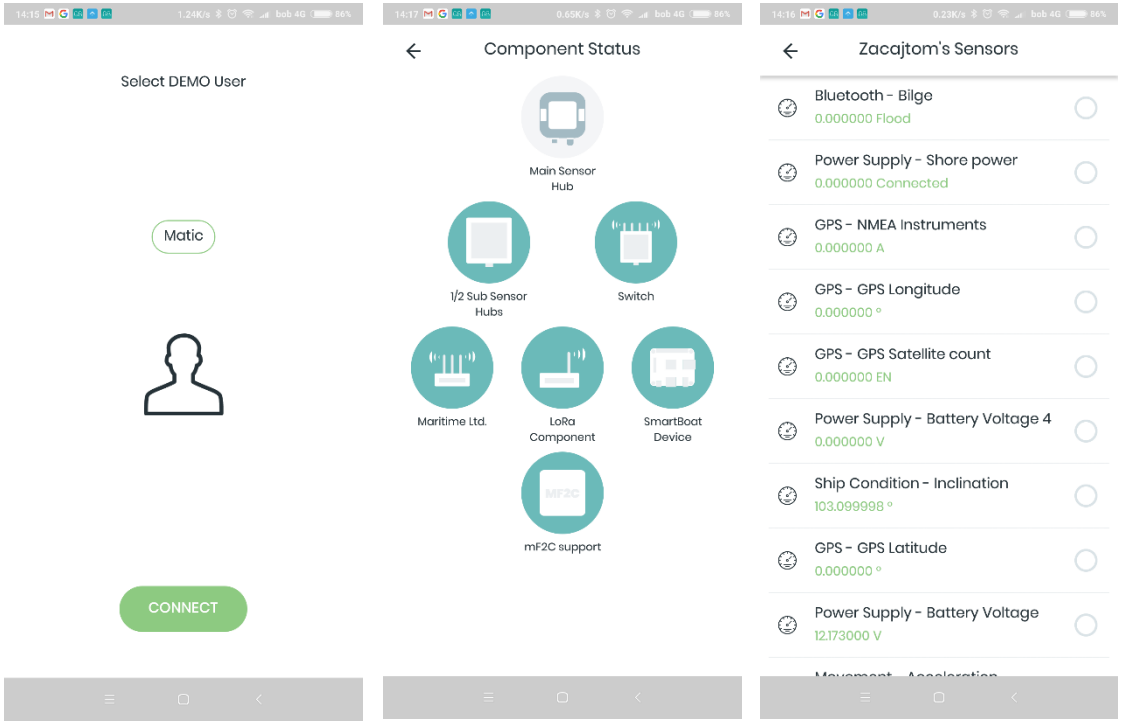

Figure 5 Smart Boat Application screenshots

Apart from the overview of the sensors, the user has the possibility to retrieve events and mark them as *seen*. This action also resets the LED light on the Smart Boat device. For example, the bilge water level or door sensor can be monitored to detect whether the boat is sinking or if someone is trying to break into the boat when it is not being used. In these cases, the Smart Boat device turns on an LED and the user immediate receives notification on a mobile device, allowing him to perform responsive actions. The Smart Boat Use Case deployment includes:

- Two agents
  - one leader one worker
  - each agent connected to its own boat environment
- Two users:
  - The owner: who has visibility over both boats
  - The guest: who has visibility over only one boat
- The displaying methods will use the mF2C *DER* component (COMPSs engine) to prepare some calculations.

The details of the deployment are detailed in Section 5, Use Case #2: Smart Boat Service (SBS) and presented in Figure 19.

2.4 Smart Fog-Hub Service (SFHS)

According to recent researches by McKinsey and Gartner, business applications of IoT have greater economic impact than consumer applications. While consumer uses of IoT got a lot of attention and show a tremendous potential for creating value, there is even greater potential value from IoT use in business-to-business applications. This happens more frequently in environments with high concentration of IoT devices. These places can be considered the building blocks of the Smart City, that can be interconnected leveraging the main pillars of the so-called OpenFog Reference

Architecture. The airport is such an example: some billion passengers will concentrate in the airports with an average of two connected devices for each passenger. Current technology infrastructures and architectures have not been designed to process in real time the great amount of information data that is being made available from such a number of devices with a so high concentration, so Fog Computing is emerging as an architectural model that places itself between the Cloud and the IoT, expanding Cloud Computing and Services to the IoT objects.

The third experimental use case aims at extending the concept of "cloud hub" to a new concept of "fog hub", driven by real market needs. This hub caters to spaces with recurring concentrations of people and objects that communicate and interact, like airports, railway stations or shopping centres. The use case is under development in the Engineering Labs and will be moved to the Cagliari Elmas Airport next year.

The main service will be proximity marketing and topics adviser for travellers: the ability to track the presence of people and other objects in the field, creating engagement between them, and providing suggestions on shops or services nearby based on preferences, or suggestions on the best use of airport services to minimize queues and delays.

In the final testbed, the fog elements will be deployed in the airport lounge area in order to create a grid for Wi-Fi coverage.

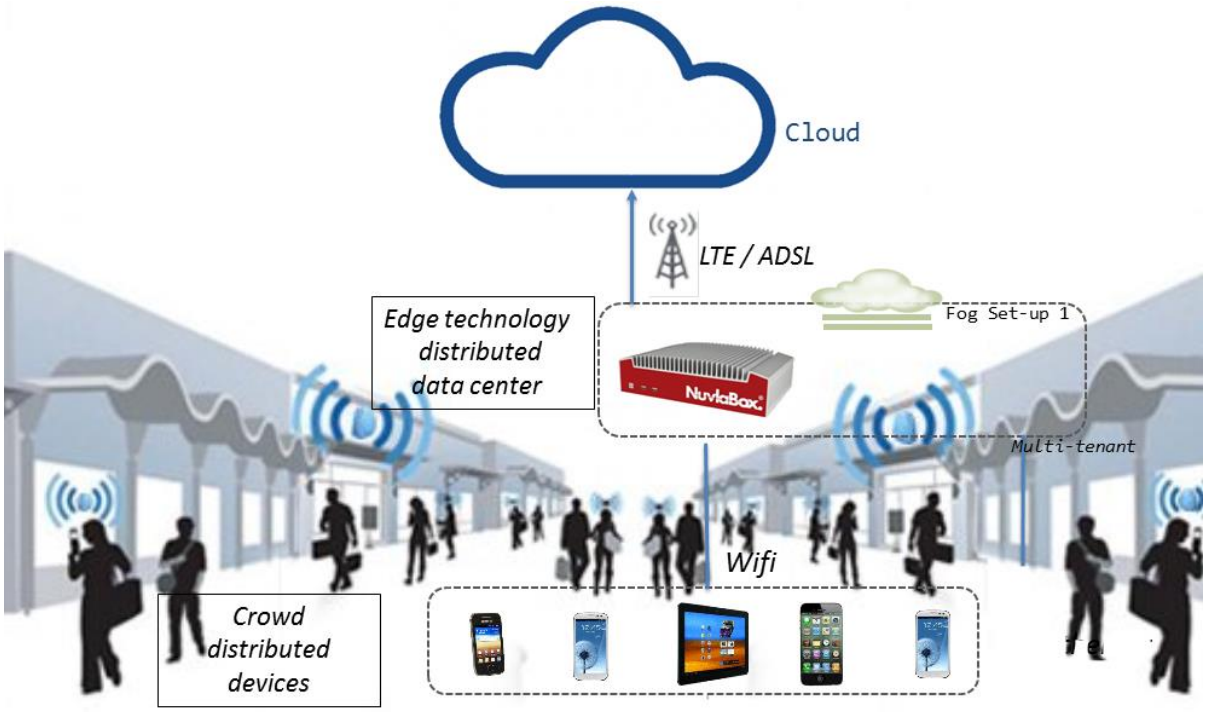

Figure 6 Objects tracking

A provisional testbed has been developed by Engineering. The current environment supporting IT-1 version is an open space on the Engineering Campus, used to simulate the airport lounge, with shops and other Points of Interest (POI). Also, all relevant airport events are simulated: the official timetable of the airport has been used to create the full list of events related to departures like open check-in, assign gate, call flight, last call, close flight/gate.

This testbed is being used to validate the proximity marketing and topics adviser for travellers' service for the Use Case 3.

Figure 7 Test lab with objects tracking

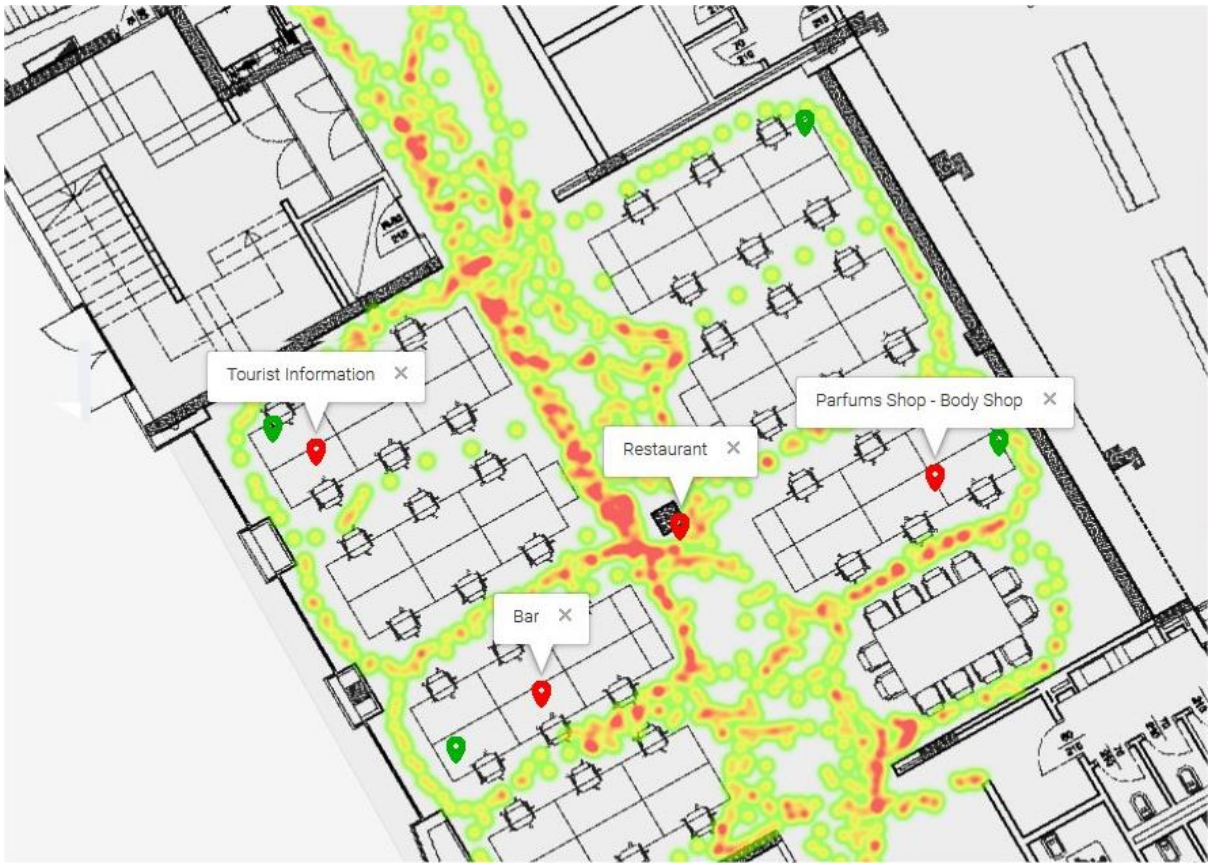

Figure 8 Heatmap to represent most visited places in the field

For IT-1, a first version of the system architecture has been defined with the following layers:

- Android smartphones, used by travelers, running a custom android app to be engaged with the system;

- The edge layer, composed by four RaspberryPI3 acting as access elements with secure data transmission, providing session management and supporting position tracking;
- A first fog layer, with a HP laptop that runs the mF2C agent and acts as fog worker node, providing processing and resource capabilities;
- A second fog layer, based on a NuvlaBox mini (commercial product from project partner SIXSQ[1]), acting as fog leader and providing real-time computing and storage resources;
- A Cloud layer, based on an OpenStack instance, for scalable computing power and long-term storage for all data produced that needs to be analyzed;

The mF2C agent runs in the Cloud and Fog elements as the current version is not supported on RaspberryPI and android smartphones. The android app to be installed in the smartphone implements both security and privacy features to preserve managed data both at rest and in transit, with a security level comparable to those adopted by the mF2C agent.

The resulting architecture is described in more details in chapter 6 Use Case #3: Smart Fog-Hub Service (SFHS).

[1] NuvlaBox mini is a “Plug-and-play edge device, delivering savings with a simple, secure and private "cloud-in-a-box" solution provided by project partner SIXSQ. More information available at: https://sixsq.com/products-and-services/nuvlabox/tech-spec

3. Demonstration strategy for IT-1

3.1. PoC deployment strategy

This section describes the strategy used to demonstrate, and thus validate, the proposed PoC. It is a summary of the whole description made in deliverable D5.1.

In IT-1, the demonstration strategy is divided into two main categories:

- Individual or combined mF2C functionalities: Some of the mF2C functionalities are demonstrated independently of the execution of an mF2C service. For the sake of illustration, the tested functionalities are:
  i) the registration process (both of a user and also of a new mF2C service)
  ii) the discovery of a new agent, and
  iii) the failure of a leader.
  It is worth mentioning that although identified as individual functionalities, they are not deployed as a unique feature. For example, the discovery process includes obtaining credentials for authentication and categorization.
- Execution of mF2C services: The mF2C agent will be demonstrated in the three different use cases included in the project. Since not all functionalities will be deployed in IT-1 for the three proposed use cases, we have also proposed an additional use case, referred to as "Hello world", aimed at representing a generic mF2C service, using all mF2C functionalities linked to the execution of a service.

3.2 Pilot deployment

For IT-1, the mF2C system are deployed through Docker Compose, having each component running as a service in its own container. By default, Docker provides enough portability, isolation, security and flexibility to enable the collaborative development of a modular architecture, where every component can be executed on its own, without any core dependencies. At the same time, by using Docker, the IT-1 functionalities can be demonstrated on multiple device types, without having to specifically build architecture-specific components. This results in application resources such as JVM not being shared by the containers so the resource usage is not optimised, but this drawback will be targeted in IT-2.

The installation of the mF2C System is provided through a single Docker Compose YAML file (version 3) as indicated in section 2.1.

This YAML file has one service definition per component, plus additional auxiliary services like Traefik. All the services are deployed by default in the same Docker network, which allows the different components to find each other by name, while providing isolation from any other non-mF2C containers and system processes that might be running in the host device.

The requirements for deploying the IT-1 mF2C system are:

- Docker CE 17.12.0+
- Docker Compose 1.18.0+
- 2GB of RAM or more

The steps for deploying the mF2C agent in a device are:

1. Docker and docker-compose is required for running the agent in the system. Instructions can be found at https://docs.docker.com/install/ and https://docs.docker.com/compose/install/ for the user OS type
2. Download all the required files from the original repository. This operation can be realized via git in a terminal typing the following command:

$ git clone https://github.com/mF2C/mF2C.git

3. Enter into the docker-compose directory
4. (Optional) Before starting the agent, some options and modules can be configured by modifying the docker-compose.yml for .env file. Instructions for that step can be found on the official documentation here:

   http://mf2c-project.readthedocs.io/en/latest/developer_guide/installation.html

5. Start the agent executing the following command in a terminal:

   $ docker-compose up

Alternatively, there is an automated script for Linux devices that configures some modules and launches the agent. It can be found in the Linux directory, inside the downloaded repository.

3.3 Tested functionalities and limitations for IT-1

As stated in D5.1, some mF2C components, installed as single containers, have been designated as "core components" for IT-1 because of their crucial role in the validation of the main functionalities proposed in the demonstration strategy. These components are:

- **Interfaces:** The main mF2C interface and entry point for mF2C users is the Cloud Infrastructure Management Interface (CIMI). This component also provides every other internal component with the interaction layer for DataClay and the mF2C storage backend. Besides CIMI, which obviously needs to be reachable from outside the device, the Lifecycle Manager module (in the Service Orchestration component) is also exposed and reachable over the network, so that the Lifecycle Manager components from multiple mF2C agents can communicate directly with each other. Traefik is used as an auxiliary core component for doing the reverse proxying amongst the different blocks that need to be exposed over the network.
- **Database: DataClay** is the database used in the project for both mF2C data system as well as application data.
- **Service Orchestration:** This component has the next modules:
  - **Lifecycle Manager** to deploy service instances (applications).
  - **Landscaper** to aggregate and display the infrastructure resources.
  - **Recommender** to assist the Lifecycle manager to select the best resources by means of a recipe, consisting on "recipe" = (#cores, core type, storage, #IoT need, IoT type)
  - **SLA Manager** to create and validate service level agreements (SLAs).
- **Telemetry Monitoring**
  - **Analytic**[2] to analyse monitoring data to assist the **Recommender.**
- **Resource manager.** This component has the next modules
  - **Discovery** to discover new devices joining the system.
  - **Policies** to orchestrate some of the functionalities that work together.
  - **Identification** to assign a unique identification to all the devices participating in the system,
  - **CAU client** This component is triggered when an agent arrives to the fog area and receives beacon from the leader. Then, CAU-client generates a certificate sign request (CSR) and establishes a secure channel to the corresponding CAU in the fog area, by using transport layer security (TLS). This component sends CSR through the secure channel to the CAU and waits for the signed certificate.
  - **Categorization** to characterize the resources of all the devices participating in the system.

[2] Although according to the architecture the Analytics module corresponds to the Telemetry monitoring, has been integrated in the same container of the Recommender.

- **Distributed Execution Runtime** to provide the execution of Java applications in a distributed environment and mostly composed by COMPSs.
- **Service manager** with the next modules:
  - **Categorization** to categorize services according to their characteristics and needs.
  - **QoS providing** to take into account QoS when assigning resources to services.
- **User manager** with the next modules:
  - **Sharing module** to allow the user to share part or all of his/her resources with the system.
  - **Profiling** to register the information related to the user, taking into account his/her roles and preferences.

All these modules belong to the Agent software which is installed in all the devices willing to join the mF2C system. However, there are other developed modules: CA (Certified Authority) and CAU (Control Area Unit) which will be out of the agent's software. The CA will be in cloud and the CAU in a special device devoted to this functionality and located in each one of the fog areas.

- **Security block:**
  - **CA:** It is certificate authority. It receives CSR from CAU through secure channel, signs certificate and sends back to CAU.
  - **Control Area Unit (CAU):** This component acts as a secured smart gateway between the agent and the certificate authority (CA). It has two sides, it includes a server and a client. The server side establishes a secure channel with CAU-client in the agent to receive CSR. The client side establishes a rest call through the TLS communication. CAU sends the CSR to the CA to be signed, it receives the signed certificate, and finally it sends it back to the CAU-client in a completely secure way.

All the components that are expected to interact with other components are equipped with a REST API which, unless intentionally exposed, will only be reachable by other blocks within the same local network inside the device.

All these components will be validated in the different use cases and in the mF2C individual demo functionalities, although not all the components are validated in all the demos. The following table summarizes which components are used and validated in each one of the demos:

AGENT						
Platform manager	**mF2C functionality: Registration+ Identification**	**mF2C functionality: Discovery+ Authentication + Categorization**	**mF2C functionality: Leader failure**	**Use Case 1**	**Use Case 2**	**Use Case 3**
Service Orchestration	**No**	**No**	**No**	**Yes**	**Yes**	**Yes**
Lifecycle Management	No	No	No	Yes	Yes	Yes
Landscaper	No	No	No	Yes (manual)	Yes (manual)	Yes
SLA Management	No	No	No	Yes (manual)	Yes (manual)	Yes (manual)
Recommender	No	No	No	Yes (manual)	Yes (manual)	Yes
Distributed Execution Runtime (COMPSs)	**No**	**No**	**No**	**No**	**Yes**	**Yes**
Telemetry	**No**	**No**	**No**	**Yes**	**Yes**	**Yes**

Monitoring						
Analytics	No	No	No	Yes (manual)	Yes (manual)	Yes (manual)
Agent Controller	**mF2C functionality: Registration+ Identification**	**mF2C functionality: Discovery+ Authentication + Categorization**	**mF2C functionality: Leader failure**	**Use Case 1**	**Use Case 2**	**Use Case 3**
Service Management	**Yes**	**No**	**No**	**Yes**	**Yes**	**Yes**
Categorization	Yes	No	No	No	No	Yes
QoS Providing	No	No	No	Yes	Yes	Yes
Resource Management	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**No**
Discovery	No	Yes	Yes	No[3]	No	No[1]
Policies	Yes	Yes	Yes	Yes	No	No
Identification & naming	Yes	Yes	Yes	No	No	No
Categorization	No	Yes	Yes	No	No	No
CAU client	No	Yes	No	No	No	No
User Management	**Yes**	**No**	**No**	**Yes**	**Yes**	**Yes**
Profiling	Yes	No	No	Yes	Yes	Yes
Sharing module	No	No[4]	No	Yes	Yes	Yes
SECURITY BLOCK						
Security	**mF2C functionality: Registration+ Identification**	**mF2C functionality: Discovery+ Authentication + Categorization**	**mF2C functionality: Leader failure**	**Use Case 1**	**Use Case 2**	**Use Case 3**
CA	No	Yes	No	No	No	No
CAU	No	Yes	No	No	No	No

Table 2. Components validated in each of the demonstrations

The assumptions agreed for IT-1 and reported in D5.1 are summarized below:

- Services are executed from the mF2C dashboard

[3] We assume that when the service of the use cases is executed, the resources are already discovered, identified, authenticated and categorized.

[4] The categorization module stores the device information in DataClay, including the amount of resources the user wants to share, and it is available for the Sharing module.

- The dashboard includes the portfolio of services categorized into different categories (IoT, data, smart cities…). Dashboard development is aligned with the definition of a service in CIMI, creating a JSON file compatible with the Service Manager
- For the sake of simplicity, in IT-1 the mF2C architecture considers three layers (cloud/fog/edge, where edge considers the IoT devices). All devices included in these layers deploy the mF2C Agent
- There is no horizontal communication (between devices in the same layer) among devices at control level. The communication is multilayer and vertical
- There is one leader and one backup selected in each fog area
- There is a limited set of categories for resources/services
- The processes of leader and backup selection need only to be very basic
- Clustering policy is set manually at bootstrap
- All services and mF2C functionalities fill a single container each
- A "recipe" is defined as the recommended devices in which to run a service = (cpu, memory, disk and network)
- The recommender matches the service characteristics (obtained by the Service Categorization module) as well as the analytics from previous executions (Analytics module) to generate the Recipe
- There is neither allocation nor mapping at the Agent Controller
- mF2C is a software-only agent (no specialized hardware) downloaded from the mF2C (web service) and installed at the registration process
- No QoS enforcing will be done by the QoS providing block. This block in IT-1 feeds the Lifecycle Manager with information about resources' suitability to execute a specific service, based on the SLA violation history received from the SLA management
- The SLA is set manually. The SLA management block detects violation
- The dynamic resources information is obtained from COMPSs notifying the Lifecycle Manager, when a task is done and the latter reporting to dataClay. The categorization module will also enrich this information
- The sharing block enables definition of shareable resources per user's device
- Device categorization includes:
    - Hardware: the device is static (e.g. run after the discovery process)
    - Hardware: the device is dynamic (run according to a certain policy)
    - IoT: the device is manually introduced
- COMPSs does not change the resources to be used to execute a task from those recommended through the Lifecycle Manager
- Application data may be stored in dataClay or in its own database
- The set of IDs (user and devices) are generated at cloud (mF2C cloud provider) at registration time
- A leader failure will not occur during app execution
- The interface with the mF2C agent is managed by CIMI.

Additional limitations appeared when trying to deploy the Agent Software in the use cases. Some of the use cases considered low computational devices such as RaspberryPI for implementing agents and leaders. However, in IT-1, the objective of the components integration has been the real inter-function between components, but not the optimization of the resulting agent. For this reason, the resulting agent is not suitable for implementation in devices such as RaspberryPIs, due to its memory requirements.

With this additional limitation, use cases have had to redesign the previous proposed architecture and topologies, replacing RaspberryPIs PC, VMs or laptops. For example, in Use Case 1, in a first approach and described in previous deliverables, the Smart Gateway and the Jammer detector were considered as fog devices. Also, in all three use cases, the RaspberryPIs were considered candidates to have the agent software implemented. However, due to the memory limitations of these devices and the memory requirements of the mF2C agent resulting from the integration of all the components, we have reconsidered the role of these devices. They are now considered as mere IoTs gateways. They are connected to sensors and actuators, without the agent software and with specific software for managing the reading from sensors or control of actuators. Finally, the same thing has happened with the mobile phones used in use case 3; as they cannot host an agent, they are considered as mere sensors tracking the position of the users, for IT-1.

