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

D2.7 mF2C Architecture (IT-2)

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

Work Package	**WP2 Technology Survey, business models and architecture definition**
Due Date:	*M25*
Submission Date:	*30.01.2019*
Version:	*1.0*
Status	*Final*
Author(s):	*Admela Jukan, Jasenka Dizdarevic, Francisco Carpio (TUBS)*
Reviewer(s)	*Xavi Masip (UPC)* *Alexander J. Leckey (INTEL)*

Keywords
Architecture design, mF2C Agent, Microagent, System description

Project co-funded by the European Commission within the Seventh Framework 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)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	30/10/2018	Initial ToC and doc structure	Admela Jukan, Francisco Carpio, Jasenka Dizdarevic (TUBS)
0.2	16/11/2018	Added SLA Management content. Added draft content for QoS Enforcement	Román Sosa (Atos)
0.3	19/11/2018	Added 1.2.3 on UC3	Antonio Salis, Roberto Bulla (ENG)
0.4	19/11/2018	Initial inputs	TUBS, UPC, WOS, ENG, ATOS, BSC, STFC, SIXSQ
0.5	20/11/2018	Missing inputs	TUBS, UPC, XLAB
0.6	24/11/2018	Missing inputs	Alec Leckey (INTEL)
0.7	14/12/2018	Harmonization and integration	Admela Jukan, Francisco Carpio, Jasenka Dizdarevic (TUBS)
0.8	21/12/2018	Comments addressed	All partners
0.9	16/01/2019	Internal review	UPC, Intel
1.0	30/01/2019	Reviewer comments addressed. Quality Check. Deliverable ready for submission	All partners María Teresa García (ATOS)

Table of Contents

List of figures

List of tables

Executive Summary

This document, developed by the mF2C project, represents a second version of the mF2C Architecture deliverable (iteration IT-2). The deliverable offers an overview of the updated mF2C architectural design after adopting the necessary modifications from the architectural design documented in deliverable D2.6 [1].

Based on the experiences from the IT-1 mF2C system practical implementation and prototype integration reported in D5.1 [2], some architectural blocks and functionalities have been modified and extended for IT-2. The focus of this document is identification and description of the main mF2C entities and of the updated components and interfaces within the mF2C architectural building blocks, as well as the identification of different strategies necessary for a successful deployment of mF2C. Based on the challenges faced during the practical implementation it defines with more precision different types of requirements imposed by the applications and services running in the mF2C framework.

The outcome of this document is an updated final mF2C architectural design solution with a detailed functional definition and specification of the mF2C management systems and services.

1. Introduction

1.1. Purpose

This mF2C Architecture document identifies and gives a detailed description of the necessary modifications in architectural requirements and components of the mF2C system and services for IT-2 based on knowledge gained in project execution and in particular during the practical implementation in IT-1. The efforts conducted in the implementation phase in IT-1 turned into a deeper understanding of the mF2C architecture and led to the identification of opportunities and remaining challenges in the architectural design done in IT-1. This document proposes a modified design including proper solutions for the distinct components and blocks. It also puts more focus on some of the technical requirements identified to be necessary for a successful development of the mF2C architecture that have not been in scope of IT-1, such as dynamicity, scalability and mobility requirements.

1.2. Structure of the document

This document is structured in the following way:

- Section 2 describes business goals and technical requirements and challenges imposed by the mF2C framework applications and services, along with a short description of the use cases to be used in the implementation phase.
- Section 3 gives an overview of the global design of the system architecture, including requirements and functionalities already deployed in IT-1, and identifying the ones that will be deployed during IT-2.
- Section 4 follows the general system description from Section 2 with a detailed description of the mF2C Agent. It presents each of its individual components and related functionalities, highlighting the ones that have to be modified for IT-2.
- Section 5 defines characteristics of mF2C cloud environment.
- Section 6 introduces the design of a scaled version of the mF2C standard agent, referred to as the mF2C microagent, envisioned to be used in devices which do not have enough capacity to host a full agent.
- Section 7 defines additional aspects that are important to take into consideration due to the implications they can have on the architecture, such as scalability, and how the architecture scales to the increasing number of devices and resources; software engineering implications and how the new architecture will impact the implementation and adaptability of the architecture in general use cases.
- Section 8 concludes this document.

1.3. Glossary of Acronyms

Acronym	Definition
AC	Agent Controller
AP	Access Point
API	Application Programming Interface
BC	Blockchain
BG	Business Goal
BGR	Business Goal Requirement
CA	Certification Authority
CAU	Control Area Unit
CIMI	Cloud Infrastructure Management Interface
CPU	Central Processing Unit
D	Deliverable

Acronym	Definition
DER	Distributed Execution Runtime
DLT	Distributed Ledger Technology
ESM	Emergency Situation Management
FaaS	Function as a Service
GDPR	General Data Protection Regulation
gRPC	gRPC Remote Procedure Calls
GSSAPI	Generic Security Service Application Program Interface
GUI	Graphical User Interface
HPC	High-Performance Computing
ID	Identifier
IoT	Internet of Things
IT-X	mF2C software release iteration
JSON	JavaScript Object Notation
LoRa	Long Range
MAC	Media Access Control
P2P	Peer-to-peer
PKI	Public Key Infrastructure
PM	Platform Manager
POI	Points of Interest
QoS	Quality of Service
RAM	Random Access Memory
REST	REpresentational State Transfer
RSSI	Received Signal Strength Indicator
SB	Smart Boat
SFHS	Smart Fog-Hub Service
SLA	Service Level Agreement
SSN	Semantic Sensor Network
UC	Use Case

Table 1. Acronyms

2. Requirements and Business Goals

2.1. Business requirements

D2.6 mF2C architecture (IT-1) [1] provides an overview of the fog-to-cloud context, including a set of business goals (BG) to be fulfilled by the mF2C framework. In this second iteration, these business goals have been elicited in a set of business requirements to be covered by the final architectural design.

As a reference we use the tables with the business goals from D2.6, with each business goal descriptive table being followed with the description of the business requirements.

Field	Description
Unique ID	BG1
Short Name	Connectivity
Type	Business Goal
Description	mF2C must reduce bandwidth usage and improve agility via faster response to data. Security must be also improved by reducing the amount of sensitive data to be sent over the network.
Rationale	Extending the cloud to be closer to the “things”.
Involved Stakeholders	IoT Platform / Solution Provider

Table 2. Business Goal 1

ID	DESCRIPTION
BG1R1	mF2C must ensure the protection of all the data stored, analysed and exchanged within the platform.
BG1R2	Reduce latency and bandwidth usage to maintain the QoS while accessing data.
BG1R3	Ensure security and privacy by design.
BG1R4	Develop the appropriate schemes to determine where to analyse the data.

Table 3. Business requirements for Business Goal 1

To fulfil BG1R1, mF2C is taking into account the basic principles of GDPR to manage and operate data, but also to grant to the end user the minimum required rights over it.

Regarding BG1R3, mF2C has taken both principles (security and privacy) into account from early stages in order to provide a security framework [3] [4] fully integrated with the final solution.

For BG1R2 and BG1R4, mF2C framework should include mechanisms able to determine the locality of data processing, i.e., whether to perform the operations in the cloud or at the edge.

Field	Description
Unique ID	BG2
Short Name	Interoperability
Type	Business Goal
Description	mF2C will ensure interoperability by using widely used de-facto and market standards. This is a critical topic for the internal exploitation of the mF2C platform and for a wider adoption of project results. Contributions and/or influences to existing standards, when possible, is another key aspect to be addressed.
Rationale	Ensuring interoperability of the mF2C framework with existing solutions.
Involved Stakeholders	IoT Platform / Solution Provider, Fog Service Provider, Cloud Service Provider

Table 4. Business Goal 2

ID	DESCRIPTION
BG2R1	mF2C must make use of well-known standards, de facto standards and market standards to allow its integration with other existent solutions.
BG2R2	mF2C has to demonstrate its added value for existent fog standards by proposing extensions that include some of its improvements.

Table 5. Business requirements for Business Goal 2

mF2C is using a set of standards, as already documented in D6.2 annual report on dissemination and standardization (Year 1) [5], as well as identifying potential gaps that can be covered by its proposed solution.

Field	Description
Unique ID	BG3

Field	Description
Short Name	Common framework for collaboration
Type	Business Goal
Description	Bridge the gap between cloud and fog by creating a common framework for collaboration and communication that enables businesses to work together and bring cloud capabilities closer to IoT.
Rationale	Integration of IT and OT in a single framework.
Involved Stakeholders	IoT Platform / Solution Provider, Fog Service Provider, Cloud Service Provider, Resource Provider

Table 6. Business Goal 3

ID	DESCRIPTION
BG3R1	mF2C should be agnostic of the IoT domain in order to be applied to different verticals.
BG3R2	mF2C must leverage IoT business domain creating a common framework for developing businesses.

Table 7. Business requirements for Business Goal 3

The rise in Internet of Things is creating a market with a huge potential for solutions like the one proposed by mF2C. On the other hand, Industry 4.0 has pushed Operational Technologies (verticals) to the top, however, they still lack capabilities like the ones that Information Technologies can bring to businesses. Although this is not one of the use cases of the project, mF2C aims at enabling Operational Technologies and business procedures verticals to be easily integrated when configuring Information Technology infrastructure.

Field	Description
Unique ID	BG4
Short Name	Shared and spread
Type	Business Goal

Field	Description
Description	The fog-to-cloud approach of the project is expected to enable efficient and cost-effective solutions, while expanding the scalability of businesses. mF2C opens the door to new revenue streams by enabling the development of a new generation of application and services, more efficient and therefore more cost-effective.
Rationale	Create and add new value to existent business while expanding its scalability.
Involved Stakeholders	IoT Platform / Solution Provider, Fog Service Provider, Cloud Service Provider, Resource Provider

Table 8. Business Goal 4

ID	DESCRIPTION
BG4R1	Support business growth across Europe.
BG4R2	mF2C will aim to be offered under low prices schemes to ease its adoption by small and medium enterprises.
BG4R3	mF2C must demonstrate that it is able to expand the scalability of businesses to maximise its adoption.
BG4R4	mF2C must provide a set of services that demonstrate its benefits.

Table 9. Business requirements for Business Goal 4

mF2C should demonstrate the benefits of its adoption through the three project use cases in a qualitative and quantitative way, showing the distinct benefits of using mF2C in comparison to the system that does not use mF2C. Furthermore, the use cases are asked to provide the improvements in their business models, showing how mF2C has influenced them allowing their scalability and potential growth.

Field	Description
Unique ID	BG5
Short Name	Extension and optimization of current solutions
Type	Business Goal

Field	Description
Description	mF2C will improve the current cloud services portfolio offered by adding fog extensions aiming to reduce operational costs and unnecessary data analysis from a cloud provider perspective.
Rationale	Moving data processing to the edge.
Involved Stakeholders	IoT Platform / Solution Provider, Fog Service Provider, Cloud Service Provider,

Table 10. Business Goal 5

ID	DESCRIPTION
BG5R1	mF2C should show the benefits of the fog-to-cloud paradigm.
BG5R2	The adoption of mF2C should show a significant reduction of operational costs from the provider perspective to increase benefits or to allow them to improve their offerings.

Table 11. Business requirements for Business Goal 5

As with BG4 requirements, mF2C should also demonstrate the benefits not only for the end user but also for the provider, as businesses in fog environments are twofold.

Field	Description
Unique ID	BG6
Short Name	Improve customer experience
Type	Business Goal
Description	mF2C will try to reduce response times by executing applications in the most suitable deployment scheme.
Rationale	Improving reliability and making the process as quickly and smoothly as possible.
Involved Stakeholders	IoT Platform / Solution Provider

Table 12. Business Goal 6

ID	DESCRIPTION
BG6R1	mF2C should improve the execution time when moving executions to the cloud.
BG6R2	Users should not notice any difference independently on where the application is executed.

Table 13. Business requirements for Business Goal 6

mF2C should demonstrate its viability under different circumstances in different environments. This can be done through three project use cases, as each of them belongs to a different domain and has different behaviours. These requirements are directly related with the ones elicited for BG4 and BG5.

2.2. Use cases overview

All three use cases designed for validation purposes of the mF2C architecture are real-world industrial scenarios provided by Worldsensing, XLAB and Engineering and address the following fields, respectively:

- Emergency Situation Management in Smart City (ESM);
- Smart Boat Use Case (SB);
- Smart Fog-Hub Service (SFHS).

The common requirements, as well as the specific requirements of each use case, have to be understood and matched with those of the different applications and services running in the mF2C framework. For instance, all the above scenarios share a common goal, that is to enable real time and highly available solutions in the fog computing domain, as complementary to the current centralized cloud infrastructure.

2.2.1. Use case 1

2.2.1.1. Business goals and requirements

The key business requirements of Emergency Situation Management services are service reliability and quality of service (QoS). To be commercially relevant, the solution needs an availability close to hundred percent, and certainly must be resilient to failures, as not reporting an emergency can have fatal consequences. The service deployed in this use case will greatly benefit from the intrinsic redundancy provided by the mF2C architecture.

The current solution is mainly deployed as a cloud service and, in cases where the connection to the cloud is unavailable either through intentional or accidental reasons, the solution cannot be effective and therefore becomes unreliable. Providing a service reliability superior to 99% would allow the solution to be deployed, albeit with an extra redundant layer. This would represent a quality of service increased by 10%. Also, the service has to be as fast as possible as every second counts. In IT-1, the latency was reduced by approximately 24% as reported in D5.3 [6] and the aim is to reduce it even further in IT-2, to obtain a global improvement of 30%.

In order to guarantee a satisfying service, any installation would require both physical and logical redundancy, to guarantee full functionality in the vast majority of cases. This represents a considerable extra cost in the deployment and use of the solution, both in hardware and in Cloud access fees. The mF2C architecture is expected to provide a reduction of operational expenses, of the order of 10%.

2.2.1.2. Technical goals and requirements

The embodiment of the use case deployed for IT-2 presents an industrial Internet of Things (IoT) application for emergency management, based on five main functionalities:

1. Monitoring: The monitoring has to be able to show real-time data of the emergencies and the location of the closest worker when relevant.
2. Detection: The detection functionality is responsible for handling the inclination data as well as for raising an alarm when the threshold is reached.
3. Positioning: The construction workers will be equipped with location devices. This position will be used to identify which construction workers are closest to the emergency being reported so they can be contacted to check what is really happening and report their diagnostic to validate or disprove the alert.
4. Scanning: The scanning is performed by the proprietary sensor used for this use case when a communication error is reported, to identify whether there is an alert there
5. Intervention: The intervention functionality corresponds to the intervention of the closest construction worker intervention reaction and also to the emergency vehicle path calculation, intervention, and the action on the traffic lights for clearance of the "green corridor" to optimise their intervention time.

The combination of these five functionalities has not been performed so far in a cloud structure and will be made possible by using the mF2C architecture. As can be seen in the figure below, the cloud-only application does not allow for the integration of the positioning and scanning functionalities while the fog-only application does not allow for monitoring. None of them allows for the complete intervention scenario to be deployed. The detection functionalities are the only common element as it is the basis of the whole service. The mF2C application will allow for the complete service to be deployed in an optimum manner.

Functionalities		
Cloud application (phase-1)	**FOG application (phase-2)**	**mF2C application (phase-3)**
*Monitoring	*Monitoring	*Monitoring
*Detection	*Detection	*Detection
*Positioning	*Positioning	*Positioning
*Scanning	*Scanning	*Scanning
*Intervention (limited only global)	*Intervention (limited only local)	*Intervention

Figure 1: Emergency Situation Management main functionalities

The storyline of the use case for IT-2 is as follows: the location of construction workers could be either displayed on dashboard and maps or stored for use only in the event of an emergency. Both these cases will be evaluated to ensure compliance with the GDPR. The tiltmeter (also proprietary sensor) regularly sends data to the Gateway, that forwards it to the Monitoring Software, which shows the data on the dashboard and maps. Meanwhile, the jammer detector is idle. Once the Monitoring Software detects that a tiltmeter threshold was exceeded, the response plan is activated. First, the sirens are started. Then, the closest worker or actor of the use case is identified through the location information stored previously and contacted, so he or she can go and check the emergency. He or she will then report to confirm or cancel the emergency. The authorities and the relevant actors are alerted, and the alerts visualized on the dashboard and maps in real time. The warnings are emitted, and the emergency vehicles are sent, through the optimum path calculated at that moment. Finally,

the traffic lights are changed to green on the path of the emergency vehicles in order to optimise the intervention time.

2.2.1.3. Expected challenges and requirements on the mF2C architecture IT-2

One of the main challenges regarding the deployment of the use case for IT-2 will be the indoor localization technology. The current commercial solutions are generally inaccurate. Also, in case of emergency in construction and in buildings, we need to identify the location in 3D. For this reason, this use case will rely on a localization technique similar to that deployed in use case 3, to be able to identify which construction worker is located closer to the emergency so it can be confirmed before the whole intervention process is initiated.

Another technical challenge might be a reliable messaging protocol to alert workers and actors of the use case even in the event that the internet or mobile phone protocol is affected by the alarm and cannot be used. There is a number of solutions, such as Bluetooth messaging applications that can be used. These will be reviewed and the most appropriate one will be selected.

Finally, this use case requires the mF2C architecture to provide a failure resilient architecture, and in particular a resilient leader and agent election, and a fast replacement procedure, in consideration of locality of construction workers equipped with devices.

2.2.2.Use case 2

2.2.2.1. Business Goals and requirements

Technology nowadays allow users to monitor their belongings more than ever before. In case of Smart Boats, this is even more important, because the marine goods as boats and other marine services are very expensive to buy and maintain. Our goal is to provide an efficient and reliable solution called SmartBoat to boat owners and sailors.

To fulfil the business requirements, the technology must provide a solution that can provide seamless monitoring system over the various different network scenarios that can occur on the boat in the sense of network coverage and capacity. The Fog solution brings a major improvement over the Cloud-IoT solution due to the powerful processing component on the edge, capable of providing services to the sailors also in situations when the connectivity to the cloud is not available. This difference presents an important niche and advantage over the current solutions on the market.

2.2.2.2. Technical goals and requirements

The overall architecture layout remains unchanged from IT1. Figure 2 shows the components distributed and mapped onto mF2C layers to fit with the overall picture of the platform.

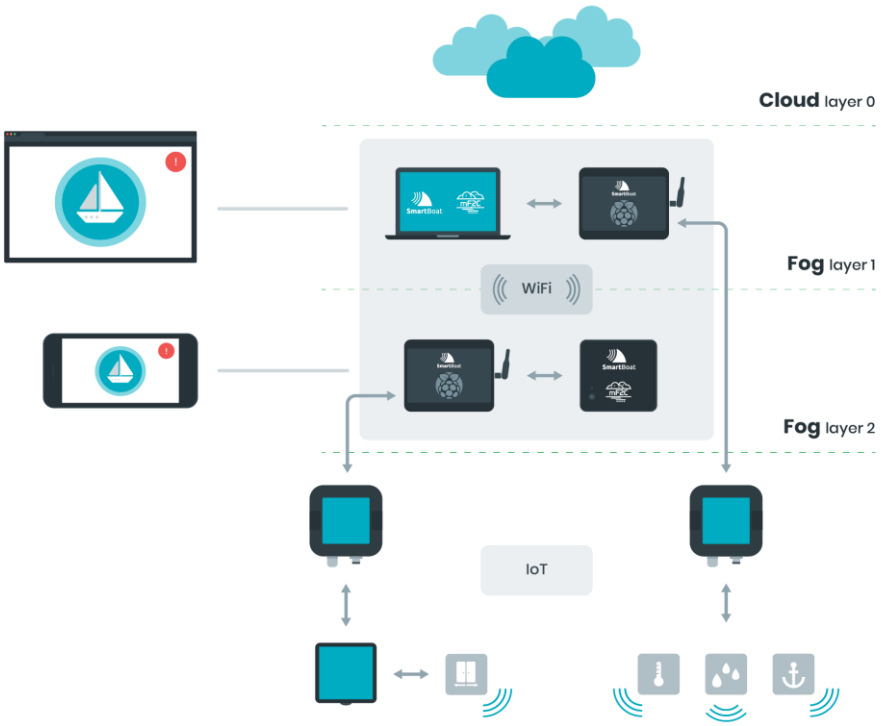

Figure 2: Smart Boat to mF2C layer mapping

Some of the functionalities this architectural topology will enable are:

- a more capable device acting as a computation offload for a child,
- data sharing, e.g. the ability to display temperatures at other boats' locations at sea,
- transparently connecting to either a fog node or a cloud node and receiving the same functionalities, and
- alternative networking functionalities supporting long-range communications.

A potential scenario is a boat A on the open sea without a connection to the Internet - the boat is only equipped with WiFi and LoRa transceivers. mF2C and the Smart Boat application are deployed on a device on the boat and both share local data with limited functionality to users connected directly to the device. Sensors collect boat status and parameters and store them in a local cache, which is used by the local application to display data.

Boat B then arrives within a close proximity to boat A. Boat B has a mobile network connection, which provides Internet access, and can thus use the full cloud-enabled functionalities of the platform, including transmitting data to the cloud or accessing data both through the local application or through a global, cloud-hosted interface.

This scenario is similar to scenarios 3 and 4 in Figure 3, which represents different deployment settings and combinations for inter-boat communication.

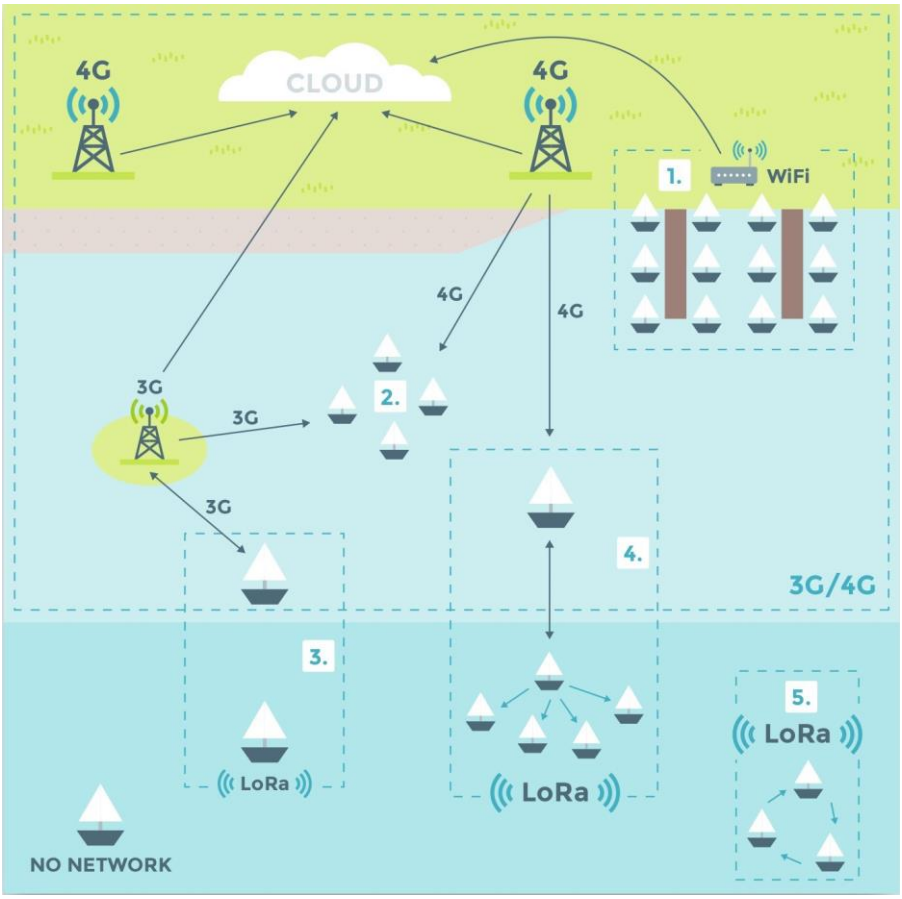

Figure 3: Smart Boat networking scenarios

Because mF2C, with the Smart Boat application, is installed on both boats, they discover each other and create a fog area via a short-distance wireless connection. The Smart Boat application can take advantage of this connection and provide functionality to boat A through boat B by sharing data and executing some jobs through COMPSs. This peer-to-peer data sharing is an essential functionality to enable a true seamless fog application experience.

Finally, when boats move away from each other, application-specific LoRa modules establish a connection between the boats to further extend the coverage. While this may be limited because of the limited bandwidth of the connection, some functionality can be provided in addition to a local-only application, for example the ability to send short messages between boats through LoRa. Because of the inherent long-range communications, this will improve communication between vessels, reduce the need to follow protocols when communicating via radio and enable additional functionalities due to the machine-readable communication in text.

2.2.2.3. Expected challenges and requirements on the mF2C architecture IT-2

There are certain obstacles we need to overcome in IT2 which are specific to this use case. The first one is the issue of data synchronisation. A rudimentary implementation of data sharing was introduced in IT1, but it will need to be extended to more complex scenarios, which would include multiple hops of sharing over LoRa.

This LoRa connection will need to be extended from a proof-of-concept into a stable interface for reliable or unreliable transmission of messages, depending on the usage. This will present numerous challenges akin to designing a communication protocol.

In addition, not only on the lower layers of communication, the application will need to be able to work with intermittent, low-bandwidth and unstable connections. The data sharing protocol will have to be designed in such a way that will allow boats to privately share others' data, maintaining network usage efficiency. Pending this, functionality of the application without a connection to the cloud, both connected directly to the lone boat or cloud, will be expanded.

2.2.3.Use case 3

2.2.3.1. Business goals and requirements

There is an increasing demand for identifying and evaluating new market sectors, opportunities and interest in the IoT as a potential arena where current Cloud offering could be enriched and differentiated. This can be seen in set-up hubs in public environments (e.g. airports, train/metro stations, malls and related parking areas, harbours), 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 the path/behaviour of consumers, and making real time decisions.

The Use Case 3, Smart Fog Hub Service in Airports, starts from a customer perspective, focusing on the devices that move throughout the airport (like smartphones, tablets, etc.). Customers – both travellers and visitors - have specific individual needs and/or interests (about their flight, shops, fast food and services nearby, entertainment in general, emergency situations) and generate lots of real time data. The fog helps aggregating and managing the growing number of devices, with resources usage and optimal response time. In case of increasing load, in order to fulfil real time responses, fog nodes can recursively ask for resources at the same or higher level, up to the cloud. The mobile application specifically developed as part of the hub service takes care of customer data privacy and security, with MAC address personal information being hashed and then used as device identifier and primary key.

The goal of this use case is to improve the travellers' 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 traveller towards flights that he/she has selected as relevant (e.g. the flight he/she has to take). The use case will be connected to the airport management system to receive and share flight information to the interested users. The administrative dashboard can be used to define additional information and suggestions like promotions or different alerts to be sent to customers, and visualization and analysis of data related to travellers (e.g. different reports and maps for most visited places).

2.2.3.2. Technical goals and requirements

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

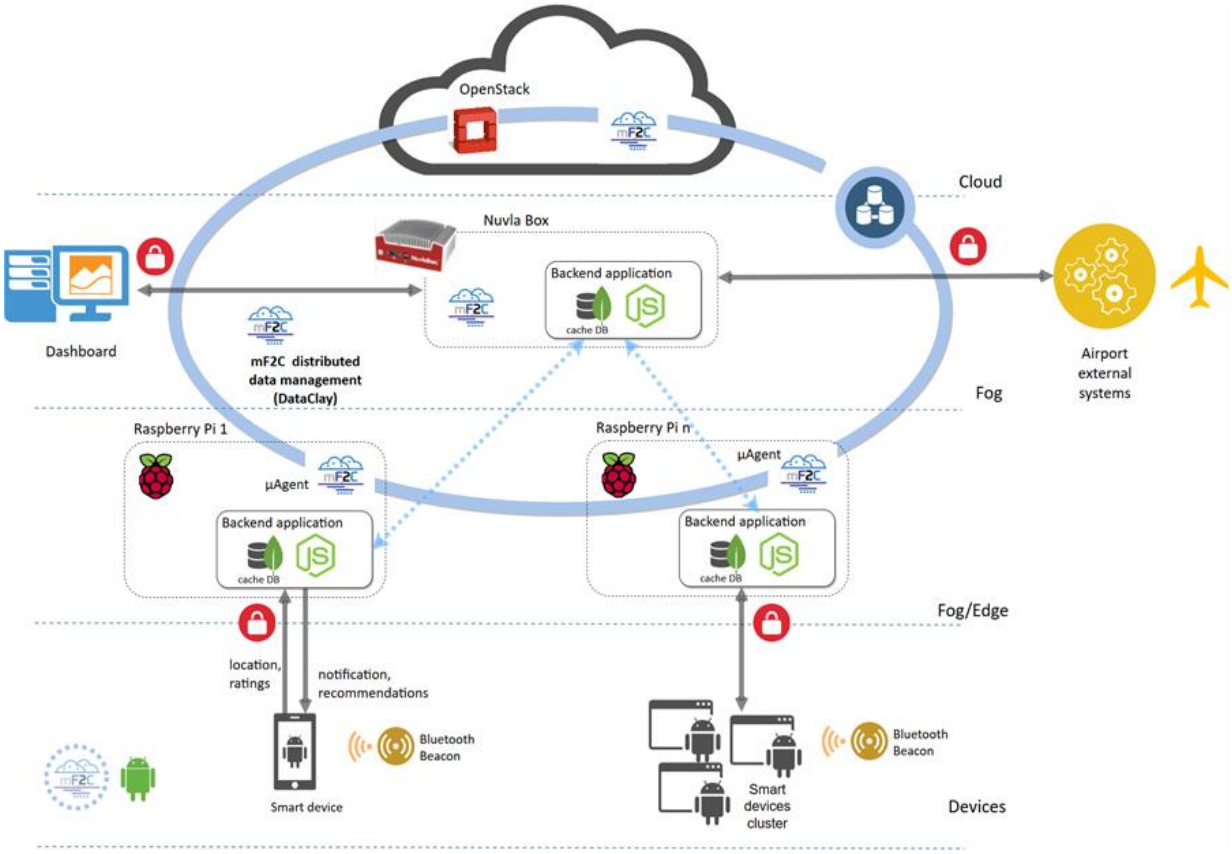

Figure 4: UC3 final system architecture

During IT-1, a major effort in implementing the base functions of the use case has been made, and some tests have been performed with different system architectures, allowing for the gathering of significant information that would be helpful in improving the system performance and resource usage. In this preliminary phase, the mF2C agent has shown adequate results in processing and load distribution. The fog-to-cloud approach has been managed completely by the mF2C agent.