4. Use Case #1: Emergency Situation Management in Smart Cities (ESM)

4.1. Complete Architecture Use Case

In case of an emergency in a city, there are two critical areas of action, as shown in Figure 9. These two represented, separated zones of a Smart City (one located near where the emergency situation occurs and the other one located where the emergency vehicles' intervention is triggered) are where most of the action takes place. The main outcome of this use case will be making decisions according to inclination sensor measurements to manage emergencies vehicle intervention in a smart city infrastructure. First of all, the emergency has to be detected and reported. In this use case, this is performed by World sensing's loadsensing solution, in which a tiltmeter detects the inclination of a smart infrastructure (for instance a building) and, as described in section 2.2, sends these measurements to the Gateway where this inclination is compared to a threshold, particularly set for the specific context being monitored. If this threshold is reached, the alarm is set off and the service will initiate three emergency applications provided by UPC. Also, the agent will forward the data to the Monitoring Software (Cloud and Fog).

The agent that detects the alarm is busy processing the comparison to the threshold and does not have enough calculation capability to perform the emergency applications so these are launched on other agents closer to the devices involved in that area of the demonstration, which launch the ambulance, the fire engine and the traffic lights. The best path for the vehicles to reach the emergency location is calculated and the traffic lights turned to green to allow them to reach their destination faster. At the same time, the availability of the communications will be verified using a jammer detector. This will detect jamming DoS attacks and help to prevent DoS situations. All this information will be visualized in the Cloud software. The following diagram represents the high-level architecture to be implemented in the first use case demonstration.

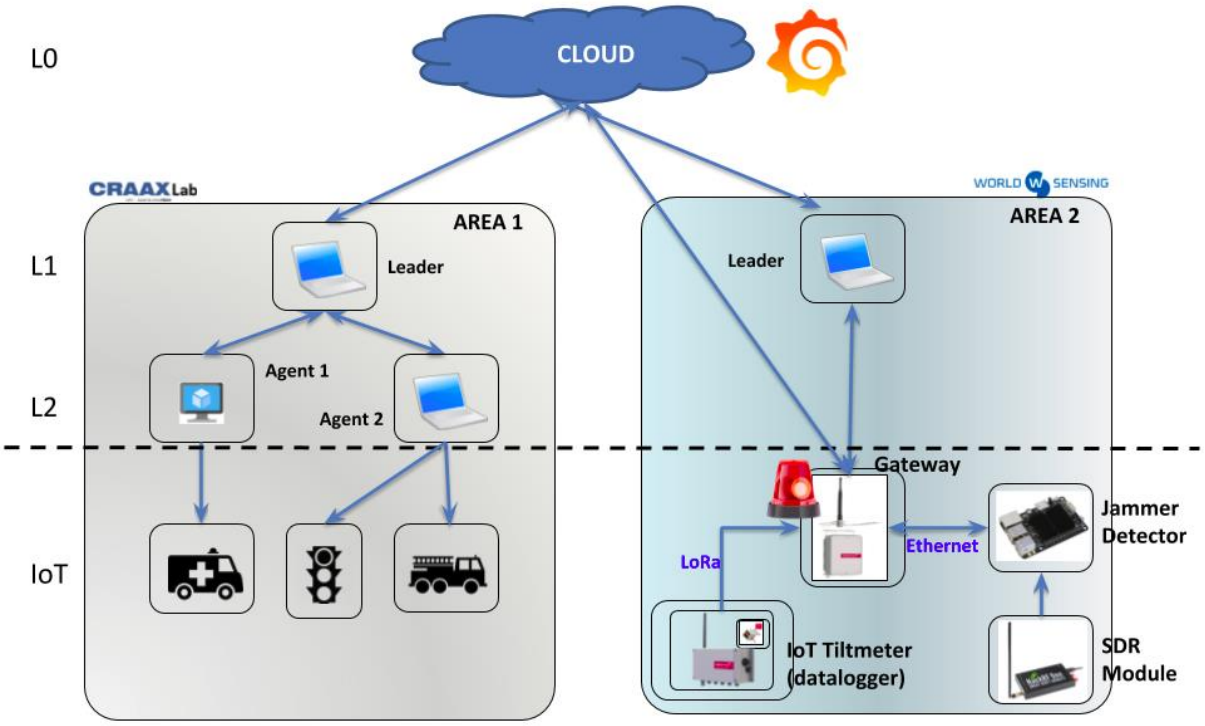

Figure 9 ESM Use Case for IT-1

Figure 9 shows the detailed architecture of the use case for IT-1 and separates the devices in four layers.

The higher level (L0) considered is the Cloud itself, composed of a Cloud Virtual Machine. This virtual machine runs on the cloud, located in a cluster at UPC premises, and is used to receive and store the data coming from the IoT sensors through the Gateway. It also provides real time visualization tools. This software is called the Monitoring Software and is in charge of receiving sensor data and controlling the alert cases. In case of an emergency situation, it will contact the Gateway to start the physical alarms and will start the emergency actions.
On the following level (L1 Fog), the first layer of the Fog, are located the leaders of both areas. A PC is present in both areas and acts as a leader. The area 2 Leader (right) is a machine that runs the Monitoring Software as a back-up. Having this software on the Fog is essential to provide more reliability and improve delays regarding the control of emergency situations. There is another fog level ((L2), which only exists in area 1. Two virtual machines (VMs) play the role of agents. In area 1, the three agents, i.e., the leader in L1 and the two agents in L2, offer their computational capacity to execute the applications of the emergency service.
Finally, the lower level includes the IoT elements only and is not an mF2C level as such, although it is part of the general architecture of the use case. This comprises various IoT devices, which are the main actuators and sensors of the use case, whose input is sent to the agents they are attached to. There are three sensing devices and three actuators:

- Gateway: used as a gateway for LoRa packets coming from dataloggers (IoT sensors) to be transmitted to the Monitoring Software. It can also transmit downlink messages to dataloggers to change configurations and sampling rate. The Gateway needs to be capable of turning on and off the physical alarms, as well as the Jammer Detector in case it is needed
- Datalogger with tiltmeters: this IoT device measures inclination on a structure and communicates it to the gateway via LoRa
- Jammer Detector: it is a smart sensor that analyses the data coming from its SDR unit to decide if interference signals are present on the wireless channel.

The previous IoT devices are sensing devices, and the IoT actuators are:

- Ambulance connected to its corresponding agent through a RaspberryPI.
- Fire engine connected to its corresponding agent through a RaspberryPI.
- Traffic light connected to its corresponding agent through a RaspberryPI.

4.1.1. Current Hardware and base Software for IT-1

The list of devices for the demonstration of use case 1 in IT-1 are the following:

Edge IoT devices (simple sensors and actuators)	
Loadsensing Tiltmeter's Inclinometer including an integrated LoadSensing datalogger	
Hardware	Software
Processor: ARM Cortex M4 48MHz Volatile memory: 32 kB Non-volatile memory: 256 kB + 4 MB flash Communication interfaces: I2C, UART, USB, LoRa modem (Semtech 1276) Sensor: Murata's SCA103T differential Inclinometer series	Operating system: FreeRTOS Software packages: sx1276 library Loadsensing application (112 kB flash, 24 kB ram, 4 MB storage)
Jammer Detector	
Hardware	Software

Processor: ARM Cortex-A53 1.5GHz quad core Volatile memory: 2 GB DDR3 Non-volatile memory: 8 GB eMMC 5.0 Communication interfaces: USB, Gigabit Ethernet Other: Great Scott Gadgets' Hack RF One SDR module	Operating system: Ubuntu minimal 16.04 LTS Software packages: Python, GNU Radio, Osmosdr, Libboost. Applications: SDRJD (1.2 GHz quad core, 1 GB RAM, GNU Radio compatible SDR module).
LoRa Gateway	
Hardware	Software
Processor: ARM 926EJS Volatile memory: 128 MB DDRAM Non-volatile memory: 128 MB NAND flash, 8 GB eMMC Communication interfaces: USB, Ethernet, LoRa	Operating system: Standard Long-Term Support Linux version 3.10 LoadSensing firmware (10 MB RAM, 40 MB flash, LoRa interface)
Ambulance	
Hardware	Software
Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit Volatile Memory: 1GB Non-volatile memory: 16GB Communication interfaces: USB, Ethernet / BCM43438 wireless LAN	Operating system: Raspbian Jessie (A Debian Linux operating system for RaspberryPI) Software Packages: Python, Car_Controller
Fire engine	
Hardware	Software
Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit Volatile Memory: 1GB Non-volatile memory: 16GB Communication interfaces: USB, Ethernet / BCM43438 wireless LAN	Operating system: Raspbian Jessie (A Debian Linux operating system for RaspberryPI) Software Packages: Python, Car_Controller
Traffic light	
Hardware	Software
Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit Volatile Memory: 1GB Non-volatile memory: 16GB Communication interfaces: USB, Ethernet / BCM43438 wireless LAN	Operating system: Raspbian Jessie (A Debian Linux operating system for RaspberryPI) Software Packages: Python, Traffic_Light_Controller
L2 Fog-capable devices	
Virtual machines (VMs) (UPC)	
Hardware	Software
Processor: Core i5 Volatile memory: 16 GB DDRAM	Operating System: Ubuntu 16.04 LTS

Non-volatile memory: 256 GB HD Communication interfaces: USB, Ethernet	Software Packages: Docker, docker-compose, Emergency_resolution_service
L1 Fog-capable devices	
WOS Leader (PC-1)	
Hardware	Software
Processor: Core i5 Volatile memory: 16 GB DDRAM Non-volatile memory: 256 GB HD Communication interfaces: USB, Ethernet	Operating system: Ubuntu 16.04 Software packages: Docker, InfluxDB Time-Series Database, Kapacitor Worldsensing's software
Leader PC-2 (UPC)	
Hardware	Software
Processor: Core i5-4570 CPU @ 3.20GHz x 4 Volatile Memory: 8 GB DDRAM Non-volatile memory: 1 TB HDD Communication interfaces: USB, Ethernet	Operating System: Ubuntu 16.04 LTS Software Packages: Docker, docker-compose, Emergency_resolution_service
L0 Cloud-capable devices	
Cloud Virtual Machine	
Hardware	Software
Processor: Intel Xeon ES-2620 V2 @ 2.6 Ghz Volatile Memory: 64 GB DDRAM Non-volatile memory: 1 TB HDD Communication interfaces: USB, Ethernet	Operating System: Debian 9.2 Software Packages: Docker, docker-compose, Grafana

Table 3. List of devices for the demonstration of UC1 in IT-1

The devices listed above are grouped into three categories:

1. LoadSensing's group
    a. Inclinometer (IoT), integrating a datalogger, called tiltmeter
    b. LoRa Gateway (IoT)
2. Jammer Detector's group:
    a. Odroid C2 (IoT)
    b. Jammer Detector's HackRf (IoT)
3. Actuator group
    a. L2- group-VMs connected to actuators and also used for computation purposes.
    b. L1- group PCs acting as leaders
    c. WOS Leader PC-1
    d. Leader PC-2
    e. Cloud Server
    f. (L0: Cloud).

They are coordinated and work together in the following way:

- When (L1 Fog) WOS Leader Monitoring Software, through a (IoT) tiltmeter + Gateway, detects a sensor-packet timeout error, the (IoT) Jammer Detector is started, to check whether a jamming DoS (Denial of Service) attack is being performed. It will report its Jamming data to the (L0 Cloud) Cloud software
- When (L1 Fog) WOS Leader Monitoring Software, through a (IoT) tiltmeter + Gateway, detects an inclination value that exceeds a safe threshold, it will start the siren alarm and trigger the emergency service

- The emergency service triggered by the detection of the excess of inclination in fact are three different sub-services:
    - An ambulance is searched
    - A fire engine is searched
    - The best paths are computed, from the location of the ambulance and the fire engine to the building where the inclination is detected
    - Traffic lights are changed from red to green (and blue indicating an emergency) in the computed paths
    - The ambulance and the fire engine move automatically to the building through the computed paths.

4.1.2. Current Software Responsibilities

There are software functionalities in all the layers of the infrastructure, as will be explained below. The functionalities of this Cloud and Fog infrastructure will be extended to execute other tasks that will perform all of the use case scenario steps. Current and new functionalities are described below.

The cloud software provides monitoring. It receives data from the dumb sensors, for example the tiltmeter, through the LoadSensing datalogger, via the LoRa Gateway. Subsequently, if the thresholds are exceeded, the software launches an alert and starts the emergency service system. The sensor timeouts are checked in the leader of this area in order to confirm that there is no service loss. In this case, the emergency service (three applications) is called and the visualisation software will show any jamming alert attack that has happened. It shows the sensor measurements in real time. Finally, the infrastructures alarms are shown.

The area 2 leader hosts the monitoring software which performs a set of functions. It:

- Receives through the Gateway the data from the dumb sensors such as the tiltmeter through LoadSensing's datalogger, via the LoRa Gateway
- Checks the sensor threshold and, if it is reached, launches an alert and starts the emergency service systems
- Detect if any sensor is having a service loss through its timeout status
- Launches the emergency services calling applications from UPC.

The role of the LoadSensing Gateway is threefold. It interacts with the jammer detector if the tiltmeter goes on timeout status in order to launch it from the gateway to check whether a DoS jamming attack is being performed. It also forwards the sensor data to the Cloud. Finally, it turns on the physical alarms (siren) when there is an emergency.
The LoadSensing Datalogger Firmware is responsible for reading LoadSensing tiltmeter (inclinometer) values and transmit them to the LoRa Gateway wirelessly.

The responsibility of the jammer detector controller software is to execute the detection of jamming attacks using a Software Defined Radio, and communicate the positives to the Gateway via Ethernet, using JSON format messages.

Area 1 puts together one leader and two virtual machines serving as agents (Agent1 and Agent 2). These devices receive the requests from the monitoring software to execute the emergency resolution service. The three agents in area 1 (Leader, Agent 1 and Agent 2), execute the software that performs the functionalities corresponding to the emergency resolution service, consisting in three applications, each one performing a specific scenario:

1. Compute the paths for the vehicles from the current location to the specific destination (collapsing building)
2. Send to the vehicles the path to follow
3. Calculate the state of the traffic lights in the computed paths and send the required actions to the involved traffic lights.

Note: Both agents in layer fog 2 and leader have the same software, allowing to allocate the different applications (corresponding to functionalities 1,2, and 3) in the different agents.

Regarding the software installed in the IoT devices in area 1, the software embedded on the ambulance drives and controls the vehicle so it reaches out the specified destination. Similarly, the fire engine runs a software that drives and controls the vehicle so it reaches out the specified destination. Finally, the traffic light is also equipped with a software that changes its status to set it to green so as to allow the vehicles to go through. This clears the path according to the signal sent by the emergency resolution service.

4.1.3. Tasks to be executed by the mF2C Agent

In IT-1, we can see how the mF2C Agents allow having a backup instance of the emergency service Monitoring Software, which is critical for the whole solution. This Monitoring Software, which is meant to run on the Cloud in order to allow having less expensive hardware as Gateways, may need to be enhanced by running either on a separate cloud instance or, in this case, on the Fog.

4.2. First Iteration Use Case (IT-1)

The use case components for the Demo V1 are summarized in figure 9.

This includes the F2C components:

L0: Cloud agent
L1: Leader agents
L2: Regular agents

And the Edge components:

IoT sensors, IoT Gateway and IoT actuators.

4.2.1 Current implementation of the IT-1 components

IoT	Loadsensing Tiltmeter + Inclinometer (SCA103T) + FreeRTOS + Loadsensing application	Ready
	Loadsensing's Kerlink Gateway + Standard Long-Term Support Linux version 3.10 + LS Applications	Ready
	Receive LoadSensing's sensing data through LoRa and forward to Monitoring Software	Ready
	Jammer Detector: ODroid C2 + Ubuntu minimal LTS + Hack RF One + GNU Radio + dependences + Worldsensing's SDRJD software	Ready
	Detect Jammer attacks	Ready
	Transmit jammer detection JSON decisions to the Gateway	Ready
	Physical Alarm (siren) activation	Ready
	Ambulance/Fire engine: Receive the input (path) from the emergency service.	Ready

	Ambulance/Fire Engine: Control the car to follow the specified path	Ready
	Traffic light: Receive the input (action) from the emergency service	Ready
	Traffic light: Change the state of the traffic light	Ready
Fog	Laptop HP + Ubuntu + Monitoring Software	Ready
	Check sensor thresholds	Ready
	Check sensor timeouts	Ready
	Interaction with Gateway on alert cases	Ready
	Compute the paths for the vehicles from the current location to the specific destination (collapsing building).	Ready
	Send to the vehicles the path to follow.	Ready
	Calculate the state of the traffic lights in the computed paths and send the required actions to the involved traffic lights.	Ready
	mF2C Agent integration	Ready
Cloud	Rackspace Cloud Virtual Machine + Ubuntu Server LTS + Monitoring Software + Historical data + Visualization	Ready
	Check sensor thresholds	Ready
	Check sensor timeouts	Ready
	Interaction with Gateway on alert cases	Ready
	Show sensor measurements presented in real time	Ready
	Show infrastructure alerts	Ready
	Show jamming attacks alerts	Ready
	mF2C Agent integration	Ready

Table 4. Current implementation of the IT-1 components

For the next iteration, the use case will be enriched so that the solution supports tracking of assets, smartphone alert messaging and integration of the Jammer Detector device to the mF2C ecosystem.

The tracking of assets is an important part of the Emergency Management on Infrastructure Construction solution, as it allows logistics improvements regarding workers and machinery. Furthermore, it improves safety in case of a critical situation as it helps to geolocate people at risk. In the case of the smartphone alert integration, this will also enhance the security of the solution and of infrastructure construction workers that should be warned about alerts and risky situations. Finally, the integration of the mF2C agent to the Jammer Detector device will add a relatively powerful fog device to the ecosystem. This smart sensor is most of the time idle (only activated when sensors timeout), which offers a set of useful resources to be exploited by the rest of the solution.

4.3. Data Flow Diagrams

It is important to understand the current dataflow of a regular installation without mF2C. In this case, if we needed to improve the solution's reliability and QoS, we would require either a more expensive Gateway or a backup cloud (or on-site) instance of the software running the Monitoring Services. Having the mF2C ecosystem available to run those services will give us the flexibility to save resources while having a more robust solution.

In the following section, the data flows from the sensors up to the Monitoring Software will be shown. We will firstly describe the current standard case, without mF2C, and later contrast it with the mF2C architecture deployed in order to see the different behaviour. The reader will be able to see the backup flow of data, which improves response time when running on the Fog and also provides more reliability in case one of the data flows fail. Later, in the results subsection, we will show the impact of this different behaviour on the results and KPIs.

4.3.1. IoT sensors to Monitoring Software (no mF2C)

The following scenarios occur when mF2C is not implemented and the Monitoring Software runs on the Cloud, which is the typical case for these types of solutions.

Normal operation:

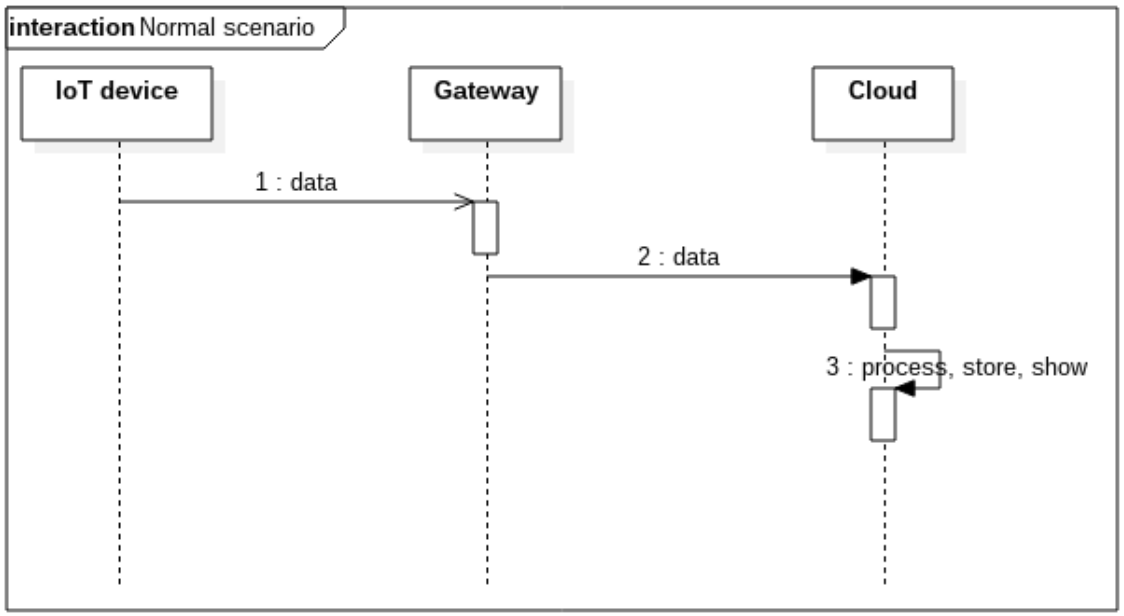

Figure 10 UC1 normal scenario workflow

In the workflow depicted in figure 10, the sensor device sends data to the Gateway. As previously described, this will be performed through the LoadSensing and LoRa, using JSON format. From the Gateway, the data will be sent to the Monitoring Software running (normally) on the Cloud and, if the value is within the thresholds, will be stored and normally visualized.

Inclination threshold alarm situation

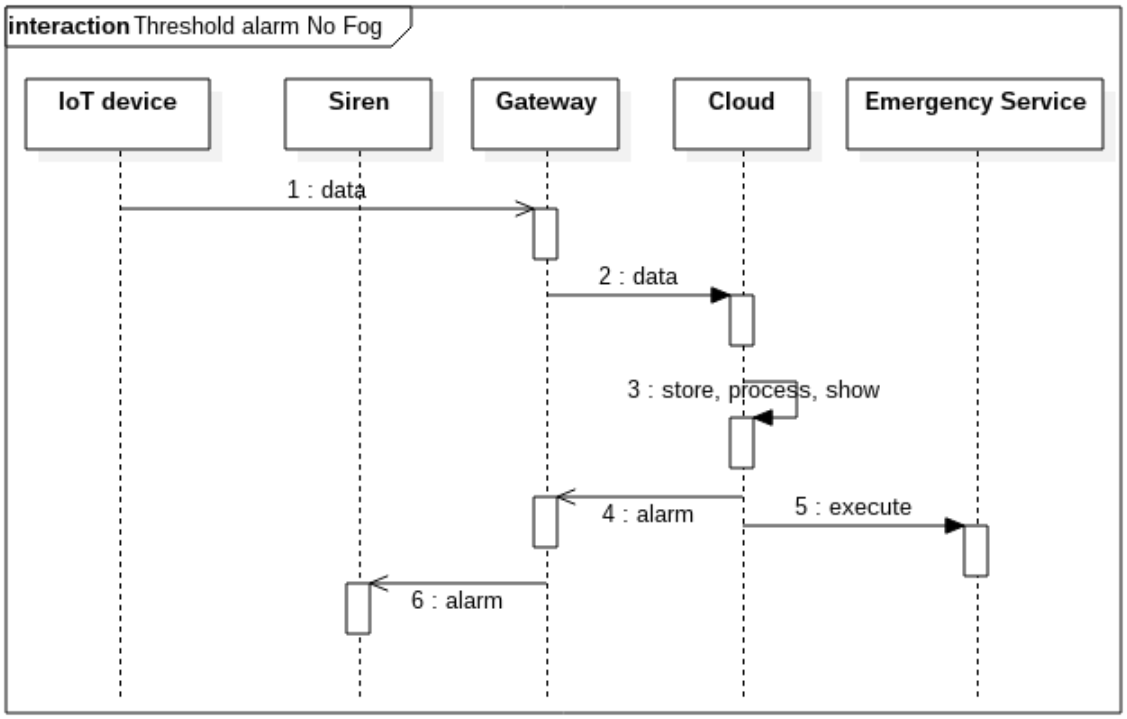

Figure 11 UC1 threshold alarm with no fog workflow

As an extension of the previous case, if an alarm situation is detected, the alarms need to be activated on site (see Figure 11). In the case where an inclination value exceeds the tolerable thresholds, the Monitoring Software will contact the Gateway to activate those alarms.