During IT-2 the use case will be moved to the Cagliari Elmas airport. Several features will be added in the user's app, and the dashboard will be extended with more reports and diagrams. The traveller advisor system, based on Machine Learning, will be developed and integrated with the app. The accuracy and stability of the position calculation algorithm will be improved evaluating the use of sensors on-board of smartphones and/or digital filters, which is expected to be validated in the testbed. Also, a particular importance will be on the privacy and security of personal data: the traveller advisor system uses algorithms that operate successfully without any personal information.

One of the main goals of this use case is to ensure efficient management of continuous stream of data (position and behaviour) produced by the set of IoT devices (smartphones) in the airport, that needs to be processed in real-time.

These data can generate a feedback to the user with 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 machine learning tasks to produce advises.

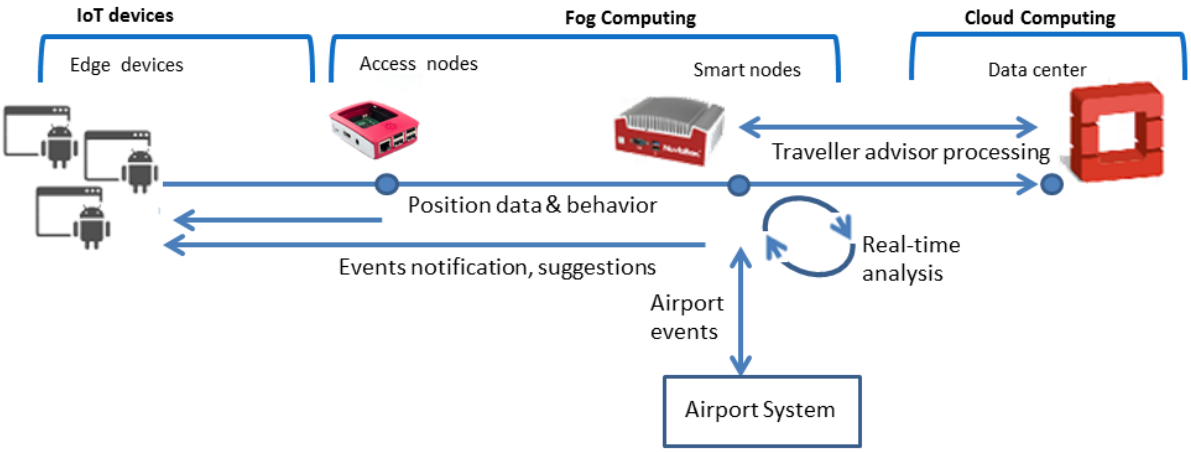

Figure 5: UC3 dataflow between different layers

The use of a traveller advisor system implies massive data processing that increases with the number of managed objects. Since a national airport like Cagliari Elmas manages about 4 million passengers per year, a relevant amount of data like 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 allows for the discovery of bottlenecks in airport services or suggestions on how to improve available resources. At the same time, service providers in the airport site will benefit from the application in their marketing proposals by using the anonymized data. For these reasons, the traveller advisor system can facilitate the win-win goal (travellers experience, shoppers' revenues, suggestions for airport planners).

During the second iteration, an effort will be done to define some performance metrics to be applied at both system (mF2C) and application level, that could be useful to determine the benefits in terms of reduced latency and load distribution in case of overloading. These metrics will help the final validation of the Use Case and the mF2C infrastructure.

2.2.3.3. Expected challenges and requirements on the mF2C architecture IT-2

The first challenge behind the deployment of the use case in airports is the indoor positioning system based on the measurement of the Access Points (AP) Wi-Fi radio signal strength in a given point. For this reason, the device, whose position we want to know, must be able to perform AP Wi-Fi scan (e.g. an Android smartphone) and measure the signal strength of each AP detected in the scan (i.e. RSSI, Received Signal Strength Indicator).

Once these measurements are made, the device position is computed and delivered to a backend which stores this data in a database for further processing.

The possibility to track the position of all travellers in real-time gives an opportunity to build value added services on top. These are:

- Proximity Marketing: The mF2C agent is expected to support the calculation of Points of Interest (POI) nearby each user in the field. This happens about every second for all users, which means a huge amount of processing with strict real-time boundaries. This seamlessly demands an intelligent distribution of processing, leveraging the Fog-to-Cloud approach, provided by the Distributed Execution Runtime (DER) in the mF2C agent;
- Traveller Advisor system: that requires the use of Machine Learning algorithms to determine similarities between users, so it would be possible to propose for every user some places that

similar users liked the most. As the processing gets too difficult to be managed at a fog level, it needs to be run offline in the cloud based on a cache of the result managed in the fog;

- Data Analysis: this 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 to the cloud. This information, coming from real utilization, gives the opportunity to analyse, even in real-time, these data, allowing to use the service as a planning tool for determining the number and distribution of people that use, or can potentially use, various services.

As additional challenge, an evaluation will be performed for the redesign of the proximity service based on microservices and the use of FaaS/Serverless technology.

To support this kind of a realistic scenario the system needs to support the distribution, resilience through system redundancy, and scaling of processing as the number of travellers grows, while at a same time guaranteeing a real-time response time. For these requirements, the use case will benefit from the orchestration features in the architecture, automatic scaling and distribution of processing according to defined resource and service policies provided by the mF2C architecture. It is expected to improve the overall response times and resource usage by 15-20% compared to full IoT-Cloud direct connection.

2.3. Technical requirements

The set of technical requirements that have to be addressed for a successful development of the mF2C architecture remains similar to the requirements from the IT-1, with the emphasis on the scalability and mobility implications, which while implicitly addressed in IT-1, will have a greater impact on IT-2. These requirements include:

- A **Coordinated orchestration** is required to generate individual service workflows, and to map these service workflows into devices best suited for each of the requested services, and to coordinate the interactions among the different devices involved in the service execution. All this coordination must be **seamless-transparent** to the user or the service, as well as, the coordination of physically allocated resources.
- The Proposed mF2C platform must enable a timely creation of **dynamically provisioned F2C infrastructure**, automatic **discovery** of resources, context-based decision-making, stateless communication and transparent connectivity. The dynamic provisioning also requires an adequate **service scheduling** to decide how service's individual functions are split into the different devices and mapped into different physical resources, and even dynamically managing schedules based on runtime conditions. This kind of infrastructure requires a **high-performance service execution** through parallel computing.
- Appropriate **runtime system** for this new service execution paradigm shall be considered to optimise the characteristics of the resources brought by the combination of fog and cloud computing.
- **Security and privacy** in mF2C - which incorporates security by design - needs to address critical security requirements for coordinating and managing distributed F2C components. Unlike in IT-1 the security and privacy will no longer be addressed per each individual block within the architecture, but rather as a feature of mF2C system main architectural building component - the mF2C Agent. Because of the complexity of this requirement a separate deliverable D2.5 [4] addresses security and privacy for the whole architecture in more detail.
- Proposed mF2C platform must be able to address the **mobility** and **scalability implications** on service execution, interaction and communication among the different device.

3. mF2C architecture for IT-2

This section shows the global design of the system architecture, indicating requirements and functionalities already deployed in IT-1, and the ones that will be deployed during IT-2. Beyond existing weaknesses and gaps detected during IT-1 (some of them explicitly moved to IT-2), we mainly discuss the multi-cloud support that will be added for IT-2 and what the implications on the architecture are when considering dynamicity of resource availability. The main questions we take into consideration are how to address dynamic discovery, what are the implications when executing services on other agents, how to monitor resources, the implications when QoS has to be maintained and how it affects the application execution runtime and finally, what are the policies that we define for the resource manager. These questions are first addressed from a general point of view, and, in the following subsections, details are given from the perspective of each involved module. Finally, we show security and privacy requirements for the intercommunication between agents.

3.1. mF2C system description

The mF2C system proposes a coordinated management solution capable of leveraging all existing and potentially available resources, from the edge up to the cloud, when executing a service. Figure 6 shows a typical scenario where different kinds of devices interconnect at different layers from the lowest, internet of things, up to the cloud. When a service wants to be executed using the set of available resources that are made available in a distributed fashion, mF2C is expected to properly manage the execution according to the service requirements and user needs. To that end, the mF2C system proposes a layered architecture (described next) where the resources are categorized according to a certain policy, using an agent entity to deploy the management functionalities in every component within the system.

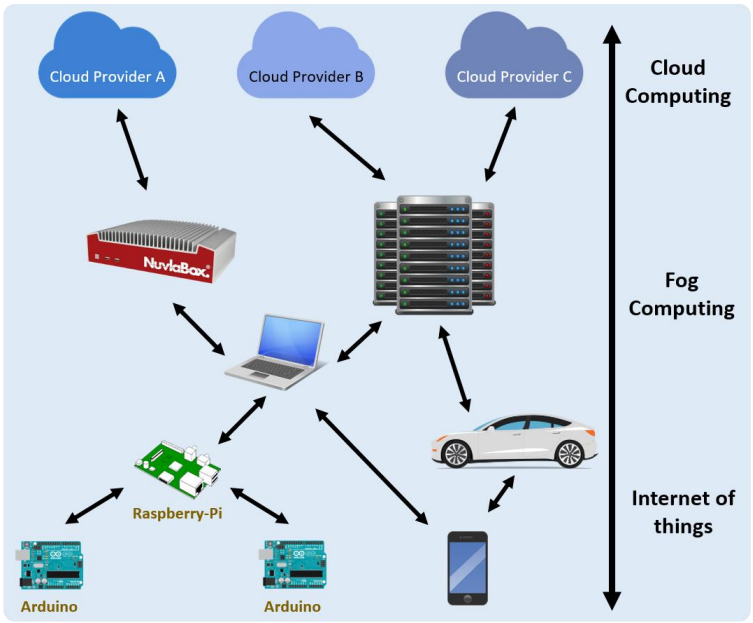

Figure 6: Set of available resources

Figure 7 shows the main characteristics of the layered architecture built for mF2C. The architecture is divided into different logical layers: from layer 0, at cloud, to layer N+1, closer to the edge. Three different kind of software entities are deployed for mF2C: agent, cloud agent and microagent. The agent is the one used by default in most of the devices of the architecture, the cloud agent is a slightly

modified version of the standard agent adapted for the cloud, and the microagent is a simplified version of the agent designed to be used in constrained devices.

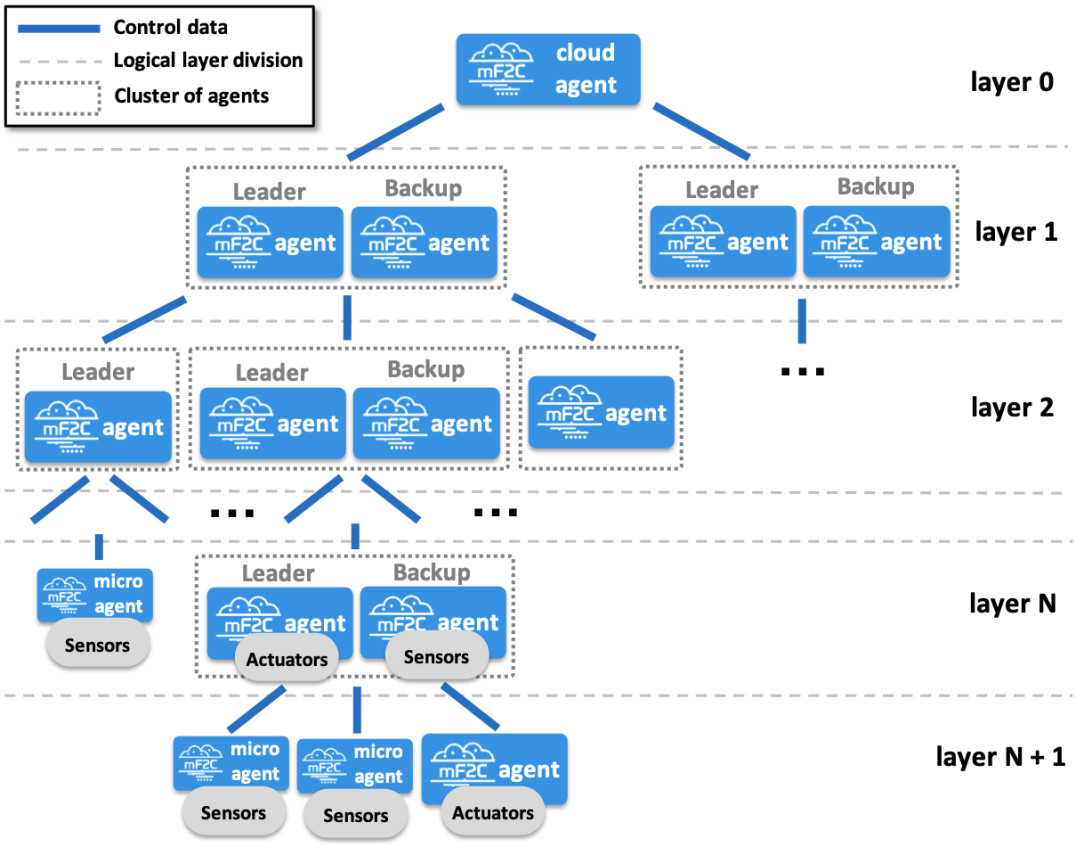

Figure 7: mF2C layered architecture

The cloud agent can be instantiated over one or multiple private or public clouds according to the specific requirements of the user. Below layer 0, the instantiation of multiple agents will start with the creation of a layered mF2C architecture. In each layer the agents can be clustered according to different policies. From this representation of the mF2C architecture in Figure 7 we can see that in each layer, multiple clusters of agents can be set up, having at least one leader and if possible one backup. The policies for determining whether an agent can be leader or not, will depend on resource capabilities, for instance taking into account the amount of RAM memory, internet connectivity, etc. In the last layer of every branch we find all those agents which do not meet the constraints for any specific layer as well as microagents. Policies in place will also define the number of layers required, creating a more vertical or more horizontal architecture depending on the specific scenario. How the architecture changes scalability wise depending on the scenario is later explained in more details in Section 6.1, while the specific policies is an ongoing work to be reported in future deliverables.

The microagent can be placed in any layer in the architecture, but without the possibility of managing other agents, acting as a leaf in a tree architecture. On any agent it is possible to attach some sensors or actuators with no computing capabilities, that mF2C will manage only considering their capacities and characteristics through categorization and classification mechanisms as well as through abstraction or virtualization strategies. In the proposed architecture, the information regarding the attached sensors and actuators is gathered and distributed by the connected agent and shared with agents in upper layers. While mF2C will support some sensors, integration of sensing into cloud and fog computing is not the main purpose of this project and has been addressed before; nevertheless, mF2C offers the possibility to the developer to tag new sensors and actuators into the system, to use

them to create new services, and to map service into resources for service execution. Only the final inclusion of the driver will be left to the developer, as well as application to read the sensor data or to execute commands on actuators.

After having the architecture established, when an agent receives a request for service execution, regardless of the layer, the agent decides where it should be executed. If the requested agent has the required resources itself, the service will be executed in that agent; if not, it will be forwarded to the leader in the layer above. In case the service execution arrives to an agent which controls multiple other agents within lower layers, the agent in question will try to allocate the service using those resources first or will forward the request to the upper layer in the hierarchy, in case there are no available resources for the proper service execution.

3.2. mF2C architecture specifications

In IT-2, we move from the necessary architecture assumptions we made for IT-1 to the final mF2C architecture specifications as a stable and real Proof-of-Concept of the mF2C concept. These final specifications are more complex and not restrictive as those considered in IT-1, fulfilling the requirements for the real scenario envisioned for mF2C and the final architecture described in the previous section. Next, we extend the list of specifications already reported in IT-1:

Architecture

- The mF2C architecture contains N logical layers, with N depending on the requirements of the scenario. When horizontal scalability is required, fewer layers are used, when vertical scalability is needed, more layers are created.
- There is no direct horizontal communication among agents in the same layer at control level. However, in case of having multiple leader/backup agents per cluster, the states will be synchronized among them in order to maintain statefulness of the system.
- Mobility requirements are considered.

Agents

- The Agent is instantiated in all devices that are sufficiently capable.
- When the device is too constrained to run the agent, a light version is used instead, called Microagent.
- For the cloud, a slightly modified version of the agent is used, called Cloud Agent.
- One or more agents can be clustered.
- Multiple clusters of agents can belong to the same layer.
- Each cluster has at least one leader, if there are agents dependent on it.

Leader/Backup agent election

- When having a cluster with more than one agent, they can be configured as leader or backup agents.
- All leader agents are able to manage service requests and their states are synchronized.
- Backup agents cannot manage service requests, but their states are synchronized with the leader agents.
- New policies are going to be defined for the creation of clusters and selection of leader and backup agents.

Services

- Services are executed from the mF2C dashboard, that is available from the interface of the agent. This GUI is accessible in all agents locally through the web browser and allows visualization of all the services in the form of portfolio of services that are reachable by the user logged in.

- Unlike IT-1, in IT-2 the set of categories for services and resources are not limited but are left for the developer to define them.
- QoS functionalities are separated into two different modules: QoS providing, to report on past SLA violations, and QoS enforcing, to deploy solutions to meet QoS in runtime.

3.3. Discussion on dynamicity of resource availability impact

For IT-2 we assume that all devices running an agent, can join and leave the coverage area of a leader dynamically. This assumption was implicitly addressed in IT-1, however, some practical aspects were not completely considered. Hence, dynamicity has different implications in the resources management specifically impacting on discovery, execution of services using different agents (inter-agent service execution), monitoring of resources, QoS management, and execution runtime, as detailed in next subsections. It is also worth mentioning the fact that some implications about mobility can affect specifications explained in Section 3.2. In IT-1 we assumed that mobility was only considered in the lowest layer (layer of agents). Therefore, leaders were considered static, without movement. For IT-2, we consider that the number of layers is not limited to three. Then, with more layers, different leaders that belong to different clusters in different layers can act as leader agents for one layer and as standard agent for another layer. In this case, if a leader leaves an area, a backup agent will become leader, while at the same time a new backup will be selected.

3.3.1. Dynamic discovery

In IT-1 when a device running the agent gets into an mF2C area, it will start scanning to find out the leader in that specific area. Once the leader is detected, the new agent joins the area, and it stops the scanning function. However, if mobility is considered, this procedure may be no longer appropriate and thus additional cases must be considered:

1. No leader or backup agents geographically moving across different areas, joining and leaving different cluster parents.
2. Leader or backup agents leaving a cluster because of mobility or failure.

For any of the previous cases, the scan done by the agent has to be re-triggered in order to update the state of the architecture. As we will detail in Section 4.1.1, the discovery module is able to detect, by means of network detection, if agents leave an area.

The set of potential agents eligible to be either leaders or backups are, based on their mobility behaviour, ranking higher those that are more static and have better availability and discarding those that are transitory. These parameters should be defined as policies that will be later used by other components.

3.3.2. Mobility implications in the inter-agent service execution

The mobility of mF2C agents and how it affects the execution of the services was postponed for IT-2. This mobility can have a great impact on the way the running services are handled. For instance, if an agent that is executing a service instance leaves the cluster, then a set of policies need be applied in order to minimize the resulting impact. In order to address the dynamicity of the system, some restrictions and rules must be added as policies to solve some of the drawbacks of the envisioned new scenario. Indeed, IT-2 explicitly considers mobility, which turns into additional concerns, characteristics as well as new policies to modify the behaviour of other components. These policies define the type of notifications and alerts that are sent to other agents, and the actions they should take.

The mobility of these agents and how it affects the inter-agent service execution, only applies to agents that are executing a service instance. Certainly, depending on the type of services that are running on these devices, we face the following scenarios when an agent disconnects, leaves the cluster, or cannot be reached by the leader:

- The agent is running a COMPSs application
- The agent is running a service required by other services running in other agents of the cluster
- The agent is running another type of service

When possible, the policies should define a way to restore or redeploy the application in another agent; they should also define a way to recover the information generated by the "disconnected" agent.

3.3.3. Dynamic resource monitoring/categorization

As it is detailed in sections 1 and 4.1, some of the proposed blocks for IT-1 have changed in the proposed architecture for IT-2, with monitoring block being one of them. For IT-2, the monitoring functionality is embedded into the categorization block, responsible for both updating the leader's database with the static information of a dependent agent in the lower layer when it joins the leader's area; and also, periodically updating this database with dynamic information. Besides, the leader also aggregates the dynamic information from all its dependent agents periodically, to be shared with higher layers. In this last case, this aggregated information includes only the information of the dependent agents in the area. If one of the agents has left or it is disconnected, the information of this agent will not be in the leader's database anymore.

3.3.4. Mobility implications for QoS management

Mobility, in the lowest layers of the mF2C architecture, affects the service execution when agents join and leave the system constantly. When this behaviour occurs during the execution of a service, the quality of service is affected. In order to overcome this problem, mF2C will offer two mechanisms: QoS provisioning, which tries to maintain QoS before a service is executed, and QoS enforcement, which tries to maintain QoS during service execution.

QoS provisioning will try to avoid agents that were not suitable in past executions, for example, because they were too volatile. When this happens, the system will launch service violations that will be used to measure how reliable the agents within a cluster are. To achieve that, the leader agent will maintain a mapping between executions of a specific service in the past, and the number of agents that were executing some tasks of that service. If some service start failing, mF2C will try to figure out which was the agent causing this problem and will try to remove it for future executions of that specific service.

When a service already running is expected to fail because agents are leaving the cluster, the QoS enforcement will take care of adding new resources to the service execution in order to fulfil the SLA agreement, where, for instance, the maximum execution time of a service is specified. To do that, the QoS enforcement will be responsible for requesting new available resources when some SLA violation is expected to happen, in order to avoid it. By using these two mechanisms together, mF2C will try to handle mobility issues regarding the QoS.

3.3.5. Application execution runtime

During the execution of a service, one of the agents used for computing a part of the application workload may disappear. In these cases, the agent hosting the execution of the main code of the application (master node), which is the responsible for orchestrating the execution of the tasks on the other agents, has to reassign the tasks running on the failed agent to another agent.

4. Agent architecture design

Based on the mF2C system description and main architecture requirements and functionalities defined in Section 2, we follow with the description of the main building block - mF2C Agent and each of its individual components and related functionalities. It should be noted that during the practical implementation of the mF2C system in IT-1, some components and functionalities have been found to be obsolete, while others have to be modified for IT-2.

The main entity within the mF2C architecture is the mF2C agent, deployed in all the resources with sufficient computing capacity. Unlike in IT-1 where the agent structure was the same independently of which layer it is located, for IT-2 there is distinction between a full-stack mF2C agent, an optimised mF2C microagent and mF2C cloud Agent. Figure 8 shows the main building blocks that compose the agent entity: Platform Manager (PM), Agent Controller (AC), dataClay as a data management related component, Security block comprised of Control Area Unit (CAU) Client, Reverse Proxy and AC library providing an agent with standard functionalities like cryptography, etc., an Event Manager, GUI and an API as an entry point. The architecture for mF2C microagent differs significantly from full-stack agent and its components and related functionalities will be defined in the Section 6. The same case is with the cloud agent, described in the Section 5.

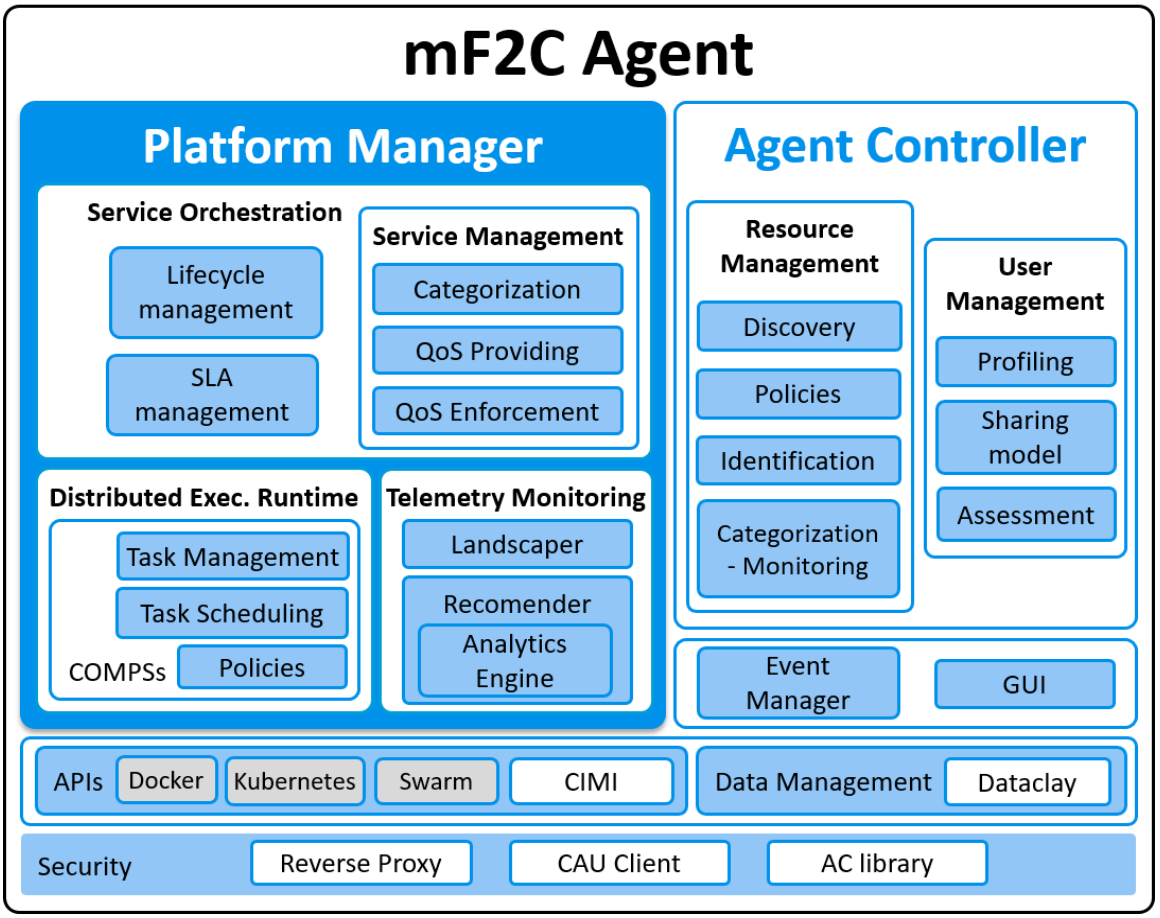

Figure 8: mF2C Agent architecture

As in the high-level definition from IT-1, the Platform Manager component is a global entity which works as a controller for agents in lower layers, and a receiver of control data, when it is being managed by agents from upper layers. It is in charge of service orchestration, monitoring of telemetry

data from different sources and the coordination of how end-user applications will be executed within the mF2C infrastructure with its distributed execution runtime block. However, some blocks and functionalities of the PM must be modified for IT-2, which is described in more details in Section 4.2. For the other IT-1 entity - the Agent Controller, the high-level definition according to which its main responsibilities included managing the local resources, services and users of each individual agent, is modified for IT-2. The task of managing services is completely left to the Platform Manager, as can be seen in the Figure 8. Now, it encompasses all functionalities dealing with the resource and user management of local resources, and this information is then shared with the platform manager. The definition of all of the blocks and functionalities of AC is given in the Section 4.1. As can be seen in Figure 8, the new mF2C architecture includes additional components, as well as some of the functionalities from AC and PM modified and positioned as separate blocks. The role of the Data Management as the component responsible for organizing all mF2C system data resources and providing an interface for accessing such data remains the same as in IT-1. But, while in IT-1 these functionalities were split into two blocks, one in PM and one in AC, in the new mF2C architecture it is positioned as an independent component, described in Section 4.3. During the IT-1 implementation phase we concluded that it was also important to include an event tracking module - Event Manager in the mF2C agent. This module described in Section 4.4 represents a broker that will be used by each of the modules to publish/subscribe to events, e.g., service deployed, device added/removed, etc. The main changes in the IT-2 mF2C agent architecture compared to IT-1 are shown in Figure 9. The red modules have been found to be obsolete during the implementation and are no longer part of the architecture, while the green labelled ones are the new modules that are going to be implemented in IT-2. The black arrows represent the changes of the original placement of some of the modules in PM and AC from IT-1 to IT-2.