Jammer detection alarm situation

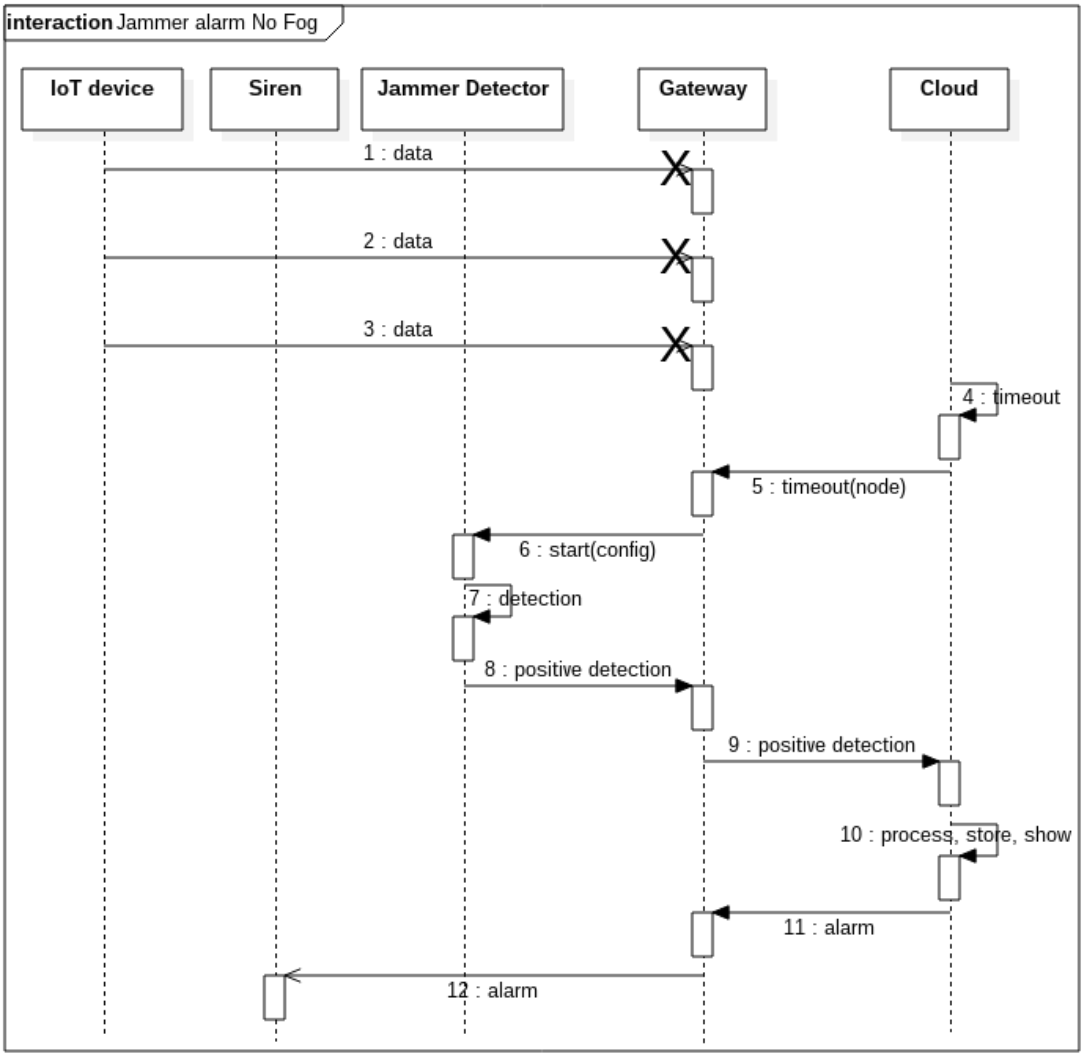

Figure 12 UC1 Jammer alarm without fog workflow

In the case presented in figure 12 where the data from one of the tiltmeters is not received after a certain time, the Monitoring Software will timeout and communicate this to the Gateway. The latter will then start the Jammer Detector with the proper configuration. The detection device, in this case, will find a threat and communicate this to the Monitoring Software, which will post-process it in order to decide whether there is a jammer present and, if this is the case, contact the Gateway to start the alarm.

4.3.2. IoT sensors to Monitoring Software (with mF2C)

The following scenarios (Figure 13) take place when the mF2C is available and the Monitoring Software is used not only on the Cloud but also as a backup on any Fog device that is capable of running it.

Normal operation

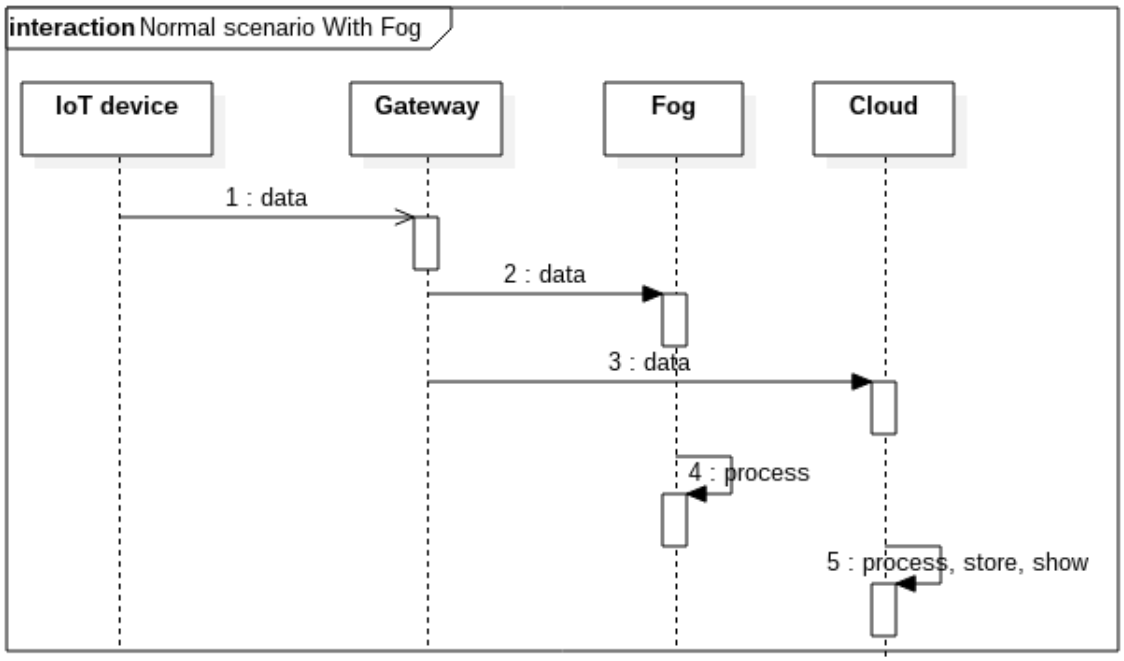

Figure 13 UC1 normal scenario with fog workflow

The difference with the previous normal scenario is that the data is transferred to both Fog and Cloud. The Monitoring Software will process the data on both. In case one of them is down or unreachable, then the other will act as a backup.

Inclination threshold alarm situation

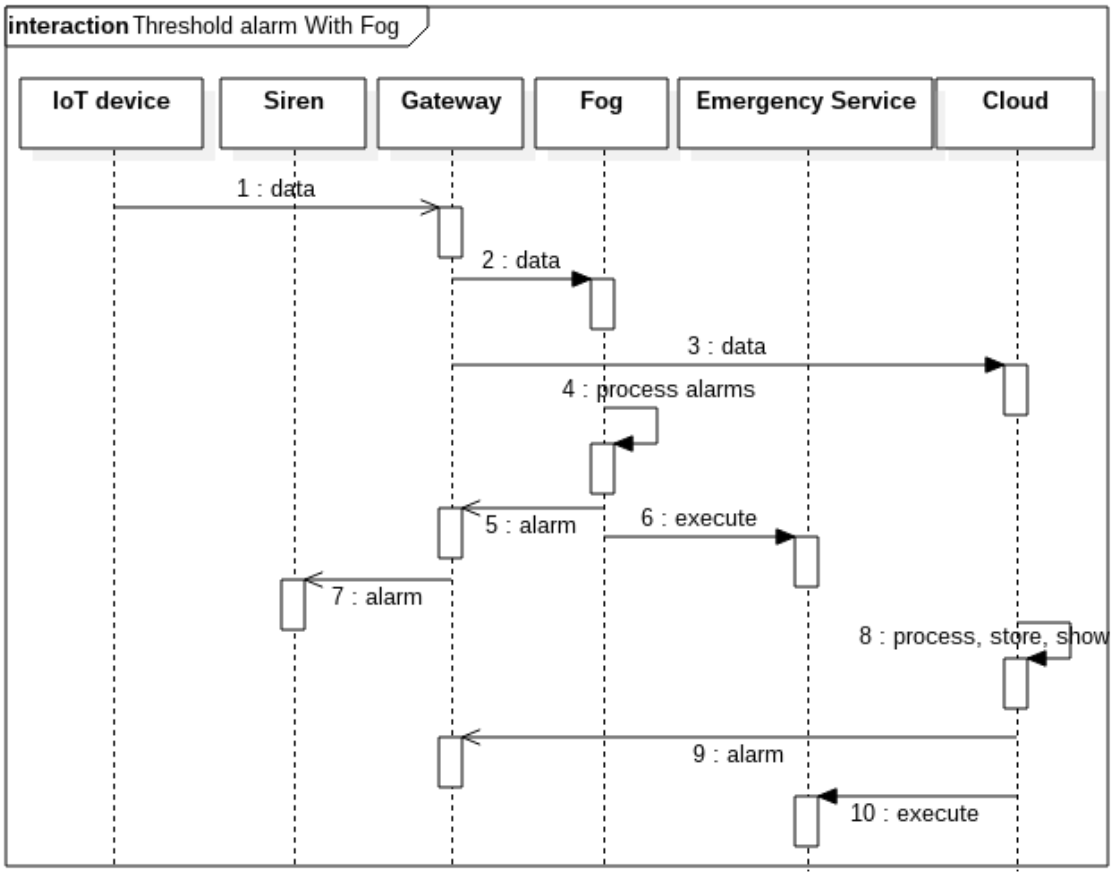

Figure 14 UC1 threshold alarm with fog workflow

Again, the case presented in Figure 14 is similar to that of the case with the threshold alarms but it includes the same processing on the cloud and at the Fog layer. This will provide a faster reaction to the alert situation than when it is processed on the Fog. In case one of them is down, the other can take its place.

Jammer detection alarm situation

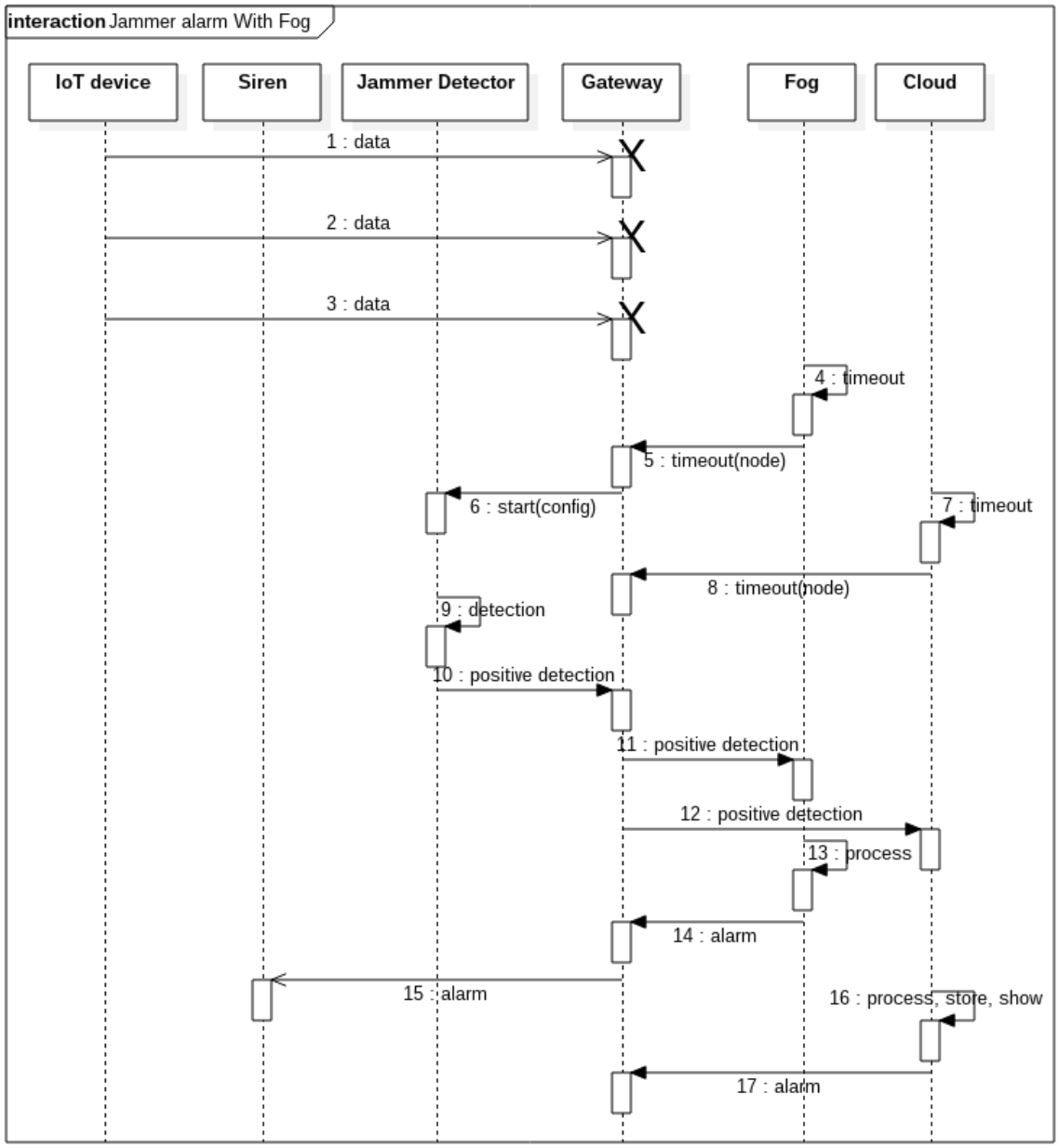

Figure 15 UC1 jammer alarm with fog workflow

Figure 15 presents the Monitoring Software running both on the Fog and the Cloud, that will perform the corresponding alerting actions. This renders the solution more reliable and reactive.

4.3.3. Emergency Resolution Service

In this case, illustrated in Figure 16, there is no difference in the workflow when it runs with and without mF2C, the only difference is where the subservices composing the service are allocated and executed. In the case of using mF2C, they are allocated in one or different mF2C agents, and without mF2C they would be allocated in the cloud or other devices with the subservice installed.

Emergency sub-services resolution

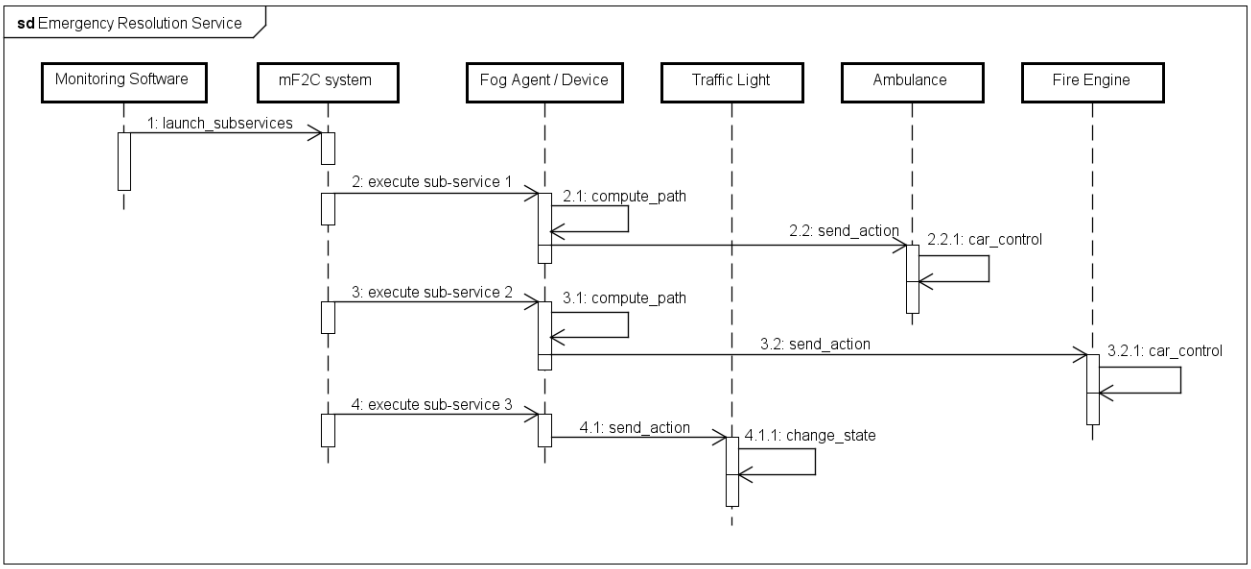

Figure 16 UC1 emergency resolution service workflow

The monitoring software sends the request to execute each one of the subservices to the Lifecycle (mF2C system), the Lifecycle looks for suitable agents in which to perform the requested subservices, then allocates and launches them in the selected agents (being agents in L2 or leader in L1).

4.4. Experimental set-up

The use case software was adapted to be integrated with the mF2C service management. In this case, the agent is capable of launching the Monitoring Software Service on the Fog and the Cloud. In addition, the software was modified to call the Agent whenever an emergency service has to be started. This provides the flexibility to run those services on different agents and benefit from the distributed platform.

The whole sequence involves the Agents starting up, running the Monitoring Software Services on the Fog and the Cloud. When the tiltmeter's inclination is more than the established threshold, the Monitoring Software contacts the Gateway to start the siren and calls the Agent to start the three different emergency services (ambulance, fire engine and traffic light). These services will be launched on different Agent machines and will calculate the paths for the vehicles and start them, as well as contacting the traffic lights device to change them. Figure 17 below represents two parts of the testbed deployed.

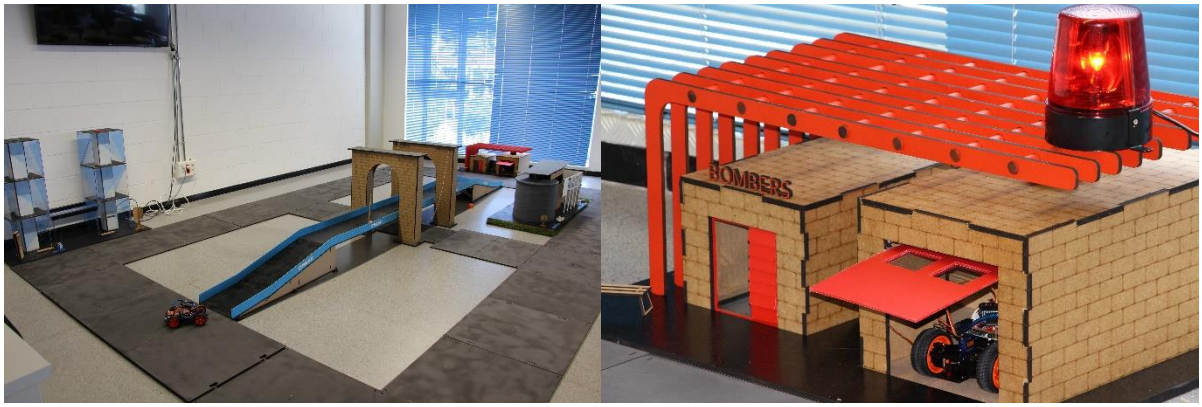

Figure 17 Photos of the testbed deployed at UPC

4.5. Results

The deployment of the mF2C architecture in the Emergency Situation Management system in a Smart City scenario provides better reliability and QoS, as well as improving the latency of a response to an alert situation when the software runs on the Fog. This will be explained more in detail in the following section, with KPI measurements. Moreover, thanks to the intrinsic redundancy provided by the mF2C architecture, the number of devices installed can be reduced without lowering the quality of the service proposed, so the cost can be reduced, mainly for the hardware, but also for the service as hosting necessity on the cloud is also reduced. Once the final architecture of the mF2C project is installed, a detailed study of the OPEX reduction will be performed, during IT-2, and will be presented in the follow-up deliverable to this one. In addition, the time response to emergency situations will improve if the services are run locally, rather than on the cloud, and the emergency services will be able to intervene faster.

4.6. KPI measurement

As mentioned, latency can be improved significantly by running the Monitoring Software on the Fog, as the local network will introduce less delays than the Cloud itself. In order to demonstrate this, the time it takes for the physical alarm to be started was measured from the moment that the tiltmeter measurement reached the Gateway, as explained in Figure 18. The previous step is identical for both architectures. The following figure 18 explains the sequence and the delays measured.

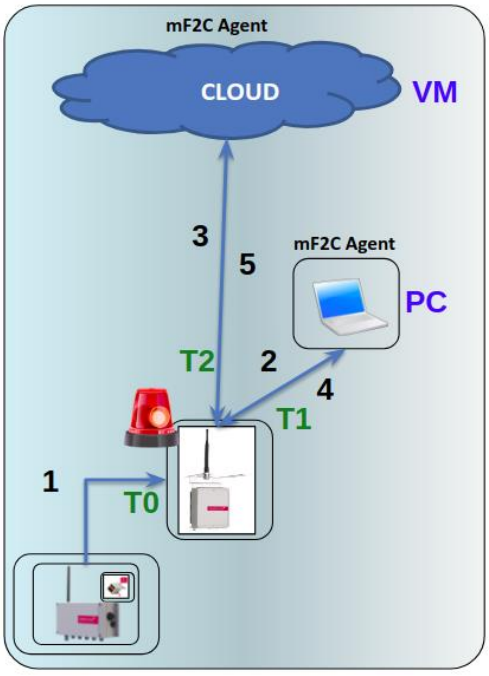

Figure 18 UC1 KPI measurements experiment

Firstly, the tiltmeter sends message 1 to the Gateway. The latter processes it and forwards it (message 2) to the mF2C agent running on the Monitoring Software on the Fog (local network). At the same time, the same message (number 3) is sent to the Cloud (Rackspace Cloud provider, UK) and the Monitoring Software, both running an mF2C agent. Both will enter these messages to the system and detect the critical value, thus the Fog and Cloud agent machines will send messages 4 and 5 respectively to the Gateway in order to start the siren. T0 is the time when message 1 arrives to the Gateway. T1 and T2 are the times when the siren is about to be started, due to the arrival of messages 4 and 5 to the Gateway, respectively.

We have calculated Tx = T1-T0 and Ty = T2-T0. These measurements have been performed various times and the latency was found to be 73.75ms while running the Monitoring Software in the Fog (local network), while the average on the Cloud is 97ms, as presented in Table 6. These results demonstrate an improvement of approximately 24% when running the solution on the Fog, instead of the Cloud.

Implementation	
Fog (ms)	73.75
Cloud (ms)	97
Percentage (%)	24

Table 5. Delay in use case 1

It is important to mention that the transmission time between the tiltmeter and the Gateway was not taken into account for these experiments. This is because it is highly dependent on the sampling rate that is configured on the datalogger, which for these types of solutions that require real time reaction, should be 1 or 2 seconds. Moreover, these latencies are independent of the use of mF2C architecture. Also, it is important to stress that although these results fall short of the 30% expected at the end of the project, these results have been obtained before the optimisation that will be performed during IT-2 so it is expected that an improvement of 30% will be demonstrated before the end of the project.

Secondly, two other KPI of huge commercial importance for the company are the reliability and the quality of service. Ensuring a high QoS is very important for critical infrastructure monitoring and to be competitive in this market. On the other hand, a catastrophe can be caused if the QoS contracted is not complied with, as well as huge losses which could be fatal for the company. In this case, the fact that mF2C aims to create a dense environment of agents willing to share their resources, allows us to run our Monitoring Software on different devices, as well as ensure a backup of the original solution which would run only on the Cloud. We could even run more than one instance on the Fog, if possible, to ensure that the QoS is almost 100%.

Assuming that there is a high number of mF2C capable devices, which would be the case in an emergency situation in a smart city, having at least one Monitoring Software instance running on the Fog as a backup of the one running on the Cloud, we can say that the Monitoring Service will be up and running close to 100% of the time, increasing strongly the reliability of the service. In order to be more realistic, we will consider a quality of service of 99 %. On the other hand, if mF2C was not being used and the solution was intended to be only run on the cloud, then we should consider that the QoS of the Internet connections to the Cloud is not 100%. For instance, considering one of our providers, Vodafone, which is one of the most important Internet service providers in Spain, offers compensation of the monthly bill if the service is down more than 48h during the month (http://www.vodafone.es/c/statics/pdf-calidad-del-servicio-conocenos/). This means that the QoS they provide on availability is lower than 93.33%. If we consider that we can bring that availability up to 99% with the use of the mF2C environment to run at least one backup instance of the Monitoring Software, we could improve our commercial offer by offering a QoS improved by approximately 7%.

4.7. Business Prospective

The key aspect in the field of Emergency Situation Management services are the service reliability and the quality of service. Indeed, a valid commercial solution needs an availability as close as possible to 100% in order to be successful, as clients want to be able to count on their provider and on the solutions contracted. For this reason, the mF2C architecture is a great improvement for Worldsensing's solution as it allows intrinsic redundancy to the system. The current solution is being performed on the cloud and is the connection to the cloud is unavailable, through a failure from the network, because of interferences, or in the case of a multitude saturating the network, the solution proposed in use case 1 might not be capable of delivering its service. For the same reason, we are not

aware of any commercial competing solution currently on the market. Being able to provide a QoS of 99% or more would provide Worldsensing with an unfair advantage to capture most of the existing market.

As demonstrated by real-life experiments at the UPC's testbed, the latency is reduced by 24% in IT-1, in which the system has not yet been optimised. This already provides an important commercial advantage which might be improved during IT-2.