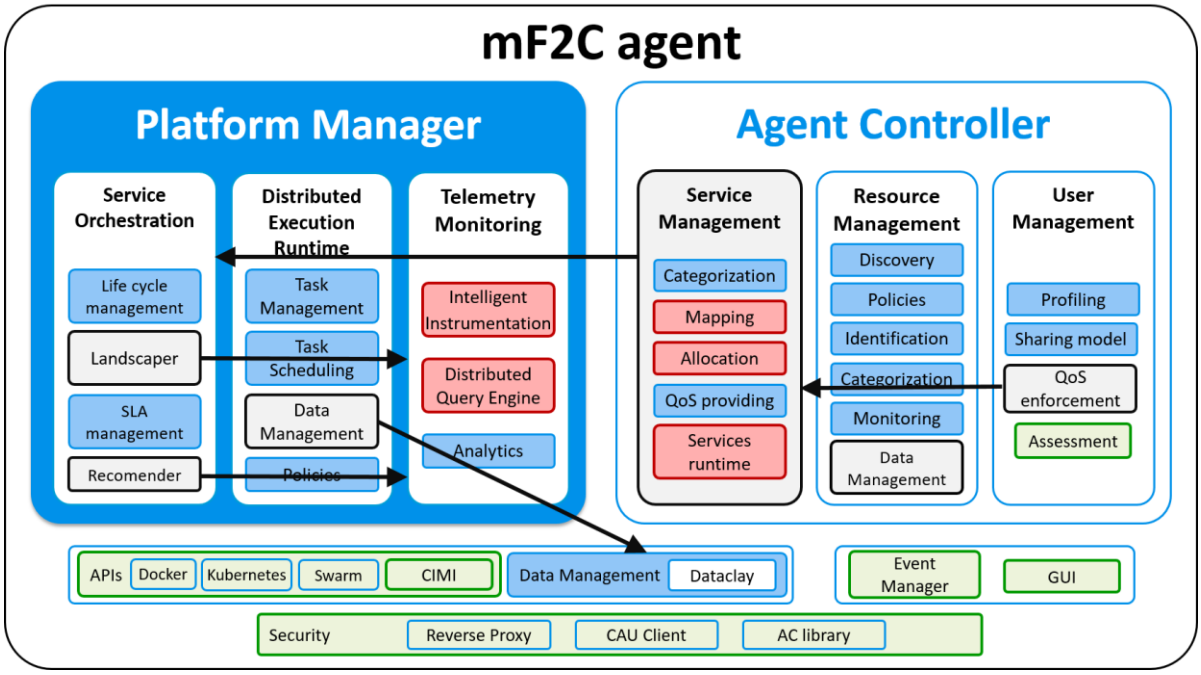

Figure 9: mF2C Agent architecture main changes compared to IT-1

The rest of the section includes the description of the agent detailing each individual block from Figure 8 and all the related functionalities.

4.1. Agent Controller

Over IT-1, we considered that the Agent Controller was in charge of managing local resources, services and users. For this reason, the main blocks inside the agent controller were (as reported in D2.6 [1]):

Resource Manager, Service Manager, and User Manager. We considered a different meaning for "local" when considering resources, services or users. For local resources, the Agent Controller is in charge of its own resources (of the device where the agent is installed) if it is a normal agent, but also in charge of the resources of its dependent agents when it plays a role of a leader. This assumption of local resources is the same for IT-2. In D2.6 (IT-1) when considering services, we assumed that local services are the ones executed in the own resources (in the device where the agent is installed). However, during IT-1 we realized that a service may not be executed in a single device (agent), but it can be split across different devices (such as it is the case of the tasks for COMPSs applications). Hence, the Service Manager should be a block dealing with the whole service, not only with the part of the service executed in a specific device. For this reason, we moved the Service Manager to the Platform Manager block, since it is a global entity working as a controller. Finally, in the case of the users, for IT-1 (D2.6), although conceptually we considered that users could be at the same time mF2C clients (requesting resources), and providers (providing resources), in a first approach we simplified the possible cases, and we considered that a user was the one requesting the service execution, and at the same time providing his/her only device to the system. This scenario is enriched in IT-2, for example let us consider the case where a user has more than one device. This new scenario requires modifications in the SLA, between the user and the system, the Profiling and the Assessment modules. In fact, a user will need an SLA as consumer (with his/her own requirements when executing a service); as well as an SLA as provider (with his/her own requirements when allowing to execute a service in his/her device). In the case of the profile, when a user only has a single device, the profile could contain the information of that user and that device, but when different devices belong to the same user, the information stored in the profile should change, being necessary a profile of the user (possibly stored in cloud and updated when the user is registered), and other profile for each one of his/her specific devices (possible stored in the own device); and both profiles should be related.

Finally, it is also worth mentioning that for IT-2 we discard from the Resource Manager the Monitoring module, since the functionalities included in this module are already embedded in the categorization module.

4.1.1. Resource management

Discovery

In this subsection, we review the definition of the discovery module provided in D2.6 while highlighting the aspects, which are no longer within the scope of this module. As introduced in the aforementioned deliverable, the discovery module allows an agent to become aware of nearby resources capable of contributing resources to the mF2C system. To do so, we proposed leveraging the ubiquity of the WiFi interfaces in mF2C-capable devices, such that a leader would broadcast custom WiFi beacons allowing devices in the vicinity to become aware of its presence. Such a solution is edge-centric thus alleviating core network traffic and it does not require relying on intermediate brokers to facilitate the discovery. We also note that, using this approach, the leader is also able to detect when an agent has left the area, therefore triggering a topology information update. As for the aspects which are no longer addressed by the discovery module, it is worth noting that the core part of discovery lies in "detection". Therefore, and as opposed to what has been presented in D2.6, "registration" is a different process whereby a user registers to mF2C and downloads the agent, as shown in workflow (figure 38) in D3.5 [7]. Additionally, trust mechanisms are now handled by the CAU entity.

Finally, it is worth mentioning that the consideration of having more than three layers in IT-2, has different practical implications in the discovery module. Until now, for IT-1 and with three layers, we had cloud agents (layer 0) in cloud, leaders (layer 1) and normal agents (layer 2). In this case, we only considered discovery between leaders and normal agents at the edge. Then, if more layers are considered, we will investigate other discovery approaches for the added intermediate layers. The chosen approach will strongly depend on the mechanism that has been used for building the topology.

Policies

This module was already described in D2.6 and the functionalities described there are still valid. However, new considerations are necessary to meet some of the changes in other modules and allow greater dynamicity in the agent. The policies module defines a set of rules for the management of the resources, describing for example, the way agents are grouped and organized in the mF2C system. An agent can have a different role according to its capacity and the responsibility it has. The already proposed roles in IT-1, are the normal agent, the leader, the backup and the cloud agent. These roles are acquired depending on the deployment and the characteristics of the policies. For example, we can create a set of policies to enforce that leader agents should have high availability and reduce mobility, prioritizing one or more of these characteristics in the agent resource specification and profile.

An important aspect of the policies block is to ensure the resilience of the area, defining a mechanism based on a backup agent that oversees the leader and in case of failure, changes the role to be the new leader of the area and regains control. In addition, some rules are required to correctly rank the devices according to the mobility profile (e.g. times the agent has left the area) and to avoid failures due to an agent moving outside the area. Another feature is to be able to establish some parameters, such as the scanning period to find the leader and when it is necessary to send new dynamic information of the device, according to the policies applied for a specific agent.

Identification

In the case of the identification module, the functionalities and strategy originally proposed in D2.6 remain the same. The general idea is to provide a method that allows us to assign unique identifiers to the devices in the mF2C network. In this sense, we have found that using hashes for naming allows us to meet the uniqueness requirement and not only that, but also to compute the identifier using a minimum of computer resources and still have a name with a negligible collision risk.

In mF2C, devices get the identifier (ID) during the agent installation. Such ID is calculated and assigned to the devices by the cloud agent using the IDKey (user identifier) as a part of the hash function input. Once the device is provided with the ID, it stores it locally and uses it whenever another module or device in the environment requests it.

Categorization-Monitoring

In D2.6 we have already described the meaning of Resource Categorization and in D3.5 [7] we have briefly described the functionalities of the Resource Categorization module. Based on the definition and the identified functionalities, we have already implemented the Resource Categorization module for IT-1. In advancing the procedure for IT-2, we identified some parts that require additional work, particularly on the IoTs (i.e., Sensor, actuators etc., and more precisely all sensor devices that are participating in the mF2C system). Considering the Semantic Sensor Network Ontology (SSN) and also by thoroughly reviewing their metadata, all sensor devices within the mF2C will be classified in IT-2.

Unfortunately, collecting the sensor information is not a simple task. Mainly because different vendors (i.e., manufacturer of the sensor) use their own drivers to connect the sensor to the system. So, following this issue, we found the necessity to design a unified and globalized plug-in connector software or external driver for attaching the sensor to the mF2C system. One of the main tasks of this kind of external driver is to properly identify the sensor type information and characteristics. Also, this kind of driver is responsible for properly attaching the sensor to the mF2C system and efficiently sending the sensor data to the mF2C system for further processing. So, in IT-2, our main goal is to design the prototype of this kind of external driver for mF2C. Similar to IT-1, in IT-2 the Resource Categorization module is responsible for continuously collecting information about the current resource availability (i.e., RAM availability, storage availability, processor availability etc.). That means

that for a particular agent the Resource Categorization module will keep continuously monitoring its resources availability.

4.1.2.User management

The second Iteration version of the User Management component presents some changes with respect to the architecture presented in deliverable D2.6. This section describes these changes.

The User Management component is part of the Agent Controller, and it is responsible for defining and executing the assessment of the user's device profile and its sharing model. This component is composed of three modules: Profiling module, Sharing model, and finally, user management Assessment module. This last module was renamed to better reflect its functionality and to avoid any misunderstanding with other components that are related to the Quality of Service. The first two modules were already present in the first iteration, and they are responsible for the definition of the rules and properties related to the resources shared by the user's device and the way they are shared. On the other hand, the assessment module was postponed for the second Iteration. This module will be responsible for checking that the mF2C applications running on the device behave according to the user's profile and that they do not use more resources than those defined by the user.

Profiling

This module presents some changes with respect to the one defined and implemented during IT-1. The most important change is that this module will include the implementation of the relationship between the user and the device. This way a user can define a different profile for each of his/her devices. Then, in IT-1 we only implemented two of the properties described in the previous architecture document: service_consumer and resource_contributor. These two properties refer to the way the user wants to participate in mF2C. Properties like the ones related to the device's battery will be moved to this module during IT-2, while others like the number of mF2C applications running in the device will be moved to the sharing model component.

Sharing model

This module was already described in D2.6 and its responsibilities and functionalities described there, are still valid. This component is called by the Assessment module to get the information about the device's resources that the user wants to share in the mF2C context. These resources include the number of mF2C applications running, as well as the maximum CPU, RAM, etc.

Assessment

The Assessment module was postponed for IT-2. It is responsible for checking that mF2C applications meet the profile and sharing model properties defined by the user. It will interact with other Agent Controller components to gather all the needed information, like the resources used by mF2C in the device. Then, if it detects any kind of violation, it will apply a set of policies. These policies define what to do when this module detects a violation or what to do when for example the device's battery is too low. These policies also define the mF2C components or agents that need to be notified, and the actions that need to be taken.

4.2. Platform manager

The Platform Manager is responsible for collecting application or service requests, translate them into internal calls, orchestrating the different tasks and performing resource optimal allocation. At runtime it controls service execution, according to the defined SLAs, and other supporting tasks like telemetry collection and management.

This high-level definition of the Platform Manager, already established in IT-1, remains valid for IT-2. However, after re-analysing each of the architectural blocks within the AC and the PM once IT-1 was

finished, the consortium agreed that some of the functionalities that had been initially assigned to the AC were, in fact, not local to an individual agent but required a more global perspective, which is the intended role of the PM. These functionalities are the ones related to Service Management, which in the revision of the architecture for IT-2 have been placed within the Service Orchestration block in the PM, as can be seen in Figure 8.

Also, the Data Management component has now been moved outside the PM, since it acts as a transversal functionality that is used both by the AC and the PM, the former mainly to store information that is consumed by the latter in order to take decisions. The Data Management component will be described in Section 4.3.

The PM is now divided into the same three main blocks defined in IT-1, namely Service Orchestration, Distributed Execution Runtime, and Telemetry monitoring, which now incorporate some additional functionalities in the case of the Service Orchestration. The components of each block and their specific functionalities are detailed in the following subsections.

4.2.1.Service orchestration

Lifecycle management

This module was described in D2.6 and the lifecycle deployment and management functionalities described there are still valid. One important improvement we want to add to this component has to do with the deployment of the services. During IT-1 we focused on the deployment of these services in Docker environments. And now, for IT-2, we aim to add Kubernetes and Docker Swarm to the mF2C catalogue to support more types of services and to take advantage of these containers environments (scalability, self-healing, load balancing, etc.).

SLA management

The user services running on the mF2C runtime may define an expected QoS to be satisfied during its execution. [For example, a service requires that calculating a path for the ambulance from the current position to the hospital is done in less than one second.] The QoS is formally specified in a document called SLA Agreement, and its fulfilment is observed by the SLA Management component.

The main functionalities offered by the SLA Management are the generation and storage of agreements and their assessment. For the assessment, the component relies on the performance metrics provided by the mF2C platform; for example, the execution time provided by the Distributed Execution Runtime (DER). The main difference between the design foreseen in D2.6 and the current state is the agreement specification. Instead of the WS-Agreement standard, we decided to define a new JSON schema, inspired by the WS-Agreement concepts. This way, not only the implementation is easier, but the new schema is compatible with CIMI [8], which only accepts JSON resources.

During IT-1, basic functionalities were implemented:

- Storage of agreement and violation resources in CIMI;
- Agreement assessment, which checks the agreement is satisfied, relying on execution time provided by COMPSs (thus, only COMPSs services could be managed by SLA). The assessment process generates an SLA violation each time the agreement is not satisfied;
- REST interface to manage agreements: manage the agreement's lifecycle, retrieve agreements, check their state.

For IT-2, the following functionalities are expected:

- Generation of agreements based on templates. While in IT-1 for simplicity reasons, all the service instances had the same agreement and the same QoS, for IT-2 the service definition will contain the description of the expected QoS of a service instance when being executed.

Then, an agreement for that service instance will be automatically generated from the template.

- Agreement assessment executed on leader. The assessment, being a lightweight process, was designed to be executed on the leader. This facilitates the load distribution (because it is not distributed). This functionality relies on the data distribution provided by the Data Management component.
- Assessment of other types of services (i.e., non COMPSs) and the usage of other metrics besides the execution time.

4.2.2.Service Management

Categorization

When a new service is registered into the mF2C system, the service manager will forward it to the categorization block. This block will classify the service according to the specified parameters. For IT-1 this block was just used for storing the services, providing interfaces to access or modify them. For IT-2, this module will, first, be responsible of creating categories for all the services stored in the system, and, second, for classifying new services according to these categories. While these categories can also be specified manually by the developer, the aim of this block is to do it automatically.