Finally, the mF2C provides an inherent redundancy to the system. In order to guarantee a correct service, any installation would require both physical and digital redundancy, in order to reduce issues. This would increase the cost of deployment, both in hardware and in Cloud fees. The mF2C architecture provides increase calculation capabilities as well as reduce the need for physical redundancy, offering an important cost reduction. As the architecture is not yet optimised, the cost comparison has not been performed but it will be presented in IT-2, once the final architecture is defined, and the OPEX (operational costs) are expected to be decreased by more than 10%.

5. Use Case #2: Smart Boat Service (SBS)

5.1. Complete Architecture Use Case

The Smart Boat service use case translates the Cloud and Fog concept to the marine environment where IoT-enabled vessels are anchored, tied in dock or sailing. The nature of the use case brings more dynamics into the Fog-to-Cloud concept, that need to be tackled with resilient and robust application. The basic plan is to provide the mF2C application to Boat Users that will be able to monitor their boat(s), execute control actions to a boat, and enrich the user experience on the boat. The basic scenario was already presented in deliverable *D4.1* [3] so it is not extensively described in the present document. The mentioned targeted scenarios that are among the goals until the end of the project are:

- Continuous Boat Monitoring
- Anomaly Detection
- Online Docking and Anchoring Reservation, and
- Data Plan Sharing (FEX - fair exchange).

Despite this core focused goals, the real plan evolves through the progress of the project, technology and the market. The mentioned scenarios are placed in the marine world and the complete use case environment from connectivity perspective is shown on Figure 20. Mainly all functionality of the boats and boat user's app are bound to one of the following five possible network *communication scenarios* (see Figure 19):

1. Boats have access to the Cloud through WiFi
2. Boat have access to the Cloud through 3G or 4G internet connection
3. Boat can send its data to the Cloud through LoRa communication with the boat, who can share the 3G or 4G network connection
4. The group of boats can exchange the data through LoRa and, if available, send to the boat that has 3/4G connection to the cloud. The boat provides a gateway between the group of boats and the cloud
5. The isolated group of boat can communicate with each other through LoRa, but without connection to the cloud.

Scenarios 1 and 2 have the best communication possibilities for connecting the boat to the rest of Fog and Cloud, while other scenarios use special protocols to reach the rest of the network (see communication scenarios 3 or 4) or the communication is limited to the current fog/mesh network (communication scenario 5). In the current solution demonstration, the main focus is on scenario 1 and proof of concept of scenario 3 (LoRa communication). This starting point was chosen to easily apply the lessons learnt to new scenarios.

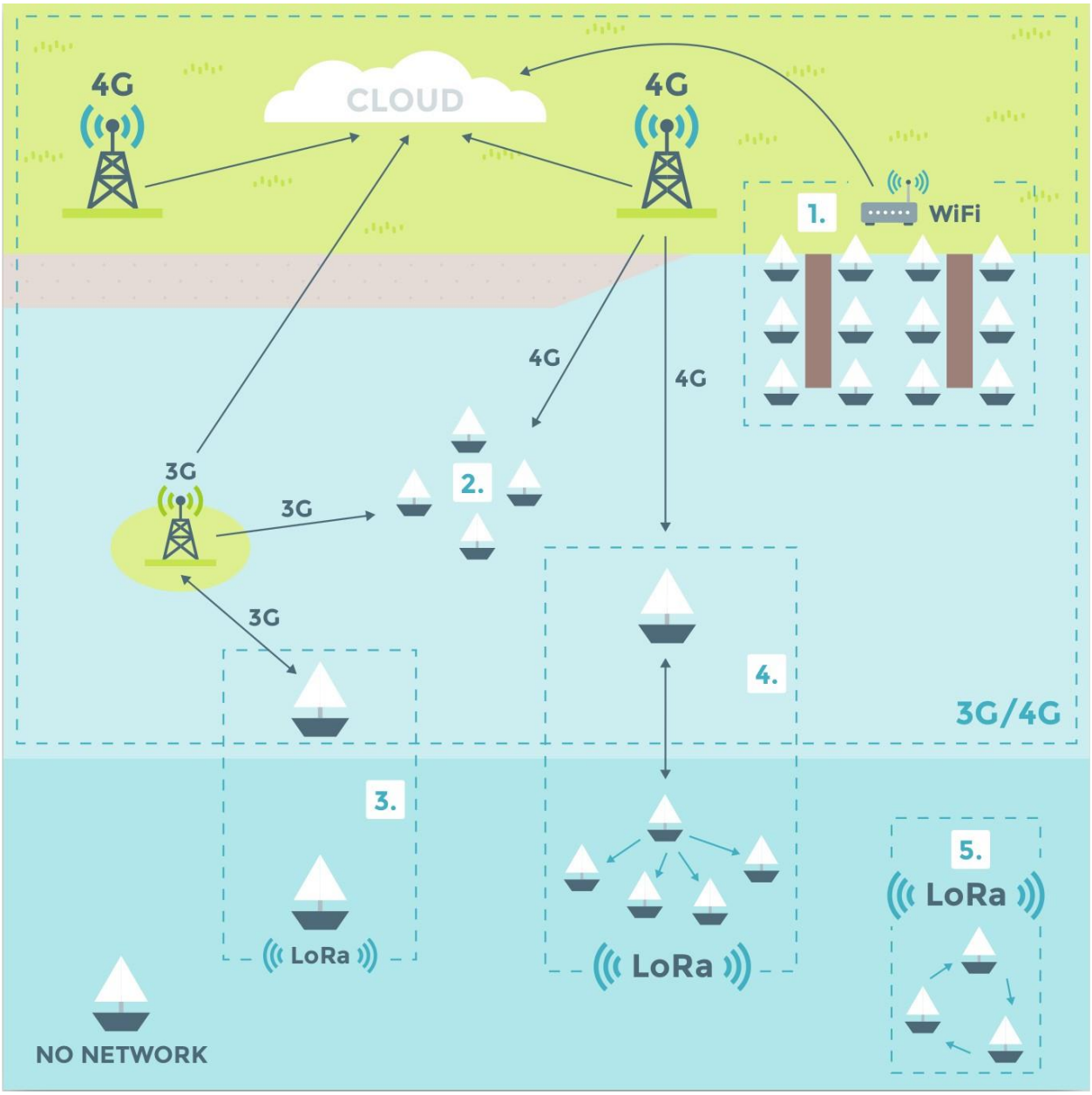

Figure 19 Smart Boat - Network availability scenarios

5.1.1. Current Hardware

From the mF2C perspective, a part of the Smart Boat application presents a top-cloud layer and a core backend with total overview but not always up-to-date information due to the absence of communication paths between devices and cloud. All boats on Figure 19 have Smart Boat devices and mF2C deployed to some extent, tailored by the user who decides which sensors and functionality should be deployed on the specific boat. The Fog and IoT devices used in our solution demonstration are presented in detail in Table 6.

Edge IoT devices	
Sentinel Boat Monitor	
Hardware	Software
- 32bit microprocessor - Dimensions : 100 mm x 100 mm x 40 mm	Proprietary Sentinel software, a special SDK for development

- Weight: 500 g
- Voltage range: 11V to 30 V DC
- Low current consumption:
  - Maximum: 300 mA (internal battery charging)
  - Nominal average: 35 mA RMS (normal operation)
  - Sleep mode: <1 mA
- GPRS modem: Works worldwide Quad-Band
- Bluetooth LE support
- NMEA2000 support
	• Measures up to 4 batteries
	• Monitors the bilge pump
	• Real-time GPS tracking and journey log
	• Geofence alerts for unanticipated movement
	• NMEA2000 compatibility
	• Supports multiple sensors

Sentinel Hub	
Hardware	Software
• Two IO pins • One shunt resistor input • External power supply pin • Humidity/Temperature sensor • Powered by 2xAA batteries • 2 years operation on batteries • Bluetooth LE support	
Fog-capable devices	
RaspberryPI 3	
Hardware	Software
(Smart Boat device) • 1.2 GHz Quad Core ARM Cortex A53 • RAM: 1 GB LPDDR2 memory • Bluetooth 4 LE • WiFi • Ethernet	Drivers and Smart Boat application modules that provide connectivity to the Sentinel Boat Monitor through Bluetooth.
LoRa Module (for RaspberryPI)	
Hardware	Software
• Output Power (dBm): 14.00 • Host Interface: UART • Operating Temperature Range: -40 to 85 • Frequency Range: 434, 868 MHz • Input Sensitivity (mVpp): -148 • Rx Input Sensitivity (dB): -148 • TX Current Consumption: 40 mA (14dBm, 868MHz) • RX Current Consumption: 14.2 mA	• On-board LoRaWAN™ Class A protocol stack • Device Firmware Upgrade (DFU) over UART • 14 GPIO for control, status, and ADC • RoHS compliant • European R&TTE Directive Assessed Radio Module
Sentinel router	
Hardware	Software
- 3G/4G	• SMS Control

<table>
<tr><td>- WiFi
- Power supply:9 - 30 VDC
- 4 Ethernet ports
- Dual SIM</td><td colspan="2">• WiFi Hotspot
• Mobile on Demand
• Backup WAN
• Status, configuration via SMS
• SIM card switch controlled by signal, data limit, roaming</td></tr>
<tr><td colspan="3">Laptop</td></tr>
<tr><td>Hardware</td><td colspan="2">Software</td></tr>
<tr><td>Intel NUC / Laptop</td><td colspan="2">mF2C Agent deployment</td></tr>
<tr><td colspan="3">L0 Cloud-capable devices</td></tr>
<tr><td colspan="3">Cloud Virtual Machine</td></tr>
<tr><td>(Virtual) Hardware</td><td colspan="2">Software</td></tr>
<tr><td>OpenStack
• Detailed definition will be described for iteration 2.</td><td colspan="2">Not included in IT1.</td></tr>
<tr><td colspan="3">Power</td></tr>
<tr><td colspan="3">The AC outlet
Power bank
• UPS-like 100% uptime switching
• 2.1 A + 2.4 A downstream port
• Max 4.5 A output @ 5 V
• 2 A upstream charging port
• 16750 mAh capacity @ 5 V
5 V -> 12 V external step-up converter</td></tr>
</table>

Table 6. Current hardware for Smart Boat use case

The hardware/devices presented in table 6 is usually mounted on boat as suggested on Figure 20 and can be grouped into the following categories:

- Boat sensors and actuators:
  - Sentinel Boat monitor (Figure 21, element 3)
  - Sentinel Hub (Figure 21, component 4 and 5).
- Smart Boat device:
  - RaspberryPI, processing unit (Figure 21, component 1).
- PCs acting as leaders (only in the iteration 1, additional power to RaspberryPI):
  - Intel NUC or
  - Laptop.
- Network/Communication:
  - Sentinel Router (Figure 21, component 2)
  - LoRa (development board on RaspberryPI, Figure 20, integrated on component 1).
- Cloud Server
  - Private/public cloud, not included in iteration 1.

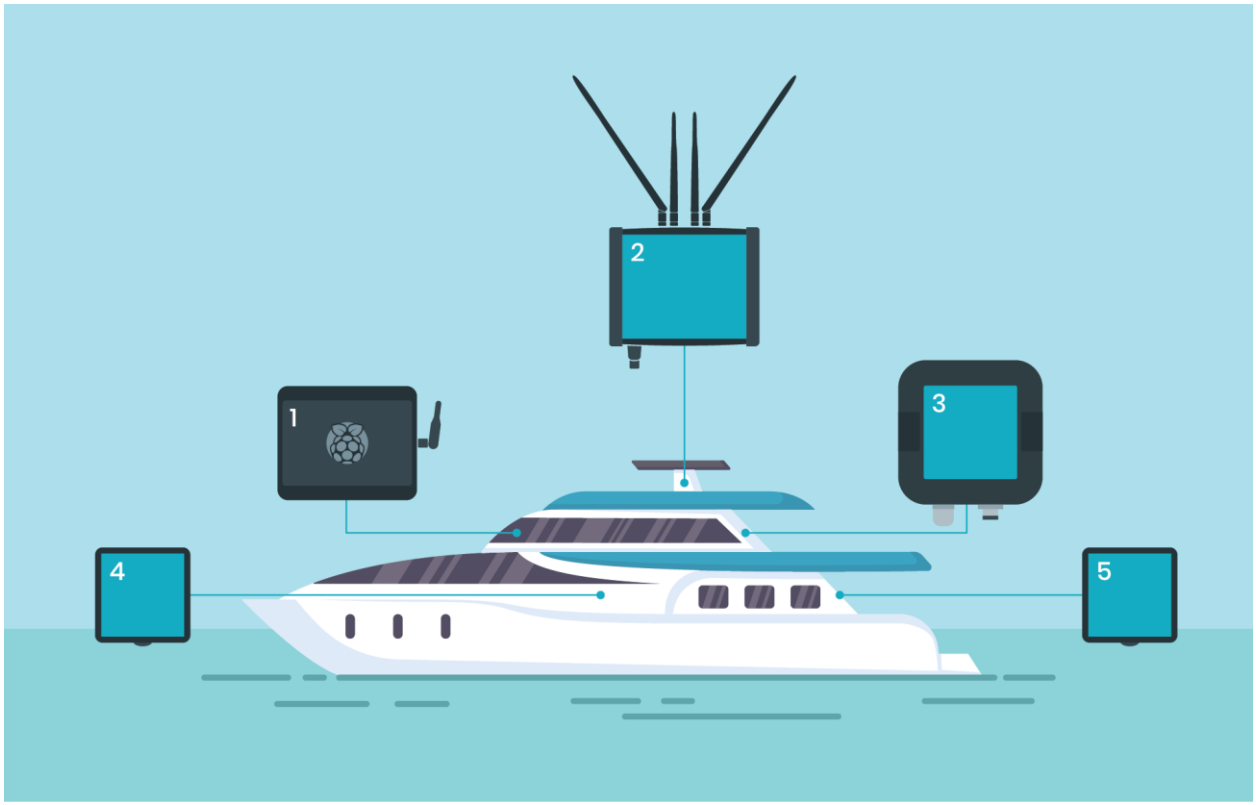

Figure 20 UC2 basic component overview

The software deployed onto the presented hardware components are presented in the next section.

5.1.2. Current software responsibilities

The core software components of the Smart Boat use case are logically divided into three execution environments, which is also shown in the Figure 21.

1. **RaspberryPI (Sensors and actuators)**: The first is the RaspberryPI environment which is used as a processing unit collecting the data from sensors and sending actions to the boat actuators. This component is very crucial to the Smart Boat use case as it acts as glue between the IoT and Edge/Cloud applications with transmission translating the data from the sensors to the application and providing ability for LoRa communication. Due to some specific hardware limitations, such as Bluetooth connectivity and LoRa development environments, this part of the software is bound to RaspberryPI.
2. **Main application**: The second component is the Main application which can run in a more hybrid environment, stretching from the Edge to the Cloud. These Main application components take care of storage, communication, task scheduling and processing capabilities. Here, the main logic and decisions are made where the actions will take place, for example on the Edge or in the Cloud.
3. **Client/Interfaces**: The last logical component is the user interface to the web and android applications. The backend of these components can be hosted on the Cloud and the Edge devices, while the graphical interface is displayed on the device owned by the user e.g. smartphone or computer.

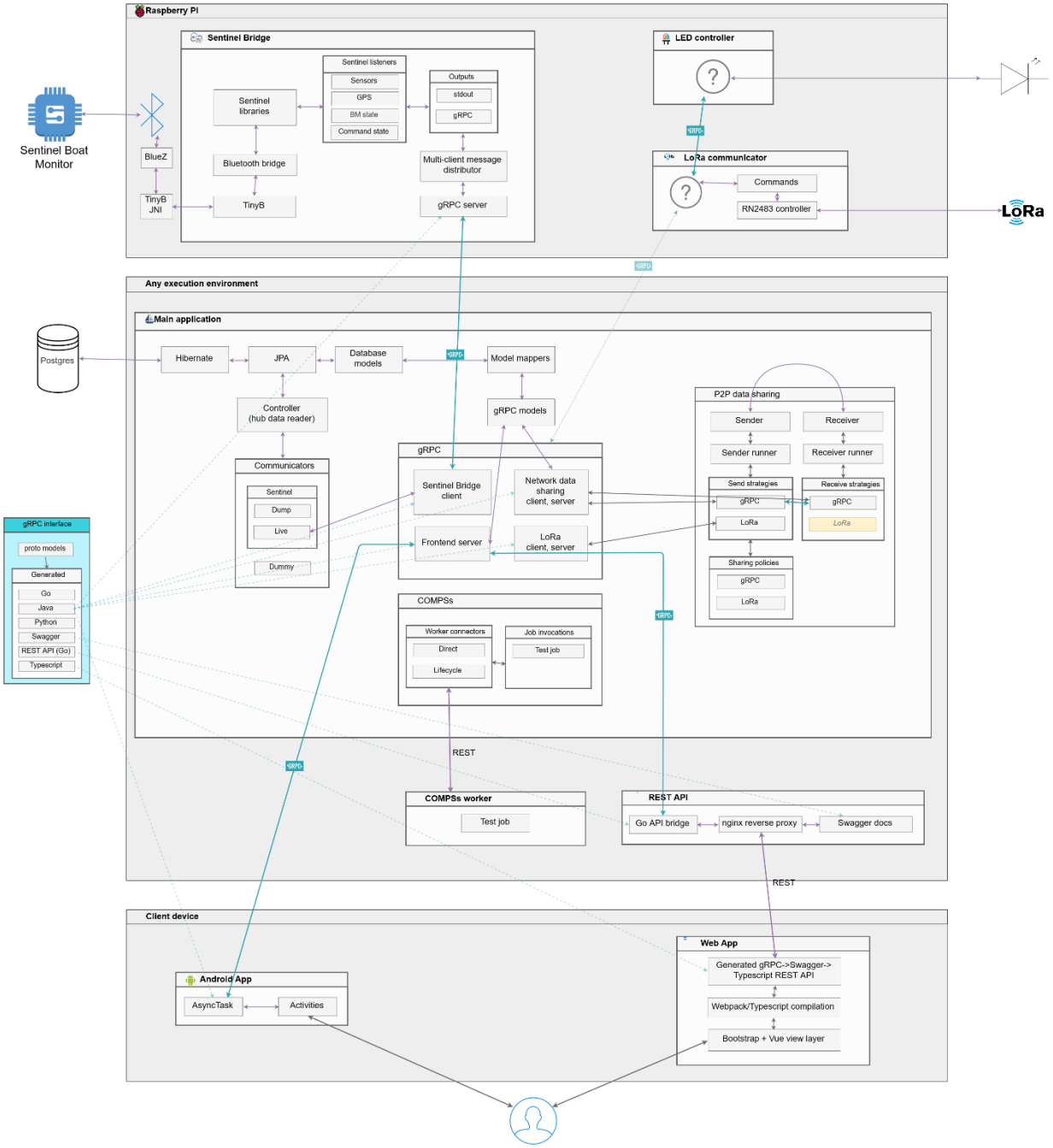

Figure 21 UC2 application - logical/software diagram

The detailed functionality of the software monitoring and controlling of the boat is presented in sections 5.3 and 5.4.

5.1.3. Tasks to be executed by the mF2C agent

In the IT-1, the Smart Boat deployment requires the mF2C Agent to deploy application modules and to execute computational task for data filtering and processing. The Smart Boat application requires pre-prepared data to be plotted on mobile devices and thus improve Android application responsiveness and user experience. To avoid transmitting all data points from Fog to mobile devices and plot unnecessary details or run additional calculations on mobile device for each sensor data view, the Fog level of Smart Boat application invokes DER (Distributed Execution Runtime) task on Agent to calculate appropriate histograms for plotting. The histograms are prepared continuously each hour

for different reviews like hourly, daily, weekly for each sensor and stored in Fog level. This optimisation will be used also in IT-2 where the filtering, processing and retrieving the data to Android device will be covered by Fog or Cloud, depending on the best approach for the current situation.

5.2. First Iteration Use Case (IT-1)

The IT-1 use case 2 deployment will be limited to the Fog and IoT layers. As the official Sentinel application is a product based on Cloud backend and Sentinel Boat Monitor devices, it seems reasonable that the development and deployment of mF2C for IT-1 start on the most separated parts: Fog and IoT. The main rationale is to unveil the least known traps of IoT and Fog on the Smart Boat part already in the IT-1, and then connect it to the Cloud during IT-2. The Cloud part of the Smart Boat application will maintain the same basic functionalities as the Fog part, only the computing and storing resources will be more powerful, reliable and persistent on the Cloud.

The IT-1 deployment of the UC2 is presented in Figure 22. In the lower two layers, two types of devices are located, namely sensors (commercial name *Sentinel Hub*), sensor hubs (*Sentinel Boat Monitor*) and user devices that can show the dashboard for interaction with the application, as computers and smartphones.

In the Fog Layer, there are two types of devices. The RapsberryPis, where part of the Smart Boat application is deployed, to maintain the communication with the sensors, and the PCs (Laptop or NUC) as the RaspberryPIs are not powerful enough to run the whole IT-1 mF2C agent.

The Cloud layer is not included in the current IT-1 deployment and will be integrated into the application in IT-2.

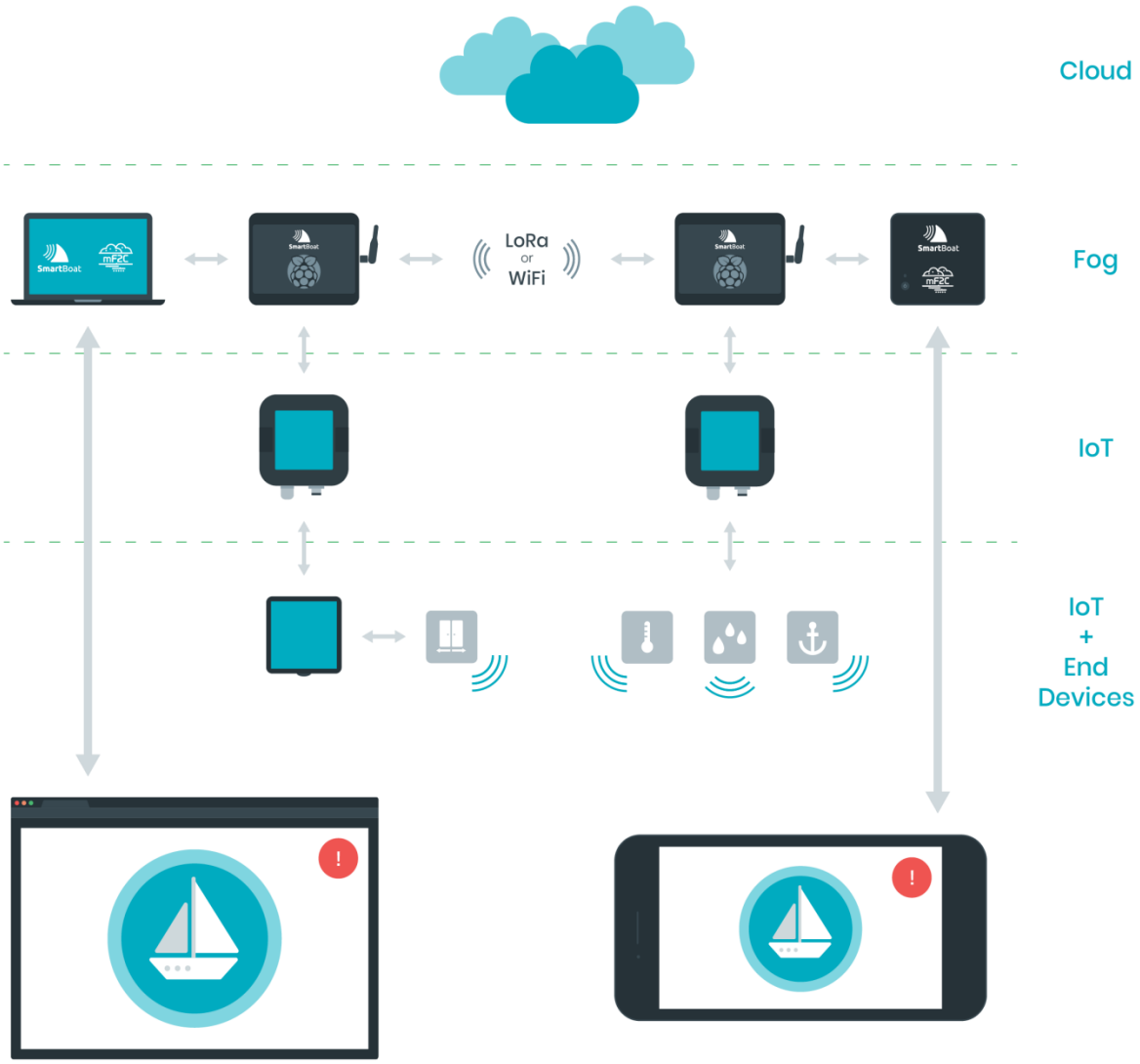

Figure 22 UC2 architecture in layers

5.3. Data Flow Diagrams

The Smart Boat use case interfaces with sensors and actuators deployed on the vessels and collects data from them. All data and information flows can be generalised and are presented in figure 23. All data flows are maintained or monitored by the main Smart Boat application. The main application connects together five logical components:

- **Storage**: component responsible for storing the data and making the data available for all application scenarios
- **User App**: component responsible for sending the data to the user devices, where user can interact with application, control the application and monitor the boat
- **Sensor Interface**: component responsible for the control of the data and actuating the sensors. This component integrates specific modules or drivers that allow the communication with the sensors
- **COMPSs jobs**: mF2C agent's component responsible to execute tasks submitted to COMPSs and retrieve the data
- **LED Control**: component responsible for the control of the LEDs and indication of the status reported by the main application.

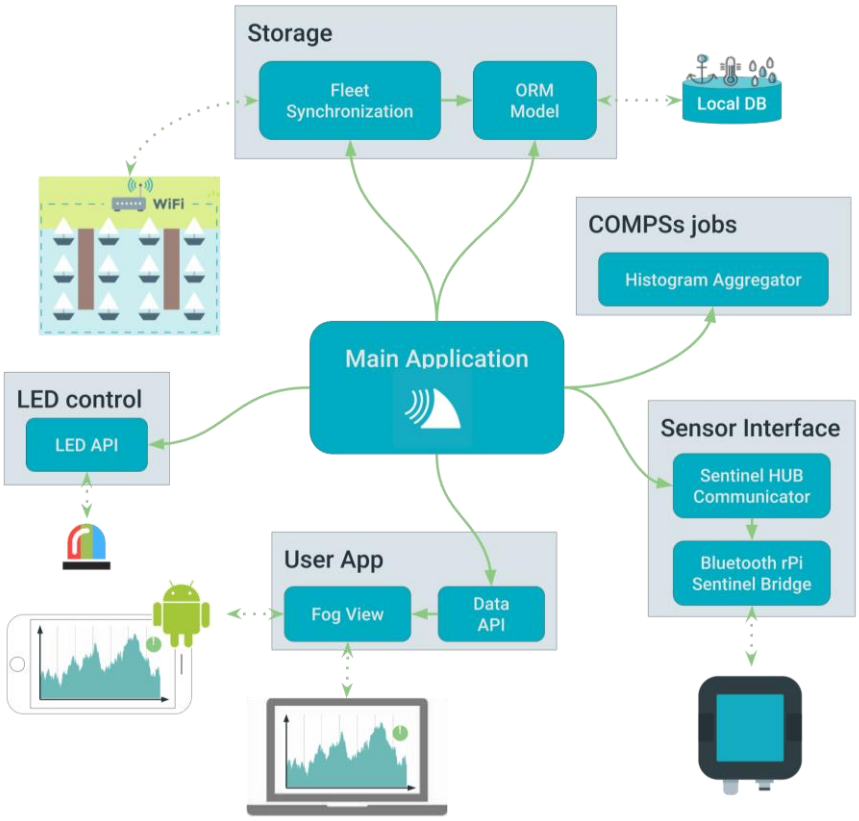

Figure 23 Data flow overview

Apart from the global data flow overview, the communication paths between components are detailed below with UML diagrams on Door Sensor event scenario and Sensor histogram preparation.

Door sensor event

Door sensor event (see Figure 24) has some preparation process that includes a new sensor listener or in other words, registers a sensor into a Smart Boat application. When the door sensor is registered and an action occurs, e.g. the door opens when nobody is on board, the notification is issued to the main application, which stores this state. Meanwhile, the main application sends an action to the LED control component, which displays the corresponding sequence on the LED lights mounted on the RaspberryPI. In parallel, the notification is sent to the user devices, and shown on the devices. The user then decides what to do or whether to disable the alert state.

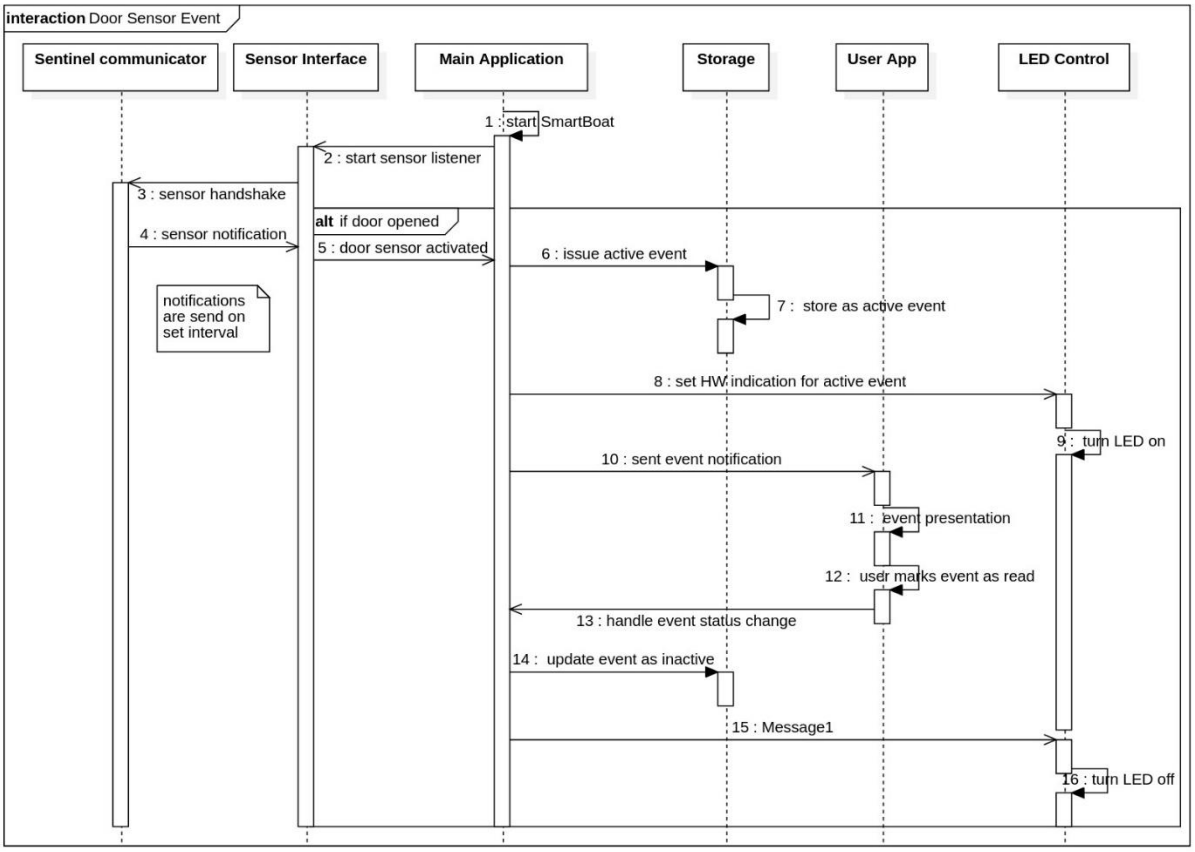

Figure 24 Door Sensor event data flow diagram

Sensor histogram preparation

The user dashboard shows graphs indicating the historical values acquired by the sensors over time. When larger intervals are reviewed, plotting all the values is not practical as this becomes too heavy for the device on which the application is running. Therefore, histograms calculated by averaging values over specific time ranges are pre-prepared in the main application and are sent to the user dashboard when requested.

The data flow for histogram preparation goes as follows (see figure 25):

- First, the main application should “register” the sensor
- Then the sensor is ready to periodically send the data to the main application through the Sensor interface module.

When the data is gathered, the request to perform the task that creates the histogram is issued to a mF2C agent COMPSs Job component. This component handles the calculation and returns the results to the main application. The calculations are stored and ready to be visualised by the user when convenient.

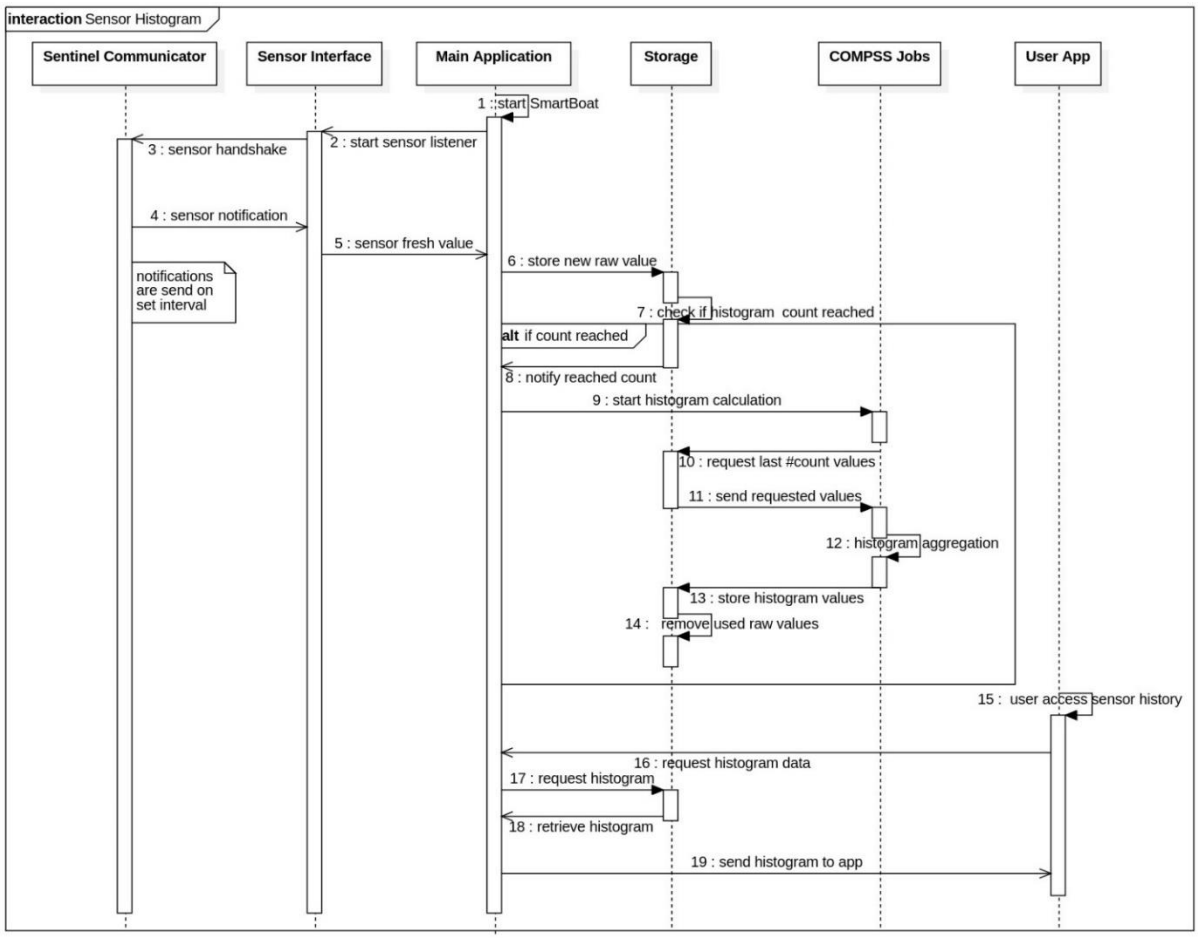

Figure 25 Sensor histogram preparation data flow diagram

5.4. Experimentations Set Up

Experimental setup includes all devices presented in Figure 22, which means that we have equipment for two boats (including mF2C agent, laptop and NUC part) required in IT-1. The boats are connected with Wi-Fi network and are collecting the boat vitals. The Sentinel application is deployed on the experimental set-up and collects the data from the sensors. The android application is able to connect to the application on the Agent and retrieve the data from the sensors of the specific boat. The owner of the boats can retrieve the data from both boats whilst the guest can retrieve the data only from a single boat – the scenario considers that the guest uses only one boat at a time.

The second experiment is a proof-of-concept of LoRa communication. For this experiment, two RaspberryPI devices equipped with LoRa modules and additional LED lights were used. The scenario demonstrates a transmission of a message queue from one device to another in Point-to-Point fashion. The length of the queue and current state of each device (sending and receiving) is indicated through the LED lights.

5.5. Results

The results from the Smart Boat application set-up demonstrate the reliability of the Fog part of the application allowed by the inclusion of the mF2C agent. The whole application is able to run

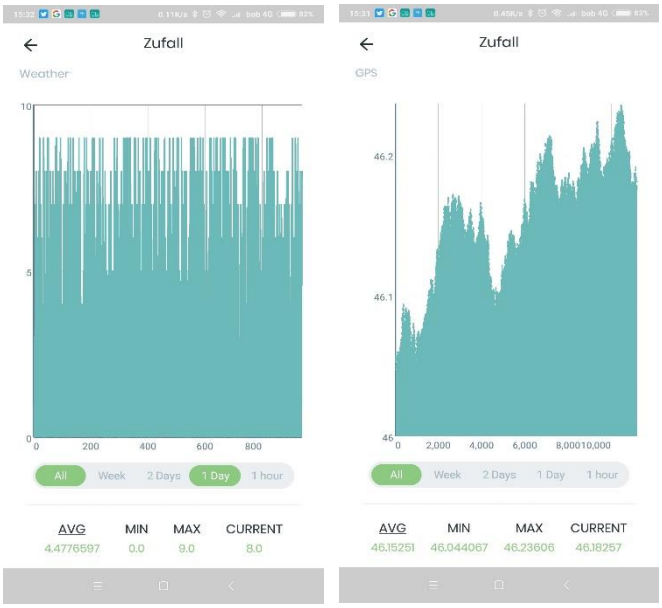

Figure 26 Smart Boat sensor screenshots

autonomously on a subset of devices and provide the basic Smart Boat monitoring and control functionality to the Android user. The application is able to limit visibility to the owner of guest profile and show detailed information about specific sensor, which is shown on Figure 26.

The second result is obtained from the LoRa proof-of-concept experiment in which we tried to send and receive messages. These results are important for theoretical calculation of the Smart Boat coverage area. The main improvement brought by the inclusion of the mF2C architecture in this scenario is the expansion of the coverage. The set of messages was successfully transmitted over a distance of 1 km, which means that we can expand the Smart Boat network coverage by this distance through this technology. Hopefully, the future tests will also demonstrate that larger distances are achievable. The goal here is to cover 3G/4G blind spots with LoRa. For example, the network providers usually obtain calculated (optimal) coverage maps (see Figure 27), which do not show the real state in the nature, moreover the coverage is available only near the coasts. With the addition of LoRa offered by the inclusion of the mF2C architecture, the communication towards the cloud would be more reliable and also available on larger area.

In numbers, we have deployed 2 agents operating and controlling services over 2 powerful devices and two additional RaspberryPI units used as sensor and hub data collectors. When integrating the mF2C functionalities that has been developed for IT 1, 10 mF2C components spanning across 5 logical components were used. We are running 4 COMPSs jobs across different periods, totalling at over 25 job executions per day.

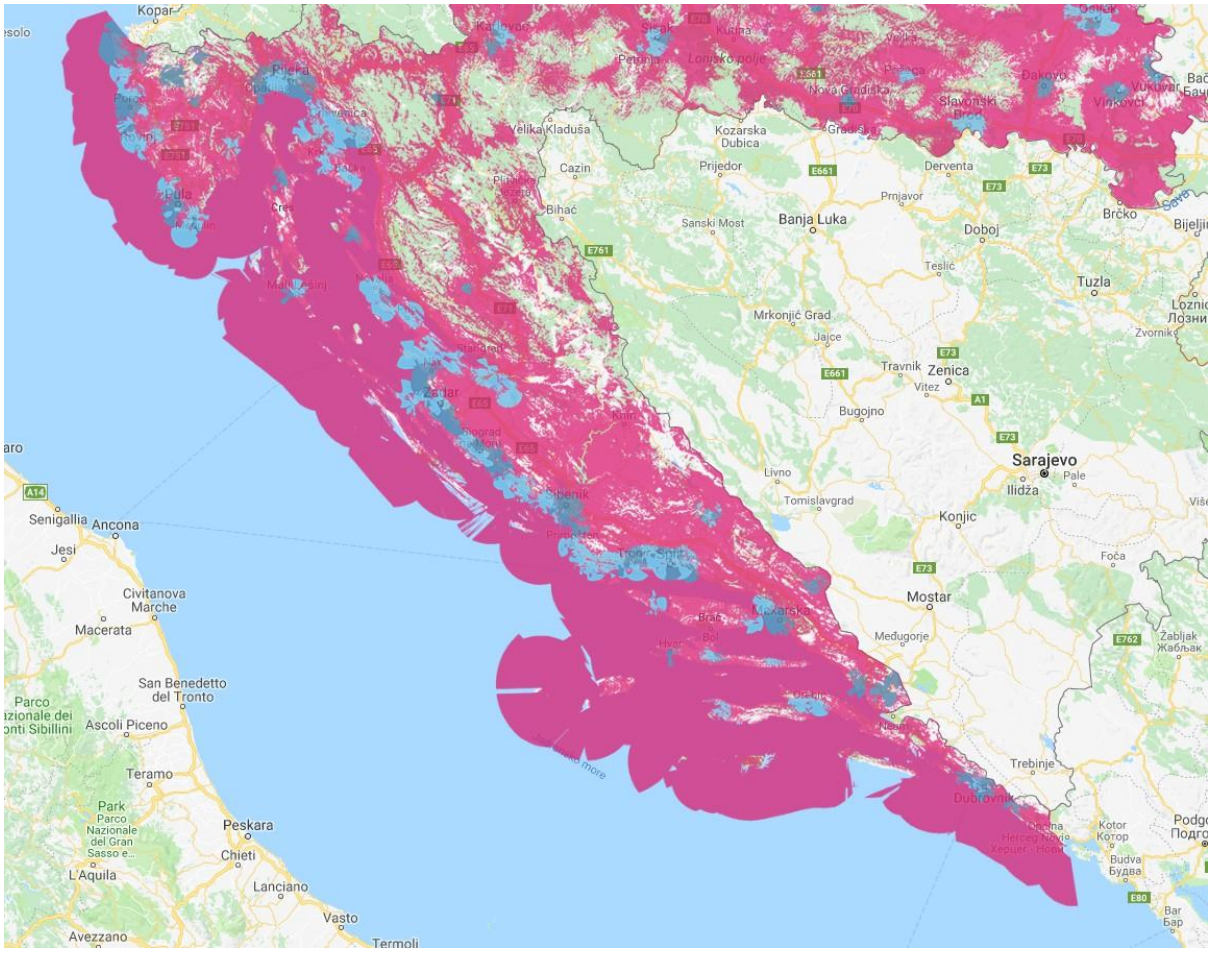

Figure 27 Calculated/optimal network coverage map for 3G/4G

5.6. Business Prospective

The Smart Boat use case foresees two business strategies to be formed and applied through the combination of Smart Boat and mF2C. The first one is based on the fog and cloud management of the mF2C platform, which takes care of software deployments, updates and computational offloading. This approach gives the Smart Boat use case an opportunity to focus on Smart Boat problems. The second one is the market demand for a variety of IoT solutions on all fields like IoT for persons, pets, buildings, cars, boats, etc. The IoT devices are already present in the marine segment but the goal is to develop a technology for a mid-end fog/cloud product that would provide the monitoring and control of the boat sensors. Beside the advantages provided by the mF2C platform, there is potential in the network coverage bridged by switching different communication protocols, namely WiFi, 3G/4G and LoRa. The exploitations of the use case could be offered as a developed technology, Smart Boat hardware and mF2C/Smart Boat subscriptions.

Boat rentals and hobby usage are becoming increasingly popular and serve a high-value market. All marine equipment, and especially upkeep and maintenance, has a relatively high cost, which is significantly larger than the cost of IoT equipment and services available for the boats. This creates a big demand for high-end solutions that provide high added-value. This market opportunity is perfectly catered to by cloud, fog and IoT solutions. The potential impact and target market is thus very large, and services do not need to make compromises that traditional mass-market consumer-oriented products face due to a requirement for low cost---products in this sector are much more flexible in terms of up-front or periodic cost and depend more on perceived premium added value that the product or services can provide the end users.

6. Use Case #3: Smart Fog-Hub Service (SFHS)

6.1. Complete Architecture Use Case

The following diagram represents the final architecture of the Smart Fog Hub Service Use Case, as foreseen for IT-1.

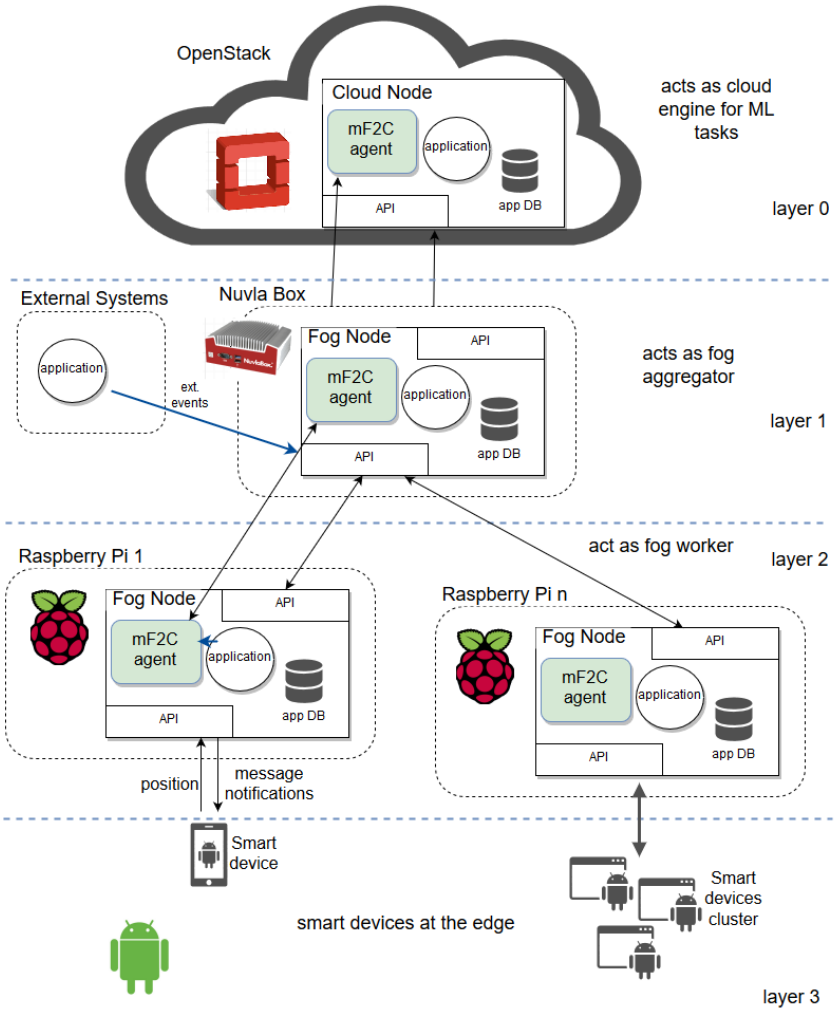

Figure 28 UC3 final system architecture

Air transportation is one of the most important modes of transport, and the number of passengers is constantly increasing, with a significant number of novice passengers. The airport structures are increasingly extended, offering an ever-greater number of services. For both novice and experienced travellers, the experience of moving through large airports to reach the right gate at the right time can be a stressful part of their travel. Monitors, indications and announcements aim at facilitating all operations, but moving in crowded and noisy places implies some form of stress when visiting airports. For this reason, the emerging idea is to identify an indoor navigator and recommender solution that could provide travellers a more enjoyable and stress-free experience. First of all, this solution would integrate all information that airports already provide through all digital monitors, information kiosks and voice announcement. Secondly, the entire airport map and list of available services with associated Points of Interest (POI), such as restrooms, shops, duty-free areas, information desks, gates should be included. This kind of solution aims at fulfilling the wish for a user-friendly and interactive indoor map offering passengers an easy way to get to preferred shops and other points of interest.

Instead of spending time finding their gate and waiting nearby because of the fear of missing their flight, travellers could explore the overall offering of shops, restaurants and services in their waiting time. This would result in excellent customer experience, but at the same time in an opportunity to increase revenue for the airport and service providers. This would be a win-win scenario.

Such a recommendation system needs to be designed as a kind of end-user interaction widely used in e-commerce (Amazon), social network websites (Facebook, LinkedIn, Twitter), streaming portals (YouTube, Netflix) or Large Retail sites.

Recommender engines try to infer tastes and preferences for a user based on his or her past actions and similarities to other users. In addition, recommender engines try to identify unknown items that might be of interest to users. People's tastes generally follow patterns. For instance, people usually tend to like things that are similar to other things they like, and they usually tend to like things that similar people like. Recommendation algorithms use these patterns to predict likes and dislikes. It is possible to generate recommendations based on either users or items.

The most commonly used and effective technique for recommender systems is Collaborative Filtering. It is widely used across social media, retail, and streaming services, and is based on the behaviour of users, and the idea that people that share an interest in certain things will probably have similar tastes in other things as well. The collaborative filtering recommender engine does not need to know details about the properties for each item to produce a recommendation. In our scenario, people that own a smartphone or other connected device get tracked and engaged to express their preferences and behaviour.