For creating categories, the categorization block will use a classification algorithm taking as input the whole set of pre-registered services and will provide N categories, where the number of categories has to be previously specified. These categories are created in order to give to the Recommender block a starting point for recommending resources for that service. Then, the Recommender will be able to update the categories when refining the recipe for each service. When new services are registered into the system, the categorization block will try to classify them into one of the available set of categories.

QoS Providing

When a new set of agents are chosen for executing a service, the QoS providing module tracks if the service execution produces SLA violations. If so, for the future executions of that service, the QoS provider will try to avoid agents that are not suitable. In order to know which agents are causing the SLA violations, the QoS provider will maintain a mapping between every specific service and all the agents that were used in every execution of that service. For example, if two independent service executions failed and there was only one agent in common, the QoS provider will determine that this agent is causing problems to the service execution. In this way, the QoS providing module will try to improve future service executions, just avoiding using agents that were failing during past executions. This block will work independently of the type of service is registered (i.e. Docker, COMPSs, Kubernetes, etc.).

QoS Enforcement

The QoS Enforcement module in IT-1 was a combined module with the SLA management in the user management. For IT-2 QoS enforcement will be an independent module running on the service management block. This block is in charge of guaranteeing the expected QoS of a service, which can be of two types: non-distributed or distributed. Non-distributed are executed in one agent using a single container or docker-compose applications. Distributed are executed in one or more agents using COMPSs, Docker Swarm or Kubernetes clusters. To be noted that differently from the QoS providing, the QoS enforcement works during the service execution, trying to fulfill the SLA agreement before a SLA violation occurs. To achieve that, this module will notify the Lifecycle to add or change agents during the execution.

Because non-distributed applications have limited means of enforcement, they lack of scaling out/in feature, instead the expected QoS to guarantee is the availability. If the service is down, because it

has crashed, or the agent is not available, the service can be migrated to another node. Using this sort of repairing mechanism policy the SLA can be enforced by guaranteeing an average availability.

Distributed applications allow richer SLAs to be fulfilled, based on scale out/in reactions. For COMPSs services, the execution time of an operation is expected to be enforced by using the following mechanism: (i) during execution COMPSs estimates the expected total execution time (ii) COMPSs publishes this information (iii) QoS Enforcement compares the expected value with the value in the SLA agreement; if this value does not satisfy the constraint in the SLA, the QoS Enforcement notifies the Lifecycle Manager to add more nodes to the execution. This mechanism makes sense for heavyweight operations, so it makes sense to add a new node to be able to enforce the execution time of a single execution of an operation. Another metric to consider is the average execution time during a minute (so a single execution does not violate the agreement but allows to perform the adaptation). This kind of metric is suitable for services to be executed on Kubernetes or Docker Swarm.

4.2.3. Distributed execution runtime

Task management

This functionality is responsible for detecting the applications' tasks and find data dependences between them. This module in IT-2 is enhanced in order to support the detection and instrumentation of tasks of Python applications. No other changes are foreseen or required from the use cases.

Policies

In IT-1 the DER only uses resources that are provided statically by the Lifecycle Manager at execution time. In IT-2 the DER includes the dynamicity of resources, allowing to use resources that are provided through the Policies interface. This interface is used to change the initial pool of agents available to the runtime, that, in its turn decides the optimal allocation of tasks on the resources. In the case of agents disappearing or failing in the execution of the tasks, the Runtime drops these agents from the list of available resources.

Task scheduling

The scheduling component in IT-2 does not include any major change in its behaviour. The relevant improvements are related to the scheduling of tasks on new resources as described in the previous section and an optimization of the usage of the agent's local resources. Another improvement is the provision of a forecast of the execution time of the application; this information is then used in the QoS component to evaluate the fulfilment of the SLA.

4.2.4. Telemetry monitoring

Landscaper

The key features of the landscaper remain unchanged since D2.6. The entry point for data is the suite of *"Collectors"*. These plugins capture properties of each physical host, docker container, etc. In the mF2C system, the physical device properties are captured by the *Resource Manager* of the Agent Controller and stored in CIMI. A *"CIMI collector"* has been implemented that queries the CIMI API for all devices and their properties to generate the relevant entries in the Landscaper. To support dynamicity of devices, an *Event Management System* is planned for IT-2 which will notify clients when updates to devices occur (new devices appearing, devices going offline, etc), this is documented in detail in section 4.4 (see the Event Manager block in the mF2C architecture in Figure 8). The landscaper will subscribe to these events allowing it to update its infrastructure model accordingly. The CIMI Collector will also be expanded in IT-2 to query service deployments configurations (based on the "Service Instance" object of CIMI). This includes Docker container metadata, the device the service is deployed on, etc.

Analytics Engine

The key function of the Analytics Engine remains the analysis of service performance. The current functionality in place at the end of IT-1 was focused on correlating compute (CPU) requirements of the service - as documented in the deployment config (service recipe) - with the actual compute utilization that was recorded during the execution of the service. The service recipe will be updated to reflect this utilization. This feature (modifying the service config) will be expanded in IT-2 to include identifying memory optimizations (RAM), disk I/O and possibly network requirements.

Recommender

The results of each of the service performance analysis completed by the Analytics Engine is stored in the Recommender. During IT-2, the recommender will perform updates to the deployment configuration (service recipe) based on these past executions. It is envisioned that with multiple deployments, this will increase the accuracy of the optimizations to the deployment recipe.

As the Recommender and the Analytics Engine are integrally linked, these two modules have been merged in the architecture.

4.3. Data management

As in IT-1, the Data Management component is the one used by other components to manage (i.e. store, update, and retrieve) the information needed for the operation of the mF2C platform.

In the previous iteration, some of the Data Management functionalities were assigned to the AC, namely those related to the direct manipulation of the data, while others, those more related to metadata, where part of the PM.

At the end of IT-2, given that the rest of components must access the data in the same way, regardless of whether they belong to the PM or the AC, the Data Management component naturally evolved to a transversal atomic component outside of these two blocks. Importantly, this is just a matter of software architecture, since the set of responsibilities assigned to the Data Management remain unchanged. In particular, these functionalities can be summarized as:

- Enabling other components to store, update, and retrieve the data they need to perform their functions locally in the agent
- Managing the appropriate replicas of each piece of information, in such a way that data is accessible from the agent that will need it, which is not necessarily the one that created or updated it.
- Supporting the Task Scheduling component in the Distributed Execution Runtime, by providing information about the location of the data, and/or creating the replicas or versions requested, so that the DER can optimise the distributed execution of tasks.
- Dynamically incorporating pre-existing data from new agents that join the infrastructure, so that the leader of the joined cluster is aware of all the required information.

As explained in previous deliverables, the Data Management component relies on dataClay [9] in order to perform its functions, so that replication and synchronization of data between a device and its leader behave according to the policies defined in mF2C. However, as in IT-1, dataClay is not directly exposed to the rest of components but is accessed through the REST-based interface provided by CIMI.

4.4. Event Manager

Some mF2C components rely on the system events which might happen at any time during the lifetime of an agent. Be it the addition/deletion of a device in the cluster, or even the election of a new leader, all system events will have an associated impact on the respective management resources which describe the state of the infrastructure. Therefore, the goal with this Event Manager is to add a

functionality to the mF2C agents whereby the remaining components can be notified whenever a certain event occurs. This functionality may be provided by an existing component, or even be a microservice by itself. Nevertheless, the desired workflow will consist of:

- components shall subscribe to the Event Manager with an explicit declaration of the events they are interested on;
- when and event of that type occurs, the Event Manager will generate asynchronous jobs;
- these jobs, when completed, will send the desired notification to the subscribed components, together with the affected payload from the event.

The Event Manager, if possible, should at least support notifications triggering both in the form of a network socket message, or via email.

4.5. GUI

Once the agent is installed and launched, the user can access to the GUI in order to register and launch services. This GUI is accessible locally through the web browser and allows to visualize all the services that are reachable by that machine. As shown in Figure 10, the user is able to register the new services by specifying, at least, the mandatory parameters.

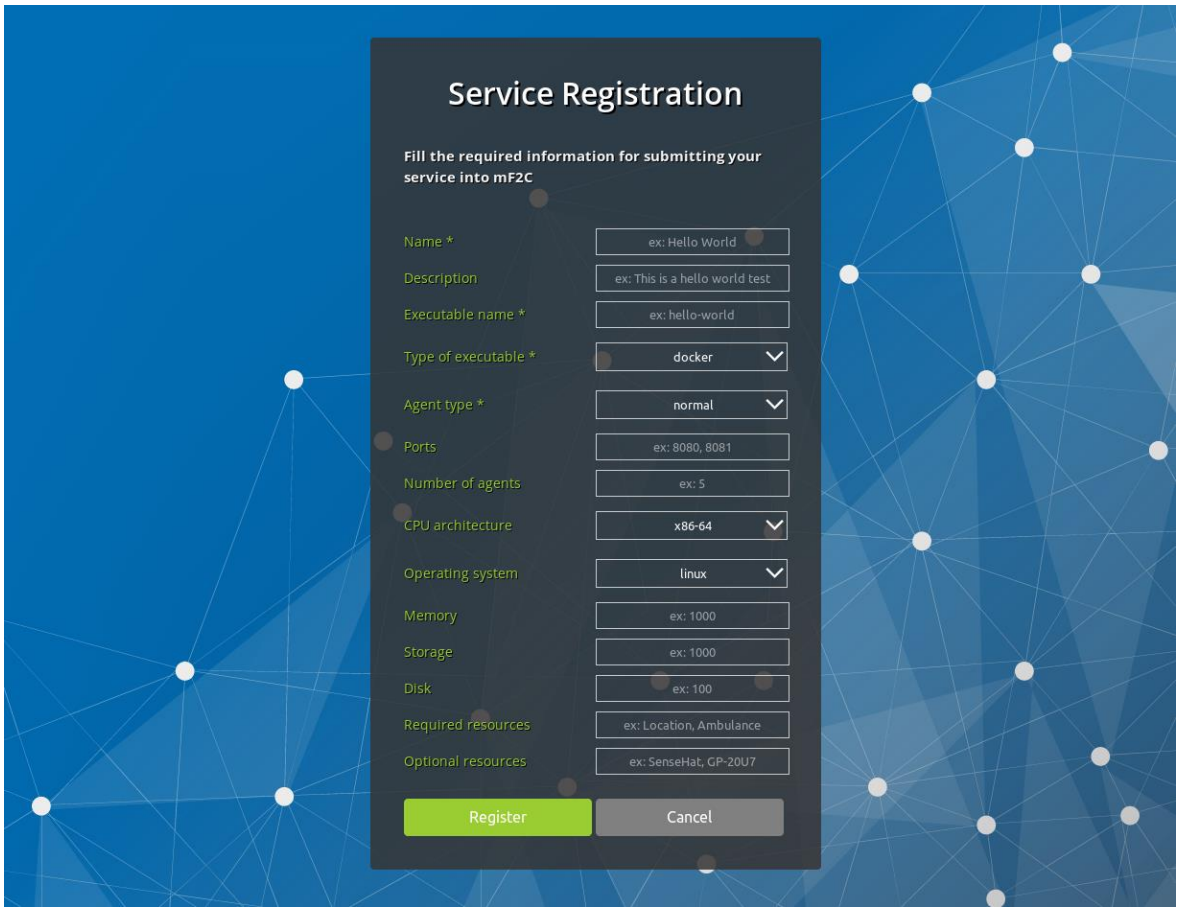

Figure 10: Service Registration

The required parameters are:

- Name: the name to identify the service.
- Executable name: the name of the executable that the user will run when executing the service, for instance, hello-world.

- Type of executable: Docker, docker-compose, COMPSs, Kubernetes or Docker Swarm.
- Agent type: the type of agent where the service has to be executed (cloud, microagent or normal agent by default).

Note that, for instance, when the type of executable is specified as Docker, then the executable name is the name of the public docker image available in Docker Hub. As optional parameters, the user can specify:

- Ports: required ports for the application (useful for docker-compose applications)
- Number of agents: number of agents where the service should be executed. If empty, then the service is executed in maximum number of available agents.
- CPU architecture: architecture required by the application to be executed (x86-64 or arm).
- Operating system: required OS required by application (linux, macos, windows, android).
- Memory: minimum amount of memory for the app to run.
- Storage: minimum required storage for the proper execution of the app.
- Disk: amount of disk I/O utilization necessary for the app.
- Required resources: required resources that the app needs to be properly executed. Those resources are identified by a tag name, that matches either by a driver name, agent name or any other resource specific name given by the developer.
- Optional resources: the same as the required resources but specifying the optional ones that the app will use only if they are available.

Once the service is registered, the user is able to see all registered services in the catalogue, as shown in Figure 11, from where a service can be launched.

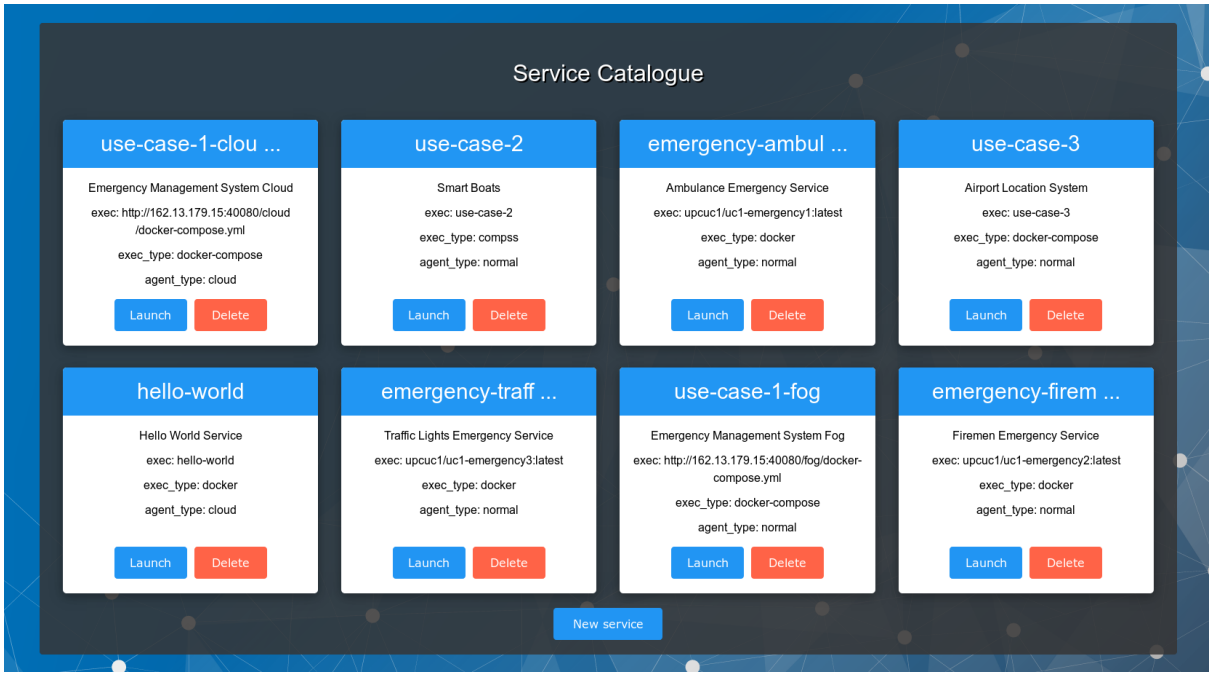

Figure 11: Service catalogue

4.6. Security

In IT-1, security revolves around protecting the integrity of an agent, while the focus in IT-2 is to move beyond an agent to the mF2C platform itself. In IT-1, we demonstrated how trust is bootstrapped during the Agent Discovery and Authentication process. The flow involved authenticating the new Agent and obtaining via the CAU a unique identity in the form of an X.509 certificate issued by the cloud CA (Section 3.5). In IT-2, we extend the scenario and cover how trust is negotiated between

agents and how we leverage the PKI to enforce access control and data security policy (see D3.1 [10]) within mF2C. In IT-1, we have prototyped a security library which facilitates the implementation of data security using the cryptographic keys bond to an agent's X.509 certificate (*viz.* identity) to sign and encrypt message tokens.