The overall idea is to track and engage all people and objects in the field and use a Collaborative Filtering based recommender system with machine learning algorithms to get the best possible customer experience, with suggestion on the best way to use available services, e.g. suggest to move close to the gate or notify the final call or recommend relevant proposals and offerings in shops close to the user. All these suggestions can be refined according to behaviour and choices made by passengers, by means of machine learning based features.

Particular care has been provided for the privacy and security of personal data: the recommender system uses algorithms that operate successfully without any personal information.

So, the set of IoT devices (smartphones) in the airport field generates a continuous stream of data (position and behaviour) that need to be processed in real-time.

These data can generate a feedback by way of event notifications (proximity, flight based, etc.) or suggestions based on users' similarities. At the same time, user behaviour and scoring of different Points of Interest require collaborative filtering (machine learning) tasks. The whole dataflow is represented in the following picture.

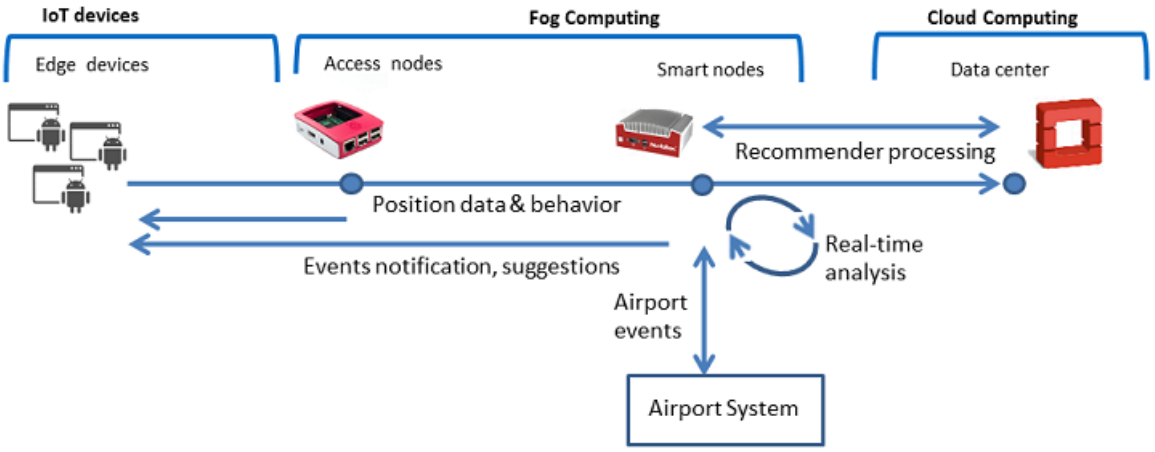

Figure 29 UC3 dataflow between different layers

The use of a recommendation system asks for massive data processing that grows with the number of managed objects. Even a smaller airport like Cagliari Elmas manages about 4 million passengers per year, so a relevant part of them (and their related behaviour tracking) will feed the machine learning algorithms.

Thus, there is a need to scale capabilities in order to balance huge amount of processing from fog to cloud nodes and manage distributed data. The mF2C framework offers all requested features to orchestrate and distribute the processing with automated scaling when needed.

The continuous tracking of people moving in the airport could allow for the discovery of bottlenecks in airport services or suggestions to improve available resources. At the same time, all dealers and service providers in the airport site will benefit from the application in their marketing proposals and offering by using the anonymized data collected by all people in the field. For these reasons, the recommendation system can facilitate the win-win goal (travelers experience, shoppers' revenues, suggestions for airport planners).

6.2. First Iteration Use Case (IT-1)

The following diagram represents the final architecture of the Smart Fog Hub Service Use Case, implemented in the Engineering test lab for IT-1.

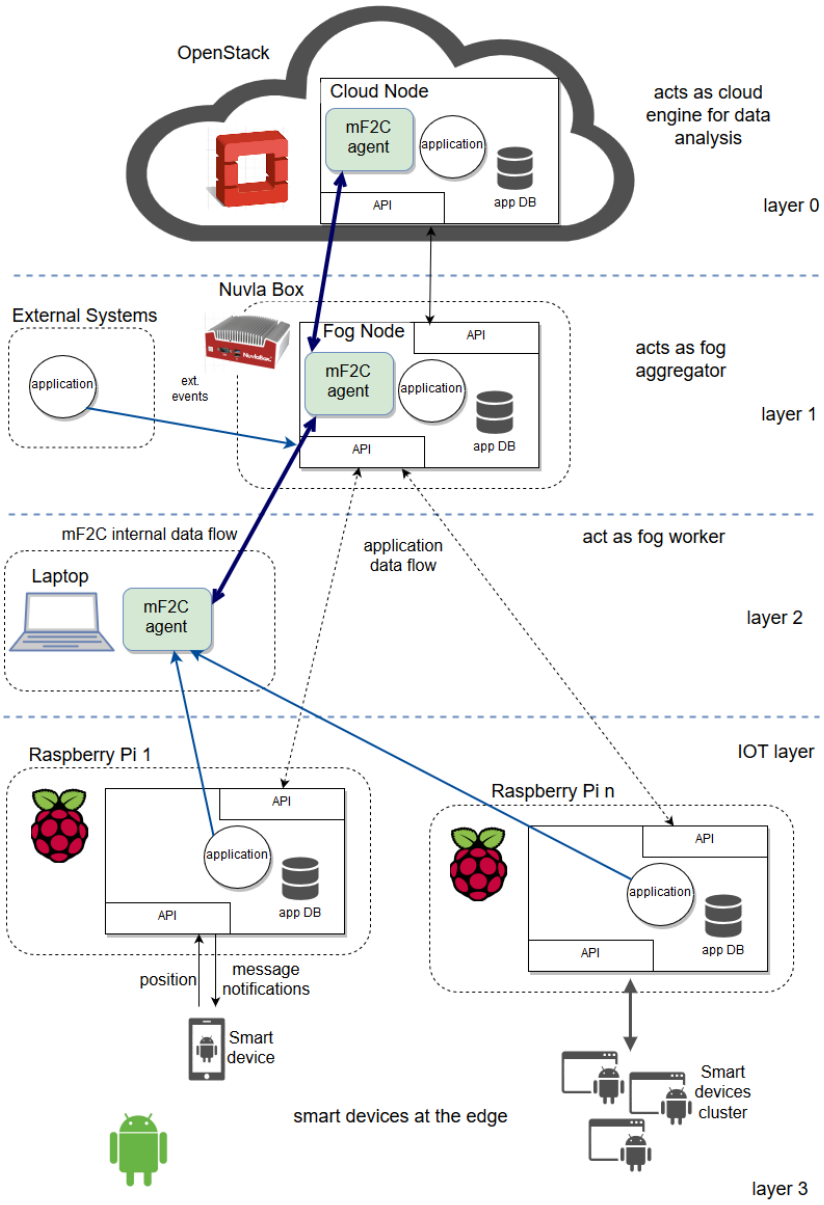

Figure 30 UC3 IT-1 system architecture

For IT-1, a preliminary 4-layers architecture has been defined and implemented with the following assumptions:

- only end-users using the android app are managed
- near real-time data sync on all layers up to the cloud is supported
- Mf2c agent is not supported on both android smartphones and RaspberryPI, so a laptop for real-time proximity computation is used
- no automatic failover management of fog leader is supported
- basic dashboard capabilities have been implemented
- external airport events have been simulated.

The resulting system architecture for IT-1 has been adapted and is composed of the following layers:

- A cloud layer, based on an OpenStack instance, wire-connected with the fog layers, that provide scalable computing power for machine learning algorithms used for the

recommendation system and historical data analysis, and long-term storage for all data produced that needs to be analyzed
- A first fog layer, that acts as fog leader, based on a NuvlaBox mini, equipped with 8 GB RAM, and provides real-time computing and storage resources to the edge elements
- A second fog layer, with an HP laptop with 4 GB RAM that interoperates with the NuvlaBox, runs the mF2C agent and acts as fog worker node, providing processing and resource capabilities to the IoT layer
- The edge layer with six RaspberryPI3 with 1 GB RAM, each of which acts as access element with secure data transmission, and provides session management, fast response to the edge devices and position tracking of smartphones used by end-users
- Android smartphones, used by the end-users, connected to the access nodes with Wi-Fi, and using an android app to be engaged with the system; in this phase, they are used as data generator.

The mF2C agent runs in the Cloud and Fog elements; nevertheless, it is not currently supported on RaspberryPI and android smartphones, this will be achieved and released for IT-2. The android app to be installed in the smartphone implements security and privacy features to preserve managed data both at rest and in transit, with a security level comparable to the ones adopted by the mF2C agent.

At the application level, the following business processes have been identified and are under development:

- App installation, device registration, privacy term acceptance and topics selection
- Continuous position calculation, checking for POI's proximity and subsequent user notification with proper messaging
- Position data sync to fog and cloud
- Airport events notification (open check-in, assign gate, flight call, to be extended also with invitation to move closer to the gate)
- Reporting (real-time and history) with the dashboard
- Management of POI and promotions (in case of shops)
- Filtering and behavior calculation in positions streams, recommendations generation based on user similarities has been postponed to IT-2.

6.2.1. Current Hardware and base Software for IT-1

In the following table all devices and related software are listed:

Edge IoT devices	
Smartphone used by end-users	
Hardware	Software
ASUS ZENFONE 4 MAX CPU: Qualcomm® Snapdragon™ 430 Mobile Platform with 64-bit Octa-core Processor GPU: Qualcomm® Adreno™ 505 (SD430) Memory: LPDDR3 3GB Internal storage: eMCP 32GB Display: 5.5-inch IPS Main Rear Camera 13 Mpix, Front Camera 8 Mpix Wireless Technology: WLAN 802.11 b/g/n; Bluetooth 4.1; Wi-Fi direct	Operating system: Android Nougat (v7) Custom app for user engagement

Sensor : Front fingerprint sensor, Accelerator, E-Compass, Gyroscope, Proximity sensor, Ambient light Navigation : GPS, AGPS, GLO, BDS	
RaspberryPi3 Model B 1.2	
Hardware	Software
Broadcom BCM2837 SoC, with quad-core ARM Cortex-A53 1200 MHz processor VideoCore IV dual-core 400 MHz GPU 1 GB SDRAM - shared by the GPU and CPU MicroSD card slot for boot and storage 4 x USB 2.0 ports (via on-board 5 port hub) RJ45 10/100 MBit/s Ethernet port HDMI and Composite video, CSI (camera) and DSI (display) connectors	Operating System: Raspbian Jessie Lite Docker version 18.03.0-ce Container A: mongodb v3.2.18 32bit Container B: nodejs v9.10.1, express, mongodb driver for nodejs, mongoose
L2 Fog-capable devices	
HP Laptop	
Hardware	Software
ibm thinkpad t420i model 4236G90 intel i3-2350m cpu 2.30GHz 4G ram + 4G swap space 160G HD	ubuntu 18.04 - centos 7.4 mF2C agent (Lifecycle + COMPSs)
L1 Fog-capable devices	
Nuvlabox Mini	
Hardware	Software
Intel Celeron processor 8 GB RAM 128 GB SSD 60 W external power supply WiFi 2 USB 2.0; 2 USB 3.0 VGA / HDMI 2 x RJ-45 network interface	Centos 7.1 CloudLayer: OpenNebula Virtualization: Kvm AppStore: SlipStream CentOS Linux release 7.4.1708 (Core) Docker 18.03.1-ce - Docker Compose 1.21.2 Container A: mongodb v3.6.0 Container B: nodejs v8.9.2, express, mongodb driver for nodejs, mongoose
L0 Cloud-capable devices	
Cloud Virtual Machine	
(Virtual) Hardware	Software
VM with 2 vCpu, 16G RAM 100G storage	Centos 7.4 MF2C Agent Application for data collection

Table 7. Current hardware and base software in UC3

6.2.2. Current Software Responsibilities

The application logic is distributed over all layers of the architecture, according to the principle that processing of data should be executed closest to the edge. Only when this does not fulfil quality and/or security requirements (e.g. latency) is the processing moved to fog and cloud (as necessary).

The following list describes the functionalities, as defined for IT-1. These will be extended during IT-2, accordingly to the continuous feedback received from internal and external stakeholders.

Cloud Software

1. Takes part in the distributed real-time computation by making resources available if needed
2. Receives and stores aggregated device position and user session data through the local network
3. Stores real time and long term historical data for further analysis
4. Analyses big data (both real time and historical) in order to identify behaviour patterns and generate recommendations to be sent back to the affected users.

Distributed real-time computation in the cloud is under development and will be fully implemented in IT-2. Big data and predictive analysis will be designed and implemented in IT-2.

Nuvlabox Software

1. Takes part in the distributed real-time computation making resources available, acting as COMPSs worker
2. Receives and stores device position and user session data from edge devices (RaspberryPIs) through synchronization tasks, and aggregates them
3. Receives and stores notification messages (about airport services and general topics) from airport external system, and forwards them to edge fog devices
4. Stores POI advertisement or related messages and forwards them to edge fog devices
5. Provides dashboard application for real time and historical data representation
6. Generates data for the dashboard.

The dashboard software currently implements basic functionalities (display of real time sessions in the field, heatmap for most visited places in the field, and historical data diagrams), and will be completed with advanced features in IT-2.

Laptop Software

1. Deploys the mF2C agent software
2. Starts the service orchestration
3. Receives data to be used in real-time computation from backend software (installed in RaspberryPIs)
4. Takes part in the distributed real-time computation making resources available, acting as COMPSs master.

For IT-1, the laptop is used in place of the RaspberryPI in order to host the mF2C agent. In IT-2, the mF2C agent will be installed on RaspberryPI.

RaspberryPI Software

1. Acts as local network Wi-Fi access point and manages clusters of connected user devices
2. Provides backend software (REST API server with SSL) for the Android app installed on user devices

3. Forwards device position data to the mF2C agent in the laptop, used for real-time distributed computation
4. Receives device position and user session data from end user devices (using a secure socket communication), and forwards them to the leader fog device (NuvlaBox)
5. Receives notification messages about airport service or general topics from the leader fog device, and forwards them to the affected devices
6. Receives and stores POI advertisement or other related messages from the leader fog device and forwards them to the affected end-user devices.

6.2.3. Use Case Services

Use Case 3 proposal starts from a customer perspective, focusing on the devices that move throughout the airport. Customers have specific individual needs and/or interests (about their flight, or about shops, fast food and services nearby, entertainment in general, emergency situations) and generate lot of (near real time) data. The fog is the way to aggregate and manage the growing number of devices, with optimal response time and resources usage. The fog nodes, upon demand, can recursively ask resources to the same or upper level (until the cloud). The app takes care of customer data privacy & security, only position and mac address personal information is managed, at the beginning of the mobile app the user is informed about all treatments done on personal data and consent on it. All Data Privacy issues are considered within the Data management deliverables.

The aim of this use case is to improve the travelers' experience while in the airport area, track their position, propose proximity-based suggestions such as information on all available shops and other points of interest nearby, or inform and guide the traveler towards flights that he has selected as relevant (e.g. the flight he has to take). The use case will be connected to the airport management system to receive and dispatch flight information to interested users. The administrative dashboard could be used to define additional information and suggestions like promotions or different alerts to be sent to customers.

The huge amount of data generated by travelers will be managed at fog level for real-time processing, while processing of heavy off-line computation or management of historical data will be moved to the cloud. The dashboard will enable the visualization and analysis of travelers, with different reports and dynamic maps (heatmaps) on most visited places.

6.2.4. Tasks to be executed by the mF2C Agent

The mF2C agent is requested to support the calculation of Points of Interest nearby each active user in the field. This happens about every second for all users, so a huge amount of processing with strict real-time boundary. This seamlessly demand intelligent distribution of processing, leveraging the Fog-to-Cloud approach, provided by the DER (Distributed Execution Runtime) in the mF2C agent.

Another relevant point is related to the recommender system, under development for IT-2, that requires the use of machine Learning Algorithms (Collaborative Filtering) to determine similarities between users, so chance to propose for every user some places that similar users liked most. This amount of processing grows as the square of registered users, so the processing gets too heavy to be managed at fog level, so the processing needs to be run offline on the cloud and a cache of the result managed in the fog.

Finally, the data analysis processing is performed according the fog-to-cloud approach: the processing is assigned to the fog first, but it is up to the mF2C agent to decide if the processing needs to be moved on the cloud.

6.3. Data Flow Diagrams

For each of the foreseen business processes, the corresponding sequence diagram have been defined. In the following paragraphs the tested dataflow is described in more detail. Other data flows will be included in the final version.

6.3.1. App setup, privacy terms, topics selection

The Android app is installed in the end-user device (smartphone, tablet) and allows access to the airport platform services. The user must accept the Privacy Terms before using the app. Actually, no personal data are required to use the app; thus, no login is required and the user is managed anonymously. The only data managed by the system are session data including device position and preferences like topics of interests.

The app connects via Wi-Fi to one of the RaspberryPI in the field which acts as access point for the local network. When the app connects, a session is created and this session uniquely identifies the device in the system. For communications, the app uses the REST API of the backend software installed in the RaspberryPI and the data are transmitted using https. For IT-1, the app includes the following functionalities:

- Displaying of device position and Points of Interest (POI) in an indoor map
- Selection of topics of interest to the user. It could be flight information, generic topics related to particular events, activities or locations in the airport, etc.
- Notification and displaying of text messages related to the topics selected by the user
- Notification and displaying of text messages related to POI when the device is in the proximity of the POI itself. The content of the message is configured in the system and is typically advertisement or other information that could be useful for the user. The proximity is defined as the spatial distance between the current user and POI position. This can be configured differently for each POI in the system. Thus, it could happen that the app is notified for more than one POI at a given time, depending on how many of them are nearby.

These functionalities are implemented in the app in different screens, accessible one at time through a multi-tab activity.

Currently, the screens are the following:

- An indoor map for device position and POI displaying
- A list of notification messages, both for topics related messages and for POI related messages.
- A list of topics of interest to the user. For IT-1, this list includes all the flights in a given day in the airport.

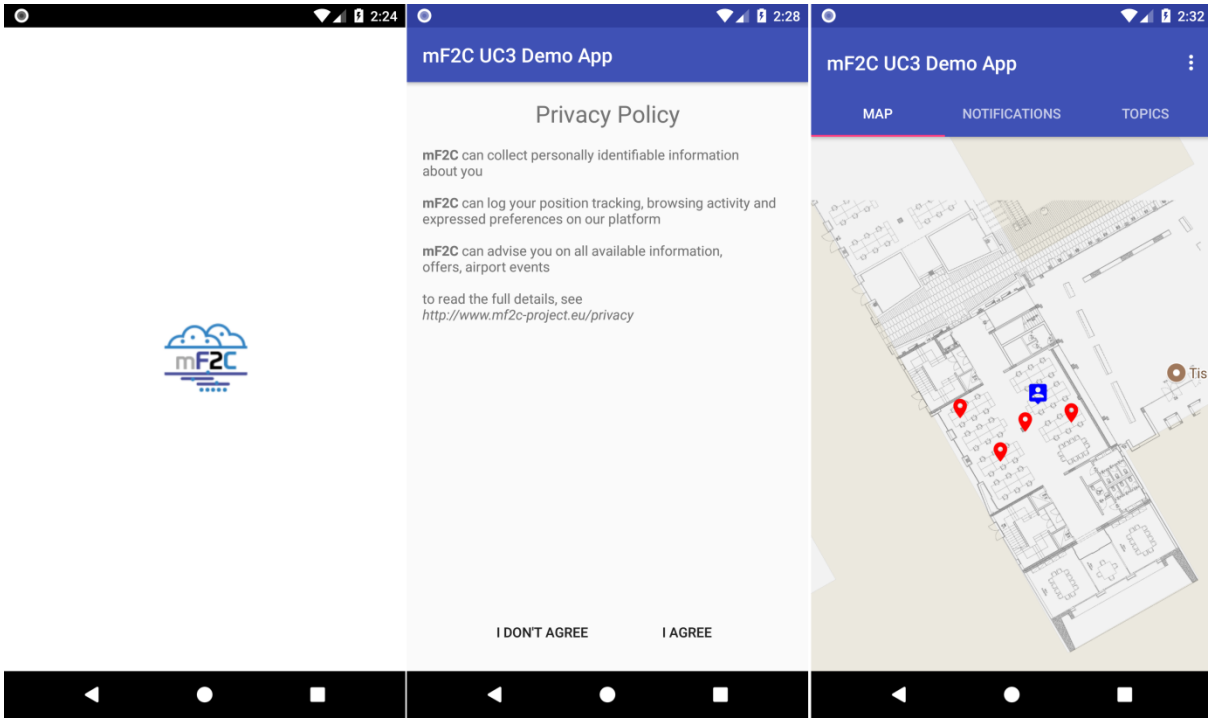

Figure 31 Android App screenshots (splash screen, Privacy Policy Terms, Indoor map)

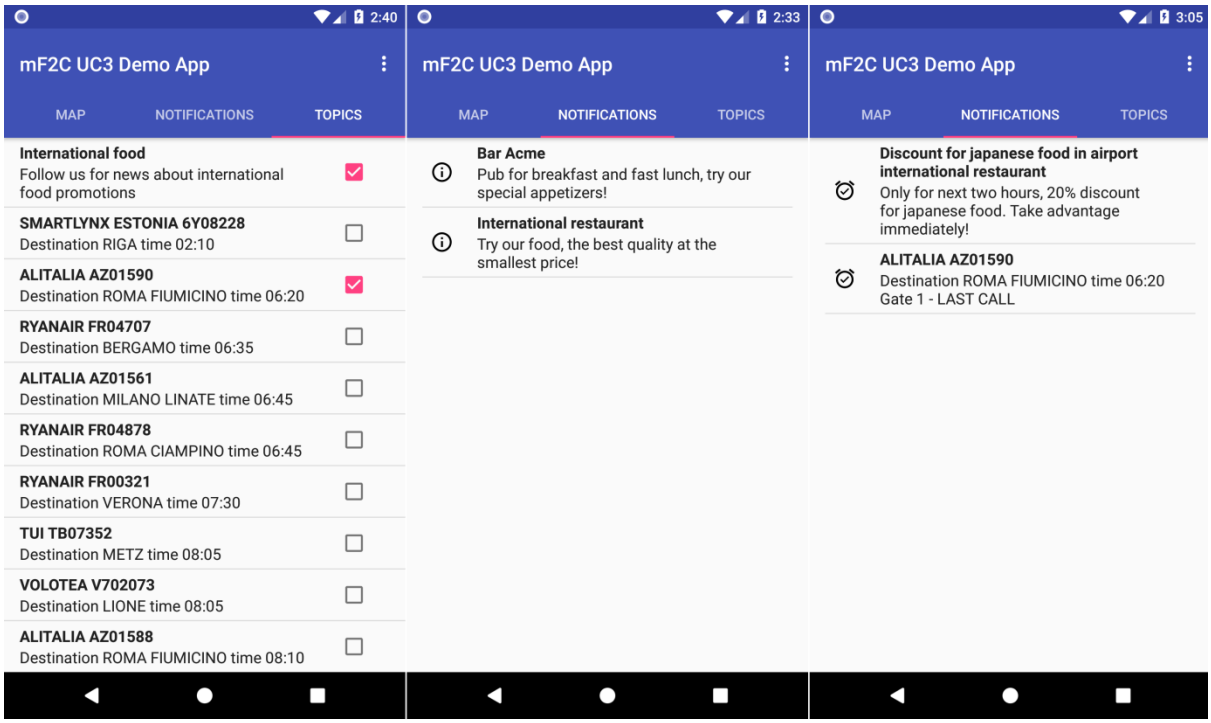

Figure 32 Android App screenshots
(topics selection, proximity detection POIs, airport and topics notified msg)

6.3.2. Object monitoring and tracking with proximity notification

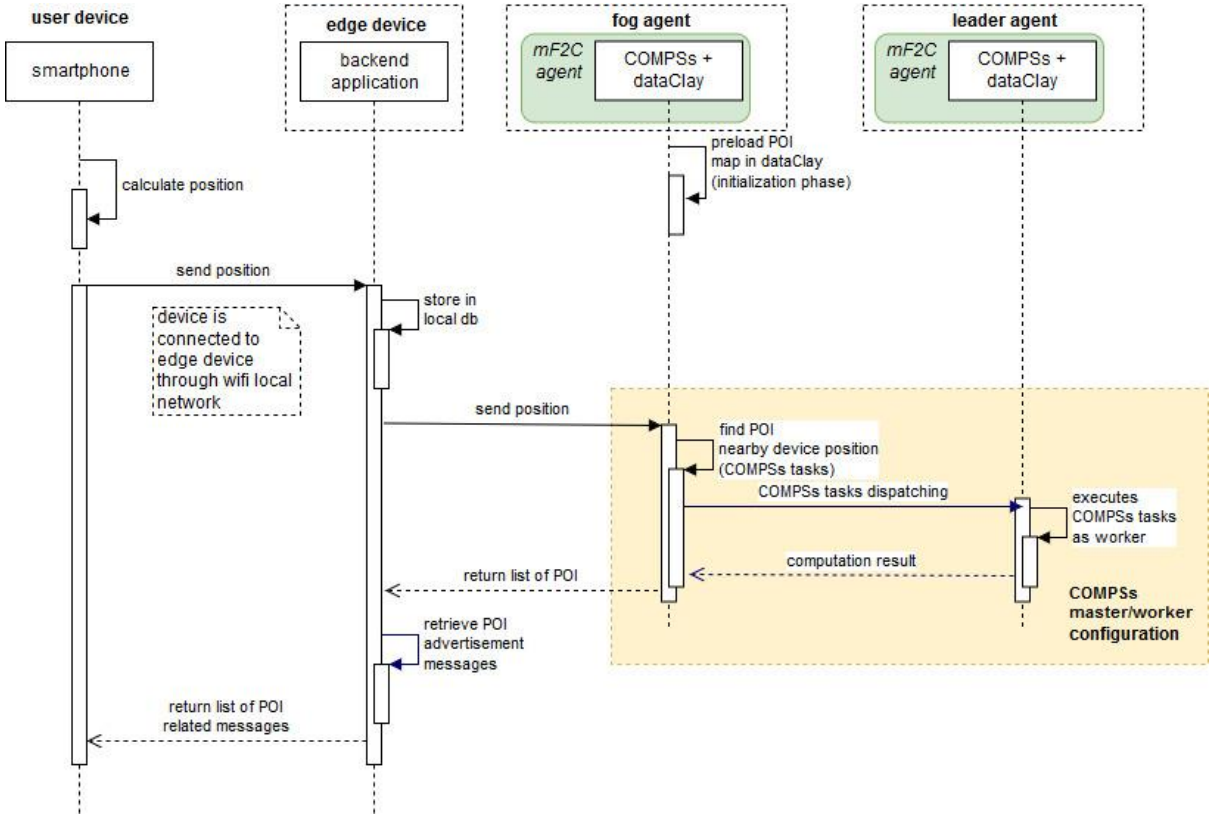

Figure 33 Point of Interest (POI) proximity handling

The Android app installed in the end-user device continuously computes and delivers device position at a given time interval to the backend application installed in the RaspberryPI (edge device) which it is connected to. The backend application stores the position in the local database and forwards this data to the COMPSs application installed in the fog device and managed by the mF2C agent. The COMPSs application is in charge of computing the distance between each POI and the device and generate a list of the POI which are nearby with respect to the proximity criteria. To perform this computation, the COMPSs application uses different tasks which are internally dispatched and managed by COMPSs itself, involving also the fog leader agent. In this case, the edge fog agent is the master and the leader agent is the worker. Currently, a COMPSs task is used to compute the distance between a single POI and the current device position. This fine granularity approach allows for scaling in case of many users in the field and/or a large number of POI to be managed.

When COMPSs ends the computation, the list of nearby POI is sent back to the backend application which retrieves from its local database all messages pertaining to the found POI. Finally, these messages are sent back to the device app to be properly displayed.

Since the COMPSs application uses the position of POI for its computation, a full map of all POI in the field is pre-loaded in dataClay at system boot phase. COMPSs uses dataClay internally for its data management.

The backend application installed in the fog area devices is written in NodeJS and uses MongoDB as local database, while the COMPSs application is written in Java.

6.3.3. Airport events notification

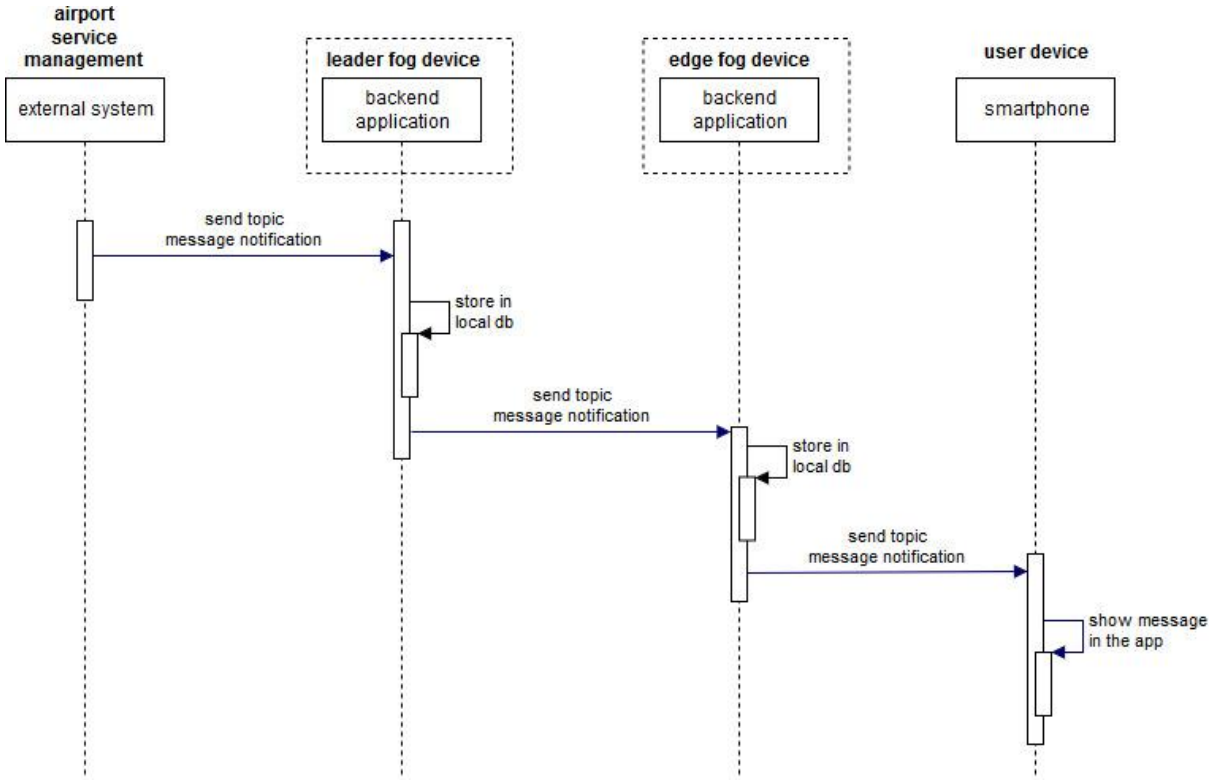

Figure 34 Topic message notification

The external airport service management system is in charge of sending messages about specific topics like events or other activities which occur in the airport. To do this, this system uses specific API of the backend application installed in the leader fog device.

Figure 34 illustrates how, when a message is available, the external system sends it to the backend application of the leader device. The backend application delivers the message to all edge device in order to make it available to the end-user device, which will receive proper notification, and the text of the message. The delivery of the message to all devices is made in near real-time through specific synchronization jobs.

Each fog device, both leader and edge, stores the message in its local database, which is used as cache and is involved in the synchronization process.

For IT-1, the external airport management system is simulated by a specific software.

6.3.4. Behaviour analysis and forecasting

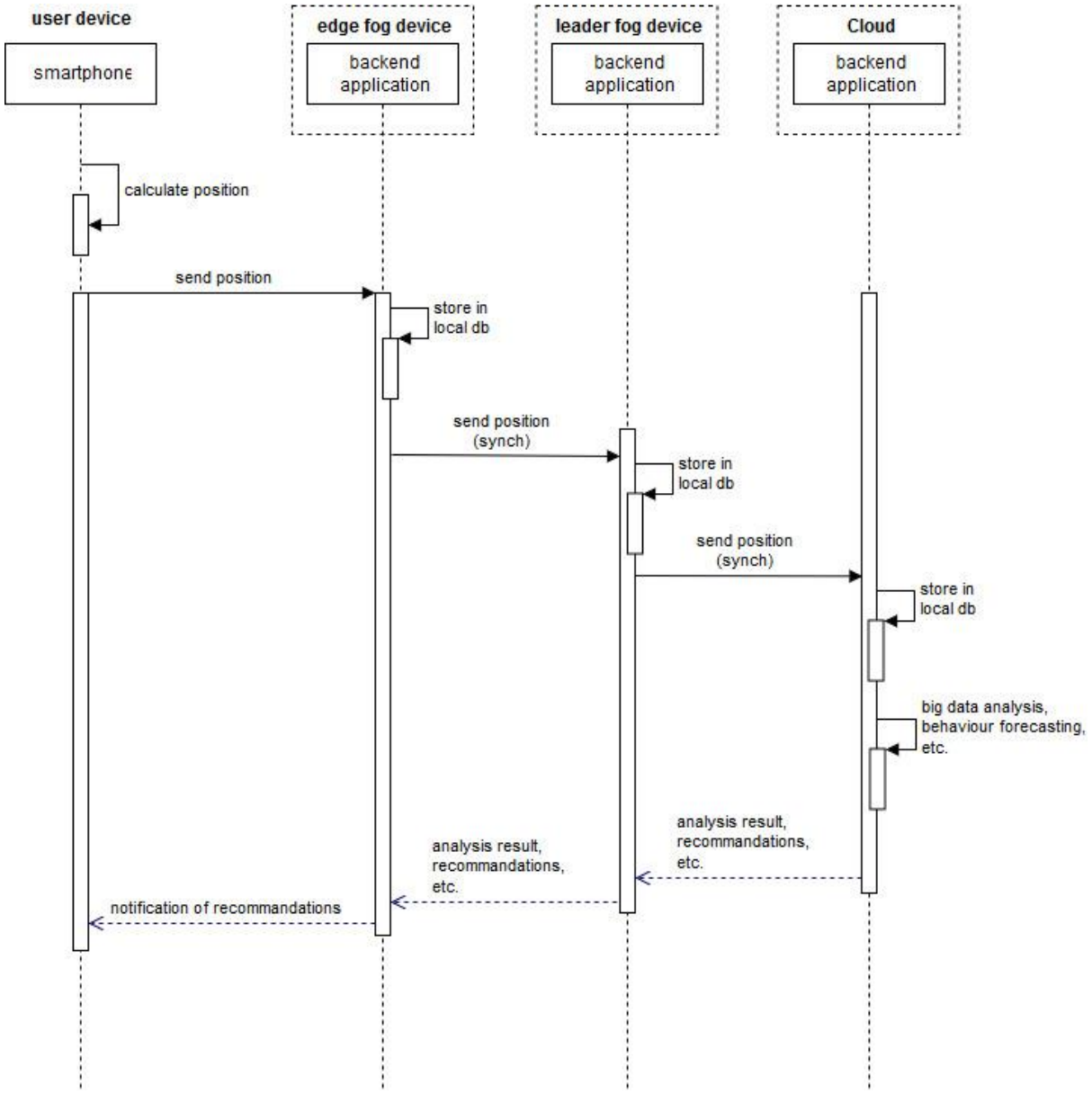

Figure 35 Big data analysis, predictive analysis (IT-2)

Predictive analysis can be done starting by data collected from the devices during their normal activities. Typically, the position is collected from the devices and, as a consequence, the POI met during their movements in the field and potential ranking on POIs. All these data are sent from the end-user device to the edge fog device which it is connected to, stored in the local databases for caching, and forwarded to the upper fog leader and cloud layers (see Figure 35). All these layers store the data in their own local databases. The forwarding of these data is made by specific synchronization jobs in the backend software.

The processes that implements the behaviour analysis for forecasting are not included in IT-1. Their evaluation and design has been started and it will be included in IT-2.

6.3.5. Dashboard

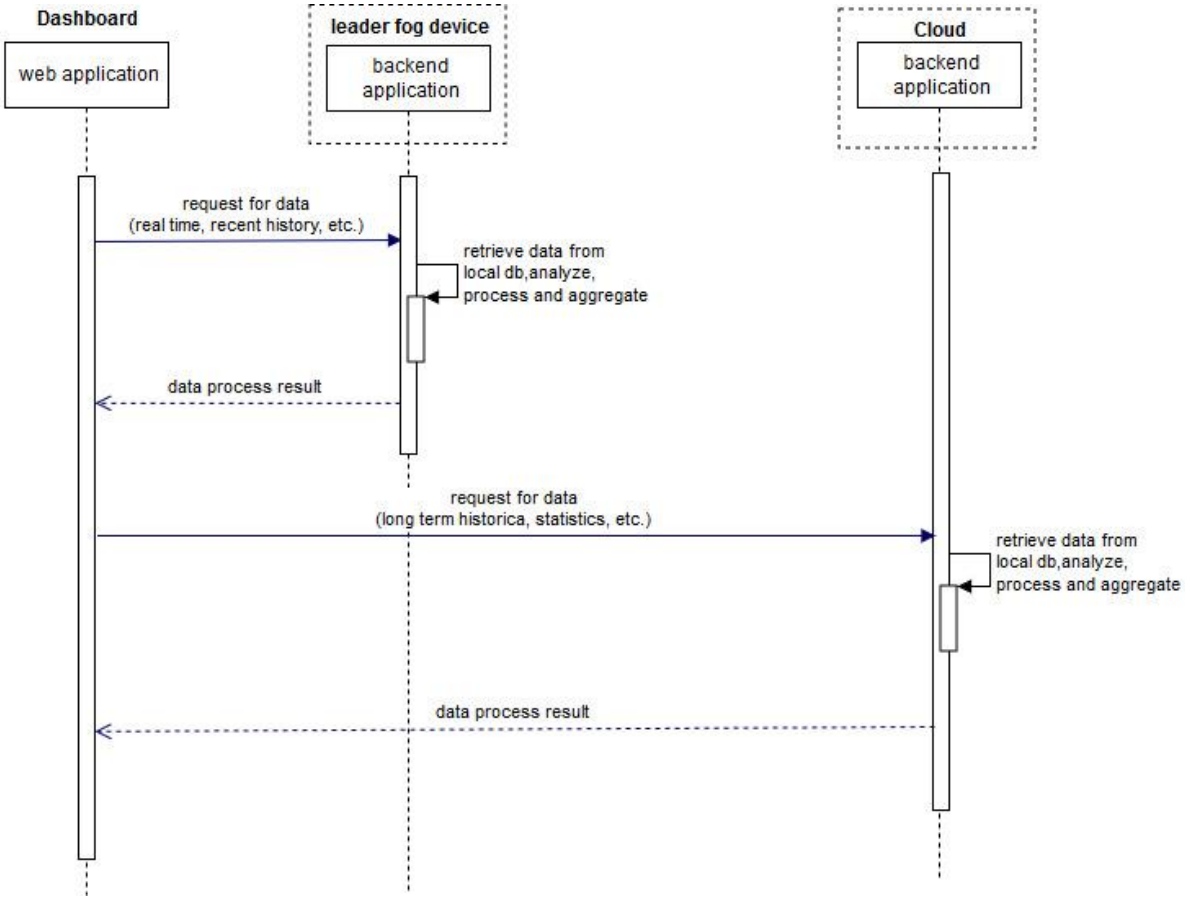

Figure 36 Dashboard (partially implemented in IT-1 and to be completed in IT-2)

The dashboard is a web-application designed for visualization of data and statistics, and for management of airport services by authorized operators. The access to the dashboard is allowed through username and password login for specific users registered in the platform. The data flow is illustrated in Figure 36.

The users of the dashboard are categorized by roles with specific permissions. For example, typical users could be administrative operators of the airport, surveillance operators, airport locations owners, shoppers, etc.

For IT-1, the dashboard does not implement user role management and includes basic features like visualization of real-time sessions of devices present in the field, heatmaps of historical data and some diagrams about session trends in the last month and last year. Due to the lack of long term historical data, in IT-1 the dashboard will use data which are collected in the fog area only.

In IT-2, the dashboard will be completed by a full set of features including visualization of big data analysis result like similarities of the users' behaviours, etc.

6.4. Experimentations Set Up

The architecture mentioned in chapter 6.2 has been deployed in the Engineering site in a specific test lab and used for the functional tests of the use case. According to the DoA, during IT-2 the use case will be moved to the Cagliari Elmas airport. Users download the android app and move in the field, receiving suggestions on their selected flights or getting information about POIs nearby. At the same time, data collection on position tracking and behaviour has been done, giving the opportunity to test all the data analysis and reporting features. Fifteen tests have been performed, with the deployed paths and behaviour, which has allowed to functionally validate the IT-1 use case, although statistically significant results should be obtained in the following months.

Some videos that represent the travellers experience have been recorded to be used for further presentations, including the M18 EC Project Review in Brussels. Also, screen recording tools have been used for both the android app and PC station, showing logs (mF2C agent and application) and dashboard for real-time objects tracking.

6.5. Results

Even if most of the work performed was functional instead of performance oriented, the results collected so far are very promising: the mF2C agent is pretty fast in processing and load distribution in case of need. The Fog-to-cloud approach is managed completely by the mF2C agent.

The base proximity computation has been developed and benchmarked with different architecture approaches, using different communication protocol for the smartphone-server link (wi-fi, 3G, 4G/LTE), and using fog or direct cloud communication. The direct cloud approach performed 2X the fog-to-cloud architecture with Wi-Fi communication. That's pretty evident that in direct cloud topology latency made the difference. We have also to consider that the fog-to-cloud topology enable filtering and optimization on resources and communications while fulfilling the real-time requirement (<100msec), not possible with direct cloud topology. Actually, we are in a first data gathering step that will enable us to get as much feedbacks as possible to be used for the improvement of the use case in the second iteration. At the same time, more capabilities are expected in IT-2 on the mF2C side, enabling even more sophisticated solutions to be deployed on the testbed.

During the second iteration, we will work on the definition of performance metrics to be applied at both system (mF2C) and application level, that could be useful to determine the benefits that we have been able to perceive during IT-1 in terms of reduced latency and load distribution in case of overloading. These metrics will help in the final validation of the Use Case and the mF2C infrastructure.

6.6. Business Prospective

There is an increasing demand in evaluating and identifying new market sectors and opportunities, and interest at the IoT evolution as a potential arena where current Cloud offering could be enriched and differentiated. In this perspective, a relevant focus in setting-up hubs in public environments (e.g. airports, train stations, hospitals, malls and related parking areas) is suggested, capable of tracking the presence of people and other objects in the field and developing added-value services on top for proximity marketing, prediction of path/behaviour of consumers, and making real time decisions.

The foreseen hubs could be adapted to be used as a planning tool for determining the number and distribution of people that use, or can potentially use, various services like public transport, etc. This kind of hub can be easily considered as a fog device that should embed cloud connectivity to either process large amount of data or request extra-data – perhaps data coming from other fogs nearby.

As an additional opportunity to be evaluated, different fogs located in near sites (e.g. airport, train/main bus/ harbour station) could interact sharing data and customer behaviour gathered to improve the effectiveness of marketing proposals, given that the identity of objects/customers is protected.

The envisioned Smart Fog-Hub Service will be set-up in public crowded environments, so a preliminary version has been tested within the Engineering Campus in Cagliari, and a final version will be deployed at the Cagliari Elmas airport. With this approach, the whole infrastructure will be tested and validated in a real scenario with possibility for Engineering to exploit the marketing potential of the developed services.

7. Combined mF2C functionalities

As explained before, some of the mF2C functionalities are demonstrated in IT-1 independently of the execution of an mF2C service. These are functionalities are detailed in the upcoming sections.

7.1. User Registration to mF2C system

In this functionality specifies how a user can register to the mF2C system, download the mF2C app (consisting in the single Docker Compose YAML file) and install the mF2C system in his/her device. The user must provide his/her email and a password to the system and he/she receives a link to obtain the user ID, as well as the device ID. If a user is already registered in the system, upon access, he/she must provide his/her user email and the password to access the system, and he/she can obtain device IDs for his/her different devices willing to participate in the system.

The registration-identification demo consists in the following steps:

1. Access to mF2C dashboard, http://dashboard.mf2c-project.eu:800/registration.html, to provide the user's email and create a password

2. Reception of a link to obtain the user ID, the device ID as well as access to the mF2C dashboard to download the mF2C agent

3. Installing the mF2C agent.

Once the user is already registered, he/she has access to the whole mF2C dashboard, where he/she can access to the service catalogue, with all the services available for the user. In addition, the user as developer of new services, can register and upload a new service in the mF2C catalogue. The steps to register a new service are:

1. Access with the user and password to the mF2C catalogue

2. Go to the link "Register New Service" http://dashboard.mf2c-project.eu:800/scripts/services/php/service_registration.php

3. Fill in the form with the characteristics of the service such as:
    - Name, description, name of the executable and needed ports
    - Requirements CPU, memory, Storage, Disk and Network
    - Type of sensors connected to the agent
    - Type of executable:
        - Docker
        - COMPSs application
        - Docker-compose.

7.1.1. Architecture

Figure 37 shows the architecture of this functionality. The device connects to the mF2C provider in http://dashboard.mf2c-project.eu:800/registration.html to be registered in the system.

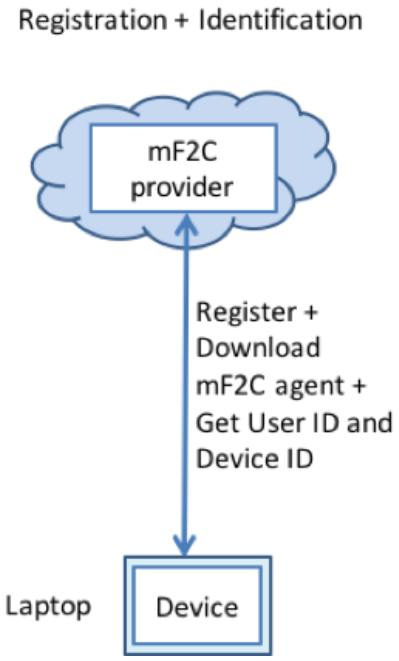

Figure 37 Combined mF2C Functionalities Demo Architecture

7.1.2. Experimental set-up

To perform the demonstration of the registration and identification steps, although the agent could be run on a less powerful system, we will use a laptop with the following characteristics:

1. Intel i7 – 7700HQ @ 2.8 GHz x 8
2. 16 GB RAM DDR4
3. 250 GB PCI-E Gen.3x4 SSD + 1 TB SATAHDD

For IT-1, the mF2C cloud layer where registering and obtaining the ID is performed is located at Engineering premises and consists on a virtual machine in an OpenStack infrastructure which can be accessed through Nuvla Box. The monitoring of the VM behaviour can be performed internally through this link: http://dashboard.mf2c-project.eu:8080/containers/.

7.1.3. Results

These functionalities have successfully been tested. Once the user has registered through the dashboard (shown in the screenshot of Figure 38), access is allowed to the mF2C dashboard (Figure 39) and to the service catalogue (shown in Figure 40) with four available services.

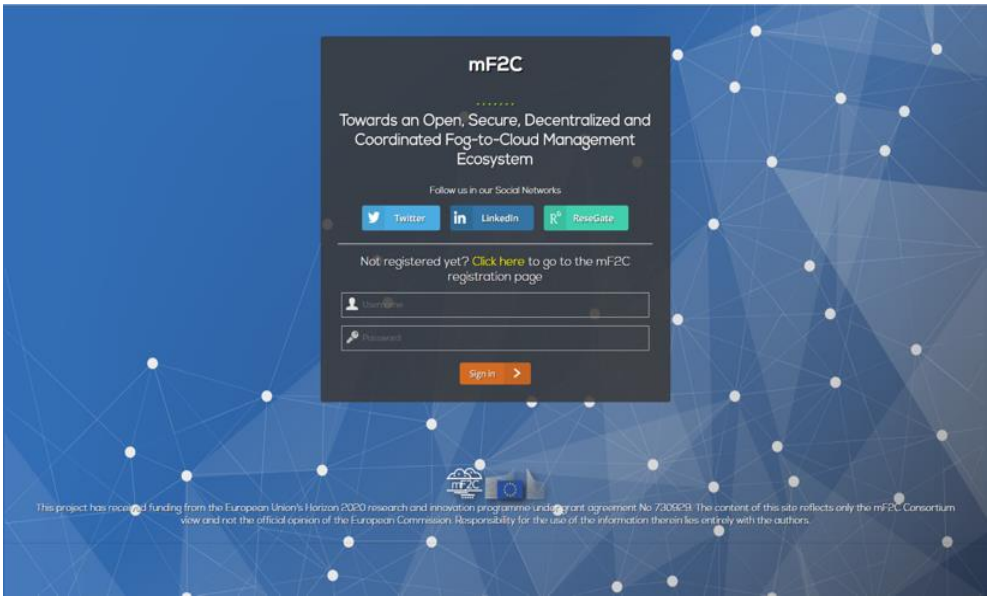

Figure 38 Registering a new user

Figure 39 mF2C dashboard

Figure 40 Service Catalogue

Finally, the user in his/her role of developer can register a new service, as shown in Figure 41.