In IT-2, we consolidate the different security functionalities which cross-cut both the Agent Controller and Platform Manager into a logical Security block (see Figure 8). The main functional components included covers: trust (CAU client), web application endpoint security (Reverse Proxy) and data protection (AC Library).

4.6.1.CAU Client

In the IT1 Discovery and Authentication process, the fog CAU acts as the gateway to the cloud CA and is responsible for validating and handling a new agent's request for an X.509 certificate to identify itself to the leader of the fog cluster that it wishes to join (see section 5.2 for a description of the CAU within the architecture). In this role, the CAU knows about the identity (X.509 certificate) of all local agents and the intelligence may be further exploited to provide security functionalities in two key areas:

- access management, e.g. authentication and authorisation, provisioning access tokens, and
- data protection, e.g. signing and encrypting data with an agent's key to ensure integrity and confidentiality.

In D4.2 we will describe a scenario for access management with CIMI, the mF2C API, using the Leader Agent CAU as an identity provider to support the authentication and authorisation of inter-agent communication in a fog cluster. To ensure availability of agent credentials, the CAU needs to build in failsafe mechanisms like replication and redundancy. In addition, access to the stored credentials need to be controlled and communicated securely via point-to-point encrypted channels.

In D2.5 [4], we have already highlighted that blockchain (BC) or distributed ledger technology (DLT), with its inherent security features, may be exploited in IT-2 to deliver most of the security features required of the CAU:

- Peer network supporting redundancy
- Consensus mechanism providing near-real time replication of chain data
- Access control and cryptography safeguarding confidentiality of chain data
- Tamper-proof fabric protecting the integrity of chain data
- Immutable transaction chain ensuring accountability
- Automated smart contract enforcing user-defined policies at transaction time, e.g. controlling access, triggering notifications, revoke credentials, etc.
- Secure discovery – it must be possible for an agent to discover a CAU and ensure it is talking to a legitimate CAU (described in more details in deliverable D3.2, section 4.4.1)

One remaining question is whether a fog with hierarchical communication and several leaders needs more than one CAU, and if so, how this changes the requirements on the discovery mechanism. The options are to have a CAU per layer, a CAU per leader, or a CAU associated with the leader at the topmost fog level only or choose to have a single CAU per fog cluster. From the point of view of the functionality, the choice does not matter as long as the CAU discovery mechanism returns the endpoint of a CAU to the CAU client. Further details of the CAU's place in the architecture are described in Section 5.2.

4.6.2.AC Security Library

In mF2C, an agent is assigned a X.509 certificate that provides a unique and time-limited identity. In a PKI, the certificate is bond to a cryptographic key-pair that an agent uses to sign and encrypt messages to facilitate the Data Security Policy described in D3.1 [10]. The AC library provides utility methods for creating message token based on the security level required. Metadata and payload are captured as simple key-value pairs structured in JSON. Here is an example of the data structure used in the prototype:

- timestamp: channel publication timestamp
- qos: delivery quality of service enum
- sec: security level (public/protected/private) enum
- pro: transport protocol used, also an enum
- source: friendly name of sending channel
- signature: the signature data if message is protected
- publicKey: the sender's public key
- payload: the message content

The token is then secured in line with the required protection level:

- Public: no security enforced
- Protected: the payload is signed with the sender's private key to assert integrity
- Private: the payload is encrypted with the recipient's public key to enforce confidentiality.

The AC Library obtains the required key information from the CAU middleware via the CAU Client.

4.6.3.Reverse Proxy

An mF2C agent is deployed as a collection of Docker containers, with each block, except Data Management, exposed via a single REST interface published by CIMI. We need to ensure that the exposed endpoints do not become an entry point for attackers. CIMI implements standard security mechanisms such as point-to-point security with HTTPs protocol using X.509 certificate-based authentication (D3.2, section 3.2) and an internal access control list to filter access to its managed resources. We enhance this by incorporating a reverse proxy in front of CIMI.

In IT-1, the Traefik [11] reverse proxy was used as a single front-end to agent blocks that were not integrated with CIMI, such as the Lifecycle Manager and dataClay (described in deliverable D5.1 [2] Section 2.1). For IT-2, we propose to continue to use Traefik (D3.2, section 4.7.1) to protect the CIMI endpoints. For further discussion of the proposed CIMI security for IT-2, please refer to D4.2, section 4.1.

It remains to discuss the Data Management block. As mentioned above, it communicates outside of CIMI, so it needs to implement its own authentication and authorisation mechanism. This security implementation will be described in D4.2, section 4.2.

4.7. Application programming interface

Throughout the development of the mF2C architecture components in IT-1, it was inevitable to decouple the mF2C interface from the remaining blocks, as a separate functional component which serves as a unique point of entry for the incoming requests, and also an internal gateway for the components to manage data within the agent. Hence, in terms of design, the mF2C interface no longer belongs to the Platform Manager block, but instead it constitutes a whole new functional block of its own, as shown in Figure 10.

As already described in D4.3 [12] and hereby summarized, at the core of the interface is CIMI, which provides the mF2C management API, both for mF2C users and the internal mF2C components. CIMI provides a standard for managing resources within an infrastructure. For mF2C this means that every

possible resource that is to be managed (services, user profiles, credentials, devices, etc.), will be modelled and represented according to the CIMI specification. More details about the implementation of this CIMI-based REST interface can be found in D4.3.

Apart from the management of resources, another reason the mF2C interface has become a separate block in the architecture is because it is now also functioning as a language agnostic gateway to manage data within the mF2C agents. In order to decrease the communications complexity between all mF2C components and dataClay, a decision was taken in order to eliminate the technical burden of supporting the dataClay client individually. Instead, all components adhere to a more limited, but simpler, language agnostic REST API (CIMI) which allows CRUD (CREATE, READ, UPDATE, DELETE) operations and filtering capabilities in the database.

By having this interface as the main entry point for mF2C, the overall mF2C security will also be simplified as most network security concerns for dealing with external communications will be contained to this block. In Section 4.6.3, we have already outlined the security features provided by the reverse proxy which sits in front of CIMI. Communication between blocks within an agent is carried over secure private Docker network. External communication (i.e. requests for CIMI resources originating from outside an agent) over untrusted network, however, needs to be secured via standard point-to-point encryption (e.g. HTTPs, SSL, etc.) and each request authenticated and authorised.

5. mF2C cloud environment

For IT-2 we consider different requirements of the cloud environment in terms of necessary modifications in our architectural design. In the first place there is a need for a specific cloud agent, which differs from the standard agent described in the previous section. The other specific characteristics in our mF2C cloud environment include a central controller, the CA and the user registration process. Additionally, we discuss the option of having a multi-cloud support.

5.1. Cloud agent

As mentioned in Section 3.2 the cloud agent is a slightly modified version of the standard agent adapted for the cloud. The differences between the standard and cloud agents are the absence of CAU client (described in Section 4.6.1), the Identification and the resource Categorization modules (both described in Section 4.1.1).

Because of the centralization of the CA and the central controller in the cloud there is no need to have the CAU client, but the central controller is the one responsible for authorizing and authenticating the cloud agent. The Identification module responsible for assigning IDs during the mF2C agent installation process, is not required in the cloud either. The same thing is with the Categorization module from the Resource Management whose purpose is categorizing sensors attached to the agent, which is not the case with the cloud agent since we assume there are no sensors in this layer.

5.2. Security and privacy

Because of the importance and complexity of security and privacy requirements, in this section we will only describe basic concepts used, while a separate deliverable D2.5 [4] addresses security and privacy requirements for the whole architecture in more details. Two additional deliverables, D3.2 and D4.2, provide discussions of the options for the implementations of security for the agent controller and platform manager, respectively.

The foundation of the mF2C security is a public key infrastructure (PKI). The central issue in a PKI is to establish and maintain trust in the public keys, and the two classical methods of doing so are peer-to-peer and with an authority. In the former, peers cross-validate key ownership with each other when they have the opportunity and need to maintain trust lists throughout. Architecturally, this model is flat, with all peers being, in principle, equal. In the authority model, there is an issuing authority whose signature on a public key - in an envelope containing also the name of the owner - is taken as sufficient proof that the public key is owned by the entity named in the envelope. The most common mechanism for implementing this envelope is the X.509 certificate which is an ITU-T standard [13]. This model has thus, in its simplest form, two architectural levels, with the Certification Authority (CA) on top and the rest of the PKI in the lower level.

D2.4 [3] did not make a choice as to the architecture of the PKI, leaving the choice open for an authority (D2.4, section 3.3.1) and a P2P network (D2.4 section 3.3.2), but for IT-1, the authority model was used, with two CAs, both hosted in the cloud, with one CA issuing certificates to agents in the fog and the other to the infrastructure, e.g. CAUs. The architecture of mF2C thus matched that of the PKI.

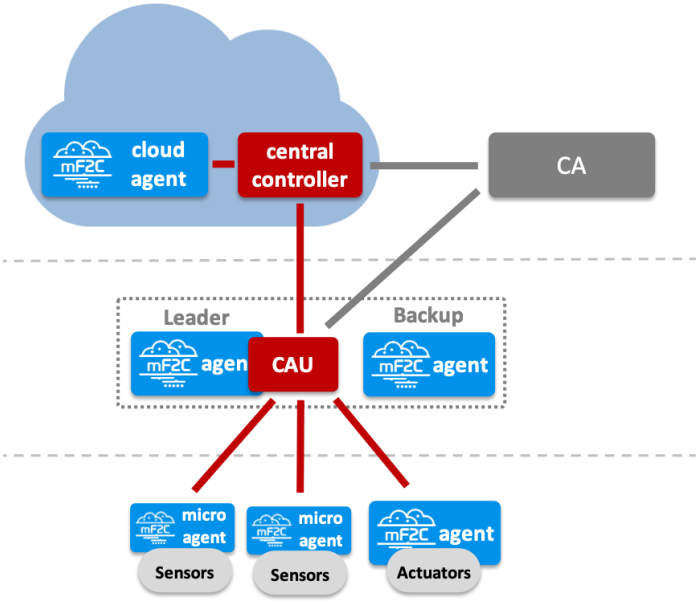

Figure 12: Security mechanisms in mF2C architecture

In IT-2, we would be moving beyond the security of an agent to securing interactions between agents. In a PKI, we need to ensure that an agent's identity (credential) can be verified by other agents in the Fog cluster to facilitate access control. This requires identity management, a functionality supported by the Control Area Unit (CAU) infrastructure in mF2C.

The CAU is a loosely-coupled security service that offers security functionality locally to nearby agents. The CAUs are organised hierarchically under a centralised controller located in the cloud with distributed controllers at the network edge acting as proxies for the cloud CAU. This design allows a global view of the constituent fog clusters protected by the CAU hierarchy to help coordinate security information while supporting low-latency authentication and authorisation of agents locally. The CAUs are part of the mF2C infrastructure and run alongside mF2C agents. To enable mutual trust between them both, CAUs are distributed with X.509 certificates issued by the infrastructure CA.

In IT-2, the CAU infrastructure provides identity and access management support to mF2C. An agent block uses its service to:

- manage its own identity, e.g. request/renew certificate
- request security tokens to access resources external to itself, e.g. another Agent, a third-party entity, etc.
- authenticate the identity of another agent requesting access to its resources
- retrieve the public key of another agent in the Fog for securing message exchanges, etc.

More detailed overview of the security framework requirements for proposed mF2C layered architecture from Figure 6 is given in sections 3.4 and 3.5 of the deliverable D2.5 [4], and the options for implementation in the forthcoming D3.2 and D4.2.

5.3. User Registration

The Graphical User Interface refers to the way the users interact with mF2C. In this sense the user can play three different roles:

- User registering him/herself
- Developer uploading services into the mF2C system
- User executing services

The first case, namely the user wants to register in the mF2C system, is done before the user downloads the agent into the devices, and thus before having the agent installed in the user's device. For this reason, the graphical interface for registering users is not part of the agent, and it is only located in cloud, as it can be seen in Figure 13. When the user successfully registers at cloud, all his/her data is stored at cloud, and a IDKey (user identifier) is generated, to be used as already described in the specific workflow to enable the registered user to download the agent.

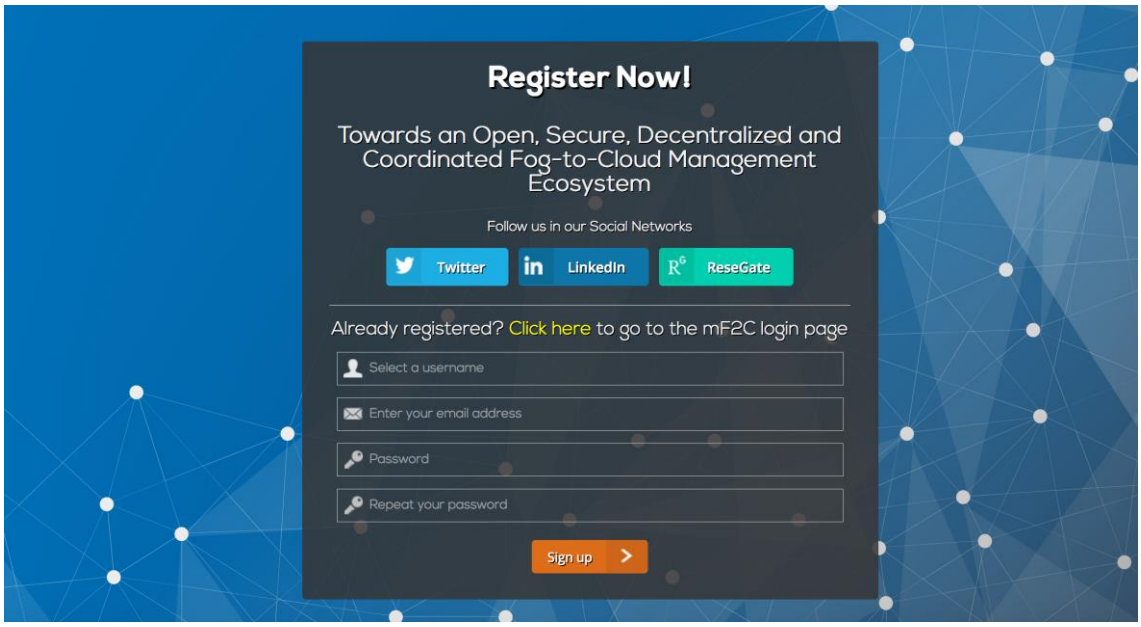

Figure 13: User Registration

5.4. Multi-cloud support discussion

There are two types of multi-cloud support we aim to address during IT-2. The first one is the capability of running mF2C services in different private or public clouds. In IT-1 we put the focus on the deployment and execution of services in clusters composed by devices like PCs and laptops, and we assumed that those devices had enough resources to run these services. But there could be situations where an agent (leader) is requested to execute a service, and this agent cannot find enough resources to run it. In such cases, mF2C should be able to forward the request to the upper layers of the mF2C hierarchy to find a suitable place to run this service. This "suitable" place could be another mF2C cluster or a private or public cloud with one or more mF2C agents.