Figure 41 Registering a new service

7.2. User (and resources) participation in mF2C

Once the user has registered and has a user ID and his/her device a device ID, the device becomes an mF2C agent that can participate in the mF2C system as a consumer or as a provider. First of all, this device with the installed agent must be discovered by a leader in order to become an agent in the fog area of this leader. The steps to be discovered, authenticated and categorized its resources are:

1. The agent activates the scanning to scan possible beacons from a leader
2. The agent detects the beacons from a leader, containing the leader ID
3. The agent answers to the leader with its MAC address
4. The agent uses the CAU-client to generate the CSR and send it, together with the leader ID, its device ID and user ID, and its MAC address, through a TLS connection to the CAU
5. The CAU forwards this request to the CA in cloud; and if it is a valid agent, returns the signed certificate to the CAU
6. The CAU returns the signed certified to the agent. The agent authentication process is performed by communicating with the leader
7. The categorization module in the agent is executed and information about its resources is stored in its database (DataClay), and later synchronized with the leader's database
8. The policies module is executed and if the device is capable of being a Leader, it becomes a potential Backup.

7.2.1. Architecture

The proposed architecture for this providing this functionality is shown in Figure 42:

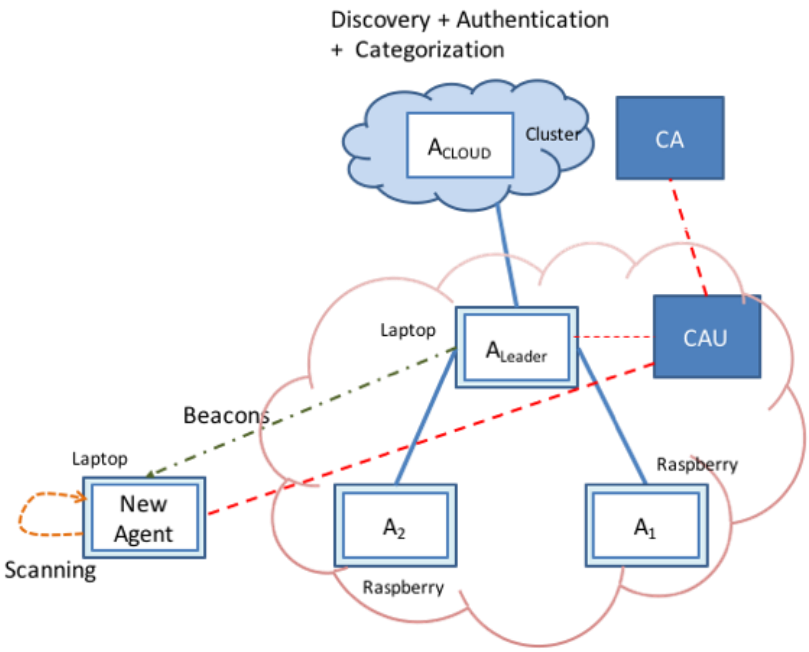

Figure 42 Discovery + Authentication

7.2.2. Experimental set-up

The experimental setup is shown in Figure 43, where the two agents in the area represented in Figure 42 are two RaspberryPIs, the leader a laptop, and the cloud including the CA is a cluster. The new agent willing to join the mF2C area is also installed on a laptop.

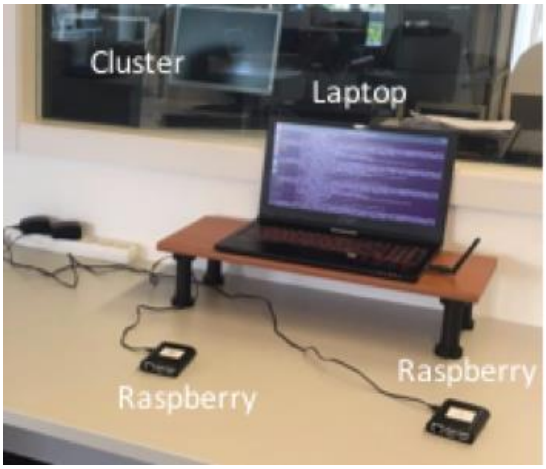

Figure 43 Experimental Set-up for Discovery + Authentication

The characteristic of the devices used in this demo are:

- Laptop1 (leader)
  - Intel i7 – 7700HQ @ 2.8 GHz x 8
  - 16 GB RAM DDR4
  - 500 GB PCI-E Gen.3x4 SSD + 1 TB SATAHDD
- RaspberryPI (agents)
  - Quad Core 1.2GHz Broadcom BCM2837 64bit
  - 1GB RAM

- 16GB microSD
- Communication interfaces: USB, Ethernet / BCM43438 wireless LAN

· Laptop 2 (agent willing to join the mF2C area)

- Intel i7 – 7700HQ @ 2.8 GHz x 8
- 16 GB RAM DDR4
- 250 GB PCI-E Gen.3x4 SSD + 1 TB SATAHDD

7.2.3. Results

A front-end or dashboard was developed to display the topology and events of a cluster of devices including the leader and the agents. This activity includes the inclusion of a new device after the process of discovering, authenticating and categorizing, as well as the changes due to failures, disconnections, etc. In Figure 44, the dashboard is represented with its 3 main panels:

- The right part of the screen graphically represents the devices in the cluster, as well as the CAU and CA. The current image corresponds to the demo of discovery, authentication and resource categorization. The laptop inside the cluster only appears in the dashboard once it is discovered, authenticated and its resources are categorized and shared with the leader
- On the left side, the information is displayed in two boxes:
  - The upper one shows all the events occurring in the system in the form of messages
  - In the bottom one, the events taking place in the selected node are displayed. The front-end allows the user to select a node with the mouse in the graphical part to visualise the events occurring there.

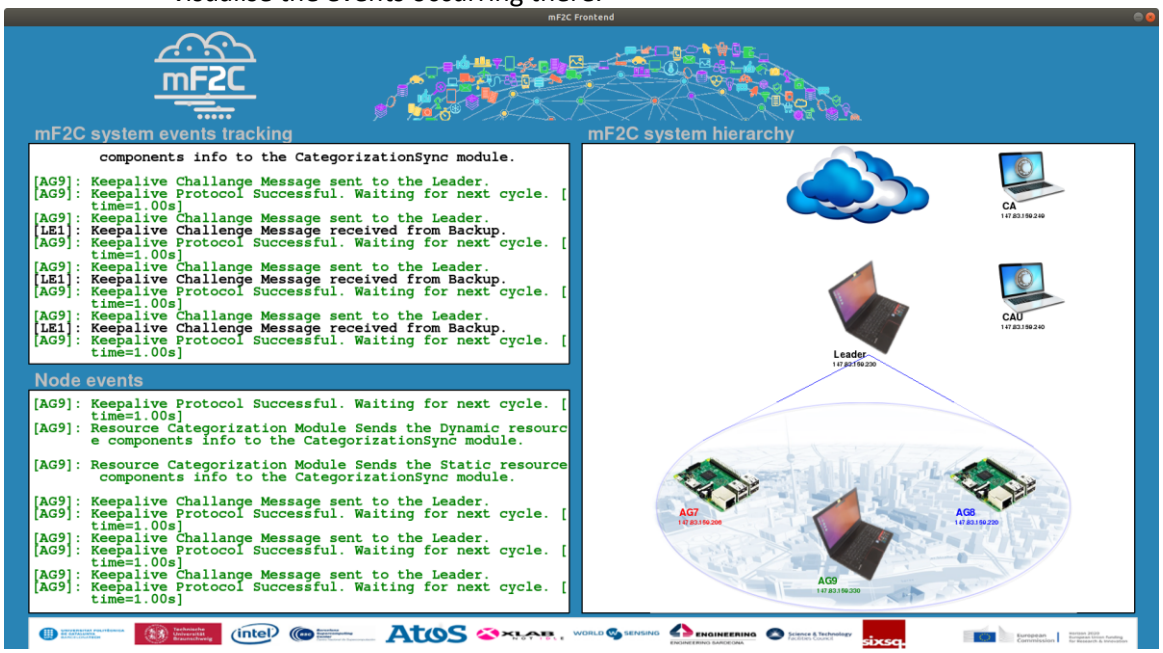

Figure 44 mF2C front end activity when discovering a new device

7.3. Failure of a mF2C area leader

In this functionality, we demonstrate how a leader can be substituted by its backup in case of failure. The steps performed are the following:

1. The leader of an mF2C area (cluster) fails
2. The backup agent in the area is continually sending keepalives to the leader. When it does not receive an answer, it detects the leader failure

3. The backup agent has already in its database the topology of the area and only has to switch some modules in order to become the substitute leader, for example the discovery to start sending beacons.

The new leader selects a new backup agent.

7.3.1. Architecture

The complete architecture is shown in Figure 45 and is the same that is being used for the Discovery demonstration once the new agent is added to the mF2C area.

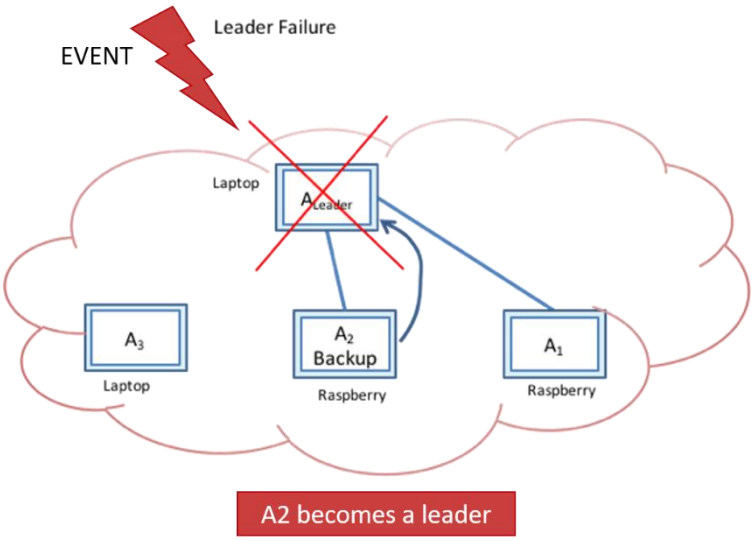

Figure 45 Architecture for leader failure

7.3.2. Experimental set-up

The same experimental set-up is used in the discovery, authentication and categorization demo, but when the new laptop joins the mF2C area, there is one leader and three agents. The laptop acting as leader fails (it is disconnected), the new laptop (A_3) becomes the new leader and one of the Raspberries becomes the new backup.

7.3.3. Results

The frontend is used to show the procedure of the demo and the state of the leader in real time. In Figure 46, the failure of the leader is represented graphically as a red cross. When the backup is ready to work as a leader, the frontend changes, showing the new leader and the actual topology, as can be appreciated in Figure 47.

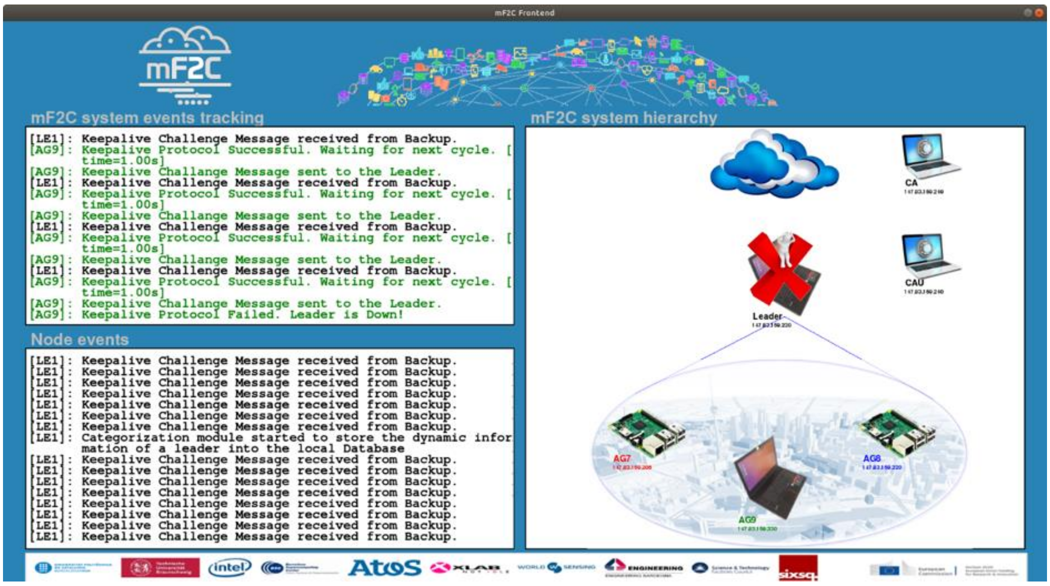

Figure 46 Leader failure in the frontend

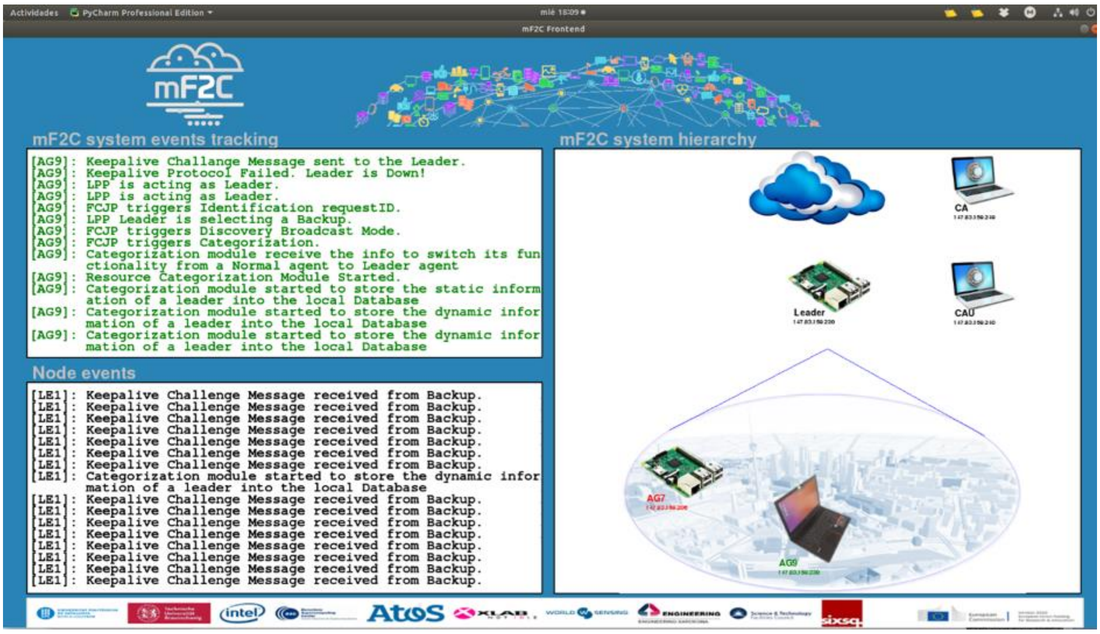

Figure 47 Leader failure, the RaspberryPI becomes the leader

8. Roadmap towards IT-2 mF2C PoC

8.1. Lessons learned from IT-1 mF2C design and implementation usage

The current mF2C system implementation has been purposefully designed to validate the architecture and functionalities defined in previous deliverables, while providing a good-enough foundation for the use cases to validate their demonstrations scenarios. For this reason, and as mentioned above in section 2.1, collaborative development and functionality have been prioritized. As a summary, the following list of pros and cons represent a general feedback to the current (IT-1) state of the mF2C system implementation and design extracted from the experiences gained from use cases development, deployment and integration in IT-1:

Pros:

- easy deployment on any Docker supporting environment
- modular architecture with resilience to component failures
- all basic functionalities implemented
- basic security measures already in place
- successful integration of mF2C specific components with existing software
- integration tests exist for all IT-1 workflows
- public documentation available.

Cons:

- components in different Docker containers not sharing the physical resources, which results in big resource consumption and high machine pre-requisites for executing the mF2C system
- big memory footprint limiting the use of smart IoT devices like RaspberryPI
- current security implementation is incomplete and needs improvement
- lack of automation and boilerplate code in some components and functionalities.

Following next implementation cycle this feedback will be injected into the rest of technical work packages in order to produce the necessary improvements at architectural and implementation levels for the final mF2C software version.

8.2. mF2C PoC extensions for IT-2

For IT-2, the two main goals are to address the cons described above in section 2.1, and to implement both the postponed and newly added components during IT-1.

Event Service:

A new component supporting event queues used by each of the modules to publish/subscribe to events, e.g., service deployed, device added/removed, etc. A first instance of this component will aim to use existing and widely adopted tools like MQTT [12] (http://mqtt.org/) and Mosquitto (https://mosquitto.org/) [13].

Improvements:

All the components that have been postponed for IT-2 shall be implement and in some cases even have their functionality extended:

- the Distributed Query Engine will provide an abstraction layer to access multiple locations of metrics published by the Telemetry Monitoring module
- the SLA Manager will be extended to allow the automatic creation of SLA agreements based on existing templates
- the Allocation component will be introduced, to provide the allocation of available resources to requests to meet security and privacy rules, cost models, guaranteeing overall optimal resources usage

- the Assessment component will be added as the responsible service for checking that the mF2C applications "respect" the sharing model and the profile properties defined by the device's user
- the CA and related authentication components and middleware will be improved in general, allowing for a smoother and more secure authentication flow, while consolidating their robustness and reliability
- the CIMI and DataClay components will converge on a more transparent deployment mechanism, which is able to update data models during runtime and minimize security risks on a distributed environment (like privilege escalation)
- asynchronous job capabilities will be added to CIMI, allowing all components which are currently publishing their interfaces to the host, to redirect their incoming traffic through a single mF2C API (CIMI)
- the Lifecycle Manager and Service Manager will be extended to allow the deployment of multi-tiered and multi-application services
- the Discovery module will aim to support other networking technologies beside Wi-Fi;
- most of the components will be improved in general, both to expand their capabilities and to tighten up their interaction with other mF2C components.

8.3. Demonstration strategy update for IT-2

This section is intended to advance some of the changes and improvements we are expected to consider for the demonstration strategy in IT-2, in order to fully demonstrate the mF2C functionalities. To that end, we assume that IT-2 agent delivery will:

- include all mF2C expected functionalities, particularly in terms of the list of IT-1 assumptions and those architectural components identified as not to be deployed in IT-1
- fix all implementation aspects that constrained the IT-1 demonstration, particularly in terms of required memory, for example easing the development of the agent in small devices.

Similarly to IT-1 demonstration, the envisioned strategy for IT-2 demonstration will also include the set of functionalities running before a service execution (i.e., registration or discovery), although unlike what has been done in IT-1, we expect to run it within the use cases, hence using devices as described in the use cases. Therefore, the agent will be demonstrated in the three different use cases included in the project, considering all defined mF2C functionalities.

Below, we briefly introduce the main expected improvements in terms of IT-2 deployment, IT-2 functionalities and IT-2 use cases.

Regarding the IT-2 deployment, some points may be open to discussion:

- The deployment of the mF2C system in IT-2 is expected to be supported through Docker Compose, as done in IT-1, although some deployment aspects may be open for discussion, such as the link between services and containers and how to widen the compatibility with devices from the Fog and IoT layer that do not support with Docker (like smartphones)
- The agent design may be improved or extended to adopt the functionalities to be embedded into a micro-agent (if necessary), with high chances to be deployed in small devices
- There are no core blocks, as those being deployed in IT-1
- The mF2C architecture will support horizontal communications at leader level
- The proposed security architecture will be completely deployed
- It is worth highlighting that while the main focus for IT-2 has been deployment, IT-2 will also focus on optimization.

Regarding the functionalities for IT-2, it is worth highlighting that the IT-1 list of assumptions and limited functionalities as described in this document in Section 3.1, IT-2 should be significantly edited to remove any assumption that may hinder a solid and stable deployment of the agent, as follows:

- **Interfaces:** CIMI is currently serving as the main mF2C interface and entry point for mF2C users in IT-1. Discussions must be driven to decide on its appropriateness to support IT-2
- **DataClay:** It will be the database used in the mF2C agent and will include the policies for data aggregation as defined in the project
- **Service Orchestration:** The Landscaper and the Recommender blocks will be extended to support IoT devices as well as other attributes for the different devices, thus enriching the final information sent to the Lifecycle. Moreover, the SLA block can be extended to identify SLAs
- **Telemetry Monitoring:** Must include a deeper analytics module to assist the Recommender
- **Resource manager.** Additional policies must be generated, for example to dynamically manage clusters, select leaders or handle a leader failure. Moreover, the categorization of resources may be also improved to accommodate the service needs the system
- **Distributed Execution Runtime.** COMPSs
- **Service manager**: Collect improvements in the service categorization, enabling the system to add some attributes (if needed) beyond those added by the service developer and also enriching the QoS providing functionality, by analyzing the opportunity to consider information different than the one obtained by the SLA manger if possible and realistic
- **User manager**: The sharing module should identify the preliminary steps towards a collaborative model, so, it should first completely identify what the set of shareable resources may be and then feed other components in the agent, thus shaping its performance (for example the clustering policy). Similarly, the profile module must be extended to completely customize services execution and resources management to the users' needs and demands.

Regarding the use cases, IT-2 will follow up with the three use cases as defined in the project. However, since only very limited features were considered in IT-1, pilots for IT-2 will significantly extend the set of proposed features. This will undoubtedly help validate the mF2C deployment into large and much more complex scenarios. Although some insights have been already discussed internally in the project about the pilots IT-2 extent, once IT-2 is triggered in M19, considerable effort will be allocated to make such scenario definition much more concrete. In fact, as the project moves forward and as new assumptions are added for IT-2, the constraints that must be fixed for IT-2, as well as the features to be added, will be identified and solved.

9. Conclusions (and Next Steps)

The deployment of the current mF2C architecture, whose components have been developed within work packages three and four and integrated in task 5.1, has been described and both the challenges and lessons learnt have been reported. The integration was successful in spite of the real-world constraints of actual experimental use cases. Four sets of experiments have been performed successfully using the resulting agent PoC and dedicated testbeds created specifically for the project. The initial tests have been reported in this deliverable and the preliminary results are extremely promising, although issues have been identified, which will represent big challenges for the second half of the project, but corrective measures have already been devised, which is a great step towards a successful IT-2 phase.

The first use case addresses Emergency Situation Management in Smart City and provides an improved solution for intervention management in case of an emergency situation, illustrated by the falling down of a building. In the situation tested in the work described in this deliverable, the latency in sending the alarm message to the emergency vehicles is reduced by 24%. The reliability of the service is also improved through the intrinsic redundancy offered by mF2C architecture and the commercial quality of service offered can thus be increased, improving the competitiveness of the solution. One of the reasons behind the modification of the use case proposed in the last amendment was the possibility to implement the mF2C architecture in the UPC testbed, allowing in this manner a closer collaboration that has led us to an efficient demonstration allowing for numerical KPI to be validated.

The second use case performed during this first iteration deals with Smart Boat scenario in which a boat and a fleet can be monitored locally and remotely by the user and/or the owner. mF2C brings improvements in terms of coverage, as communication can be re-established in dark area in which internet coverage is not available. In the experiments performed during the first iteration of the project, a coverage of above one kilometre has been obtained.

The third use case presents the use case regarding Smart Fog-Hub Service in which a user is provided with a stress-free experience in a crowded environment through the use of an indoor navigator and recommender solution. The challenges are presented, as well as the results achieved. Although the data from the tests is still being collected, the speed of both the computation and the load distribution have already been validated for this application.

In addition, we have presented an extra experimentation to illustrate some of the functionalities that are not yet integrated in the use cases, such as the user registration, identification and categorisation process for user's participation in mF2C system, or the protocol for back-up and substitution process in case of a leader failure.

Business considerations are also addressed for all three real-life scenarios, although it will be investigated in depth in work package 6.

Finally, the roadmap towards the second iteration has been presented, reviewing aspects such as the essential feedback obtained from this first round of integration and experimental tests, the necessary pilot extension and the necessary updates that have been applied to the demonstration strategy that will be applied in that second phase of the project.

References

[1] mF2C, "mF2C Project Deliverables - D2.6", [Online]. Available: http://www.mf2c-project.eu/d2-6-m6/.

[2] mF2C, "mF2C Project Deliverables - D3.3", [Online]. Available: http://www.mf2c-project.eu/d3-3-m9/.

[3] mF2C, "mF2C Project Deliverables - D4.1", [Online]. Available: http://www.mf2c-project.eu/d4-1-m9/.

[4] mF2C, "mF2C Project Deliverables - D4.3", [Online]. Available: http://www.mf2c-project.eu/d4-3-m9/.

[5] mF2C, "mF2C Project Deliverables - D5.1", [Online]. Available: http://www.mf2c-project.eu/d5-1-m9/.

[6] [Online]. Available: http://www.itene.com/blog/i/1863/239/reduccion-de-los-tiempos-de-respuesta-de-los-vehiculos-de-emergencia-a-traves-de-un-sistema-integrado-de-gestion-y-prior

[7] [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X1200054X

[8] [Online]. Available: https://doi.org/10.1016/j.trc.2012.04.004

[9] [Online]. Available: http://www.gtt.com/emergency-response/ – https://www.youtube.com/watch?v=XsrdqEluKtk

[10] US4443783 in 1983

[11] US3881169 in 1975

[12] "MQTT", [Online]. http://mqtt.org/

[13] Eclipse Foundation, " Eclipse Mosquitto An open source MQTT broker", [Online]. https://mosquitto.org/