One of the main objectives in IT-2, is to be able of forwarding the execution of "heavy" services to cloud agents in a transparent way for the final users. This way we can take advantage of mF2C agents that are running in public and private clouds, like Google cloud or AWS, where they have access to a high amount of resources. Thus, "heavy" services that cannot be allocated to other kind of "light" mF2c agents, could be deployed and executed in these other cloud agents. Figure 14 illustrates this idea: (1) a user launches a service from an agent; (2) mF2C does not find enough resources to run this service; (3) the request is forwarded to the upper layers, until mF2C finds an agent (in Google Cloud) with enough resources to deploy and run this service.

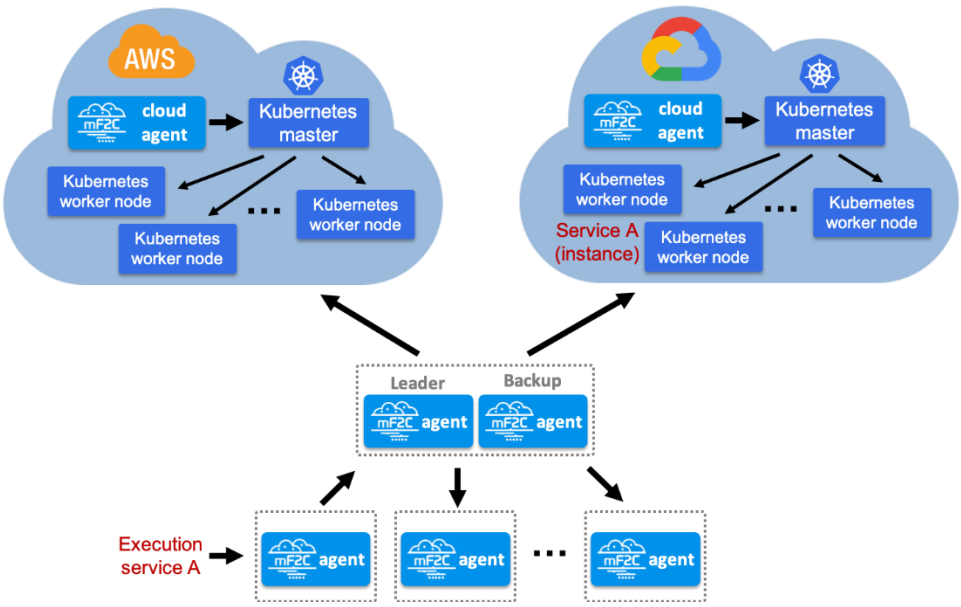

Figure 14: mF2C multi-cloud support

Other type of multi-cloud support that may be considered is the use of Nuvla [14] to deploy and install these mF2C agents in different public or private clouds.

6. Microagent architecture

During the previous iteration a single kind of agent was considered for the first proof-of-concept of the mF2C platform. In IT-2, we re-incorporate the concept of microagent that was initially envisioned in the proposal, since some devices do not have enough capacity to host a full agent. In fact, it does not make sense for those devices to host those functionalities that they will never execute.

We have conceived the microagent as the smallest possible agent that can be useful in our architecture. We have considered that a microagent only makes sense if it can be used as spare and/or additional capacity to offload computational workloads. Thus, the role of the microagent is to act as a "worker" to which the leader can delegate parts of the execution. Its role would be similar to that of a non-leader agent, but for the microagent we know it will never have other mF2C agents below. Note that this definition of microagent implies that devices that are not capable of executing parts of a service (such as sensors or actuators) are not considered as part of the mF2C platform.

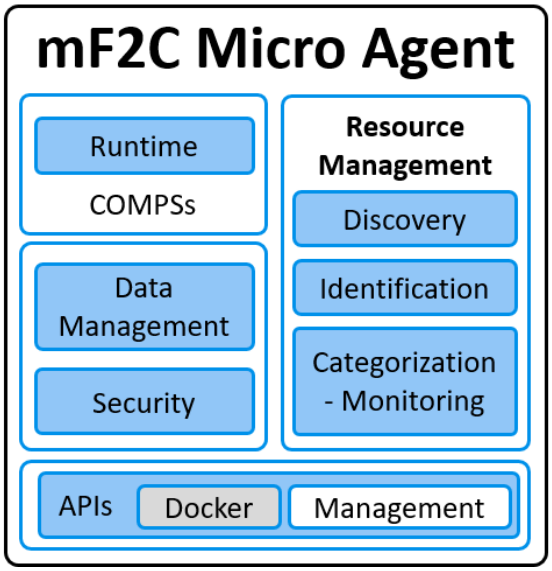

Figure 15: Micro agent architecture

Taking this into account, we have identified the minimal set of components that are necessary for this purpose, shown in Figure 15. Importantly, the distinction between PM and AC does not exist in the microagent, since it does not have any management functionalities.

As can be seen, all the components' names have an equivalent in the full agent architecture presented in Figure 10, which gives an idea of their intended functionality. This does not necessarily mean that their implementation and/or features are exactly the same, but rather provides an intuition of their role, which in general will be a simplification of the equivalent component in the original agent. The following subsections explain each of the microagent components.

6.1. Distributed Execution Runtime

There are no major changes in the usage of the Runtime in a microagent. The only limitation could be in the way the DER is deployed, possibly due to the lack of availability of container technology in the target platform, but it does not affect the functionality of the runtime itself. This anyway does not affect the behaviour of the runtime, which is independent of the way and on the platform where it is started.

6.2. Resource Management

The resource management block of the microagent will still need to comprise mechanisms for discovery, identification and categorization/monitoring. In summary:

- **discovery:** it will give the microagent the capability to find and/or be found by other mF2C agents;
- **identification:** it will ensure that the microagent can be recognised and trusted by mF2C;
- **categorization/monitoring:** it will be simplified, with respect to the original agent architecture, so that it can produce and report basic telemetry about the microagent, which should help the mF2C leaders decide on the availability of resources and allocation of services.

6.3. Data Management

Similar as is the standard agent architecture, the Data Management in the microagent is used by other components to manage the data they need to perform their functions. In this case, however, it will only take care of data regarding the resource (device) where the microagent sits, as well as providing the required support to the Distributed Execution Runtime.

The implementation may, however, be different than that of the standard agent, since it is not clear that the microagent needs to store any data itself. Instead, the Data Management can provide an interface to the client components that transparently transfers the data to the leader so that it is stored there. However, from the point of view of the components, the data will be accessed as if it was local in any case.

6.4. Security

Going back to IT-1, the intention for the “less capable” devices was that they could communicate “upward” in the architecture to a more capable device running an agent, which would relay information on behalf of the original sender. An example could be a sensor device communicating data over a serial channel to a more capable device running an agent.

For IT-2, it makes sense to see how we can extend the security capabilities of the agent to the microagent. There are four general approaches, going from more capable to less capable devices:

1. A pared-down docker environment is deployed on the (less capable) device, with containers only for the necessary security modules (plus of course any other containers it needs as a microagent.)
2. The security software is rebuilt for the restricted environment of the device running the microagent (e.g. a relevant jar (“java archive” containing code) is extracted and redeployed on the restricted device.
3. The security code is rewritten/re-engineered for the natural programming environment of the device, e.g. the relevant features are ported to C, porting only the specific code that the device actually needs.
4. Security soft/hardware is used to assist the device. For device capable of there, it could be used to assert device identity and sign communications.

It would make sense to explore all of these options during the development of IT-2, albeit of course prioritised, giving priority to specific devices such as (say) NuvlaBox Nano and Raspberry Pi.

One reason for exploring the full set of options above is that sensors are important to the use cases, so we want to explore the question of what is the smallest device that can securely send sensor data into a mF2C fog, in compliance with the security requirements. Another reason is that the less capable devices are generally cheaper than the more capable ones. If a less capable device could be made to work in place of a more capable one, the deployer would save money and would be able to deploy more devices.

We assume the following security functionality for the microagent as a minimum:

1. Security bootstrap: as the agent may connect to more than one fog, it could be pre-configured with a certificate rather than bootstrapped through the CAU but must have the ability to connect and discover the leader.
2. Agent-specific certificate/private key, used for authentication, and origin authentication (see also D2.5 [4] section 4.3.1)

6.5. Application programming interface

Contrary to the interfaces described in section 4.7, for the microagent the desired behaviour is that the device will simply be a worker node and accept any job which has been sent to it. This interface does not need to comprise the same management capabilities as for a standard agent, since system resources and local mF2C data management won't exist. Thus, this interface only needs to:

- be supported by the mF2C agent running as leader;
- be capable of holding temporary data (either in memory or in the local file system), and
- be capable of providing application management operations (i.e., create, list, delete, etc.).

7. Usability and Implementation of the architecture

This section defines additional aspects, beside the design itself, that are important to take into consideration due to the implications they can have on the architecture. These aspects include architecture scalability, analysis of different applications types that fit to the mF2C platform in terms of general use cases, software engineering implications and mF2C performance limitations.

7.1. Architecture scalability

In mF2C, scalability is a very important aspect to consider in terms of how the architecture is able to scale up the number of new agents joining the system. When the number of agents in a certain area increases, so does the probability of higher number of service requests. These aspects have to be handled within the confines of the mF2C infrastructure with the advantages of its decentralized, flexible and hierarchical topology. In order to achieve scalability of the mF2C architecture and to enable flexibility to work in different scenarios, we need the best possible combination of key enabling technologies that can be adopted throughout the system.

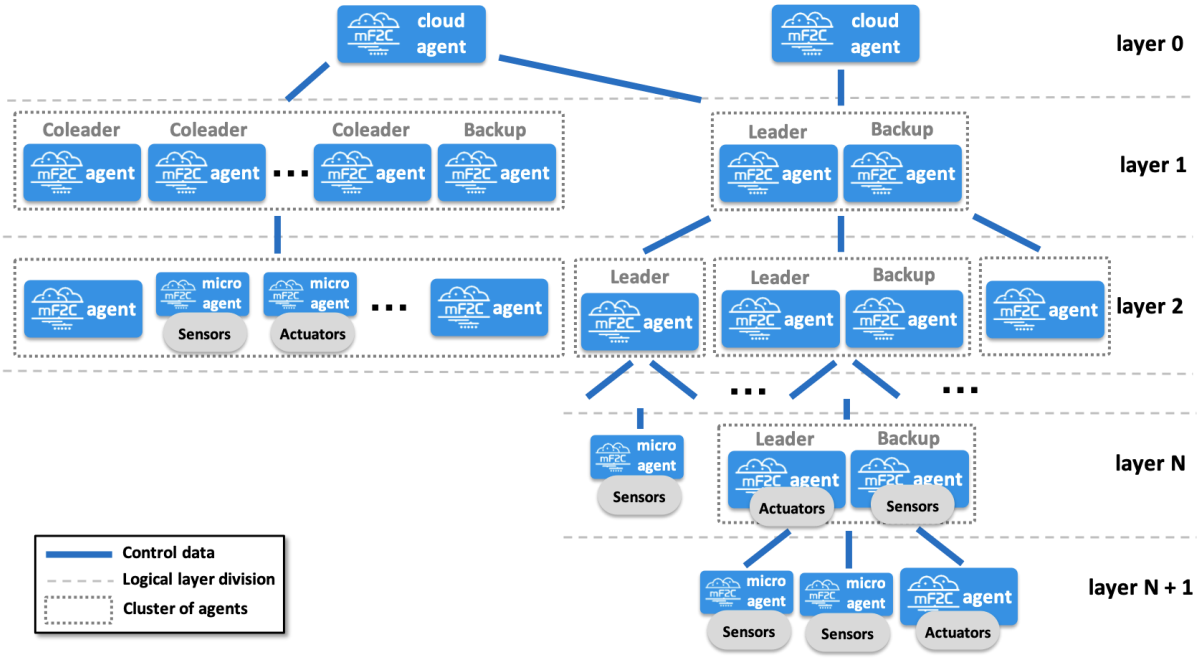

Figure 16: mF2C architecture scalability

The mF2C system topology, is organized into multiple hierarchical logical layers, where each layer contains one or more clusters of agents, and each cluster can be composed of one or more agents. When a cluster is managed by a cluster from an upper layer, the one in the upper layer contains at least one leader agent. This leader agent will be responsible for managing all service requests performed from the cluster below, having the possibility of using all these resources for service execution. A cluster can have more than one leader, in that case all of them will become coleaders. Having multiple coleaders can be used as a load balancing solution when a high number of agents from lower layers is being managed. All coleaders and backups synchronize their state data in order to replicate the same information among all. The only difference between coleader or backup is the possibility of asking for service execution or not. Therefore, all coleaders are able to execute service instances, while the backups cannot. Instead, the backups are used to improve reliability by becoming leaders in a case that some of the coleaders fail or leave the cluster.

How the number of coleaders is selected will depend on each situation. For instance, when the number of agents that need to be managed is too high, more coleaders are necessary in order to manage a higher number of service requests, and fewer otherwise. For determining the number of backups, other aspects like mobility or reliability can be taken into account. For instance, in a high mobility scenario, when agents leave and join the system all the time, having multiple coleaders which are not reliable could mean more efforts for successful synchronization. In this case, it would make more sense to select the most reliable device as the leader and place few additional devices as backups, minimizing the probability of data inconsistency due to synchronization issues. However, because there is no perfect solution that would fit all the cases, mF2C will study both horizontal and vertical scalability scenarios in order to determine which solution is better under different circumstances.

The decision whether an agent is a leader is a capability-based decision, so for instance, in one layer it may be stated that all devices with 2GB of RAM can assume leader role. So, if a device fulfils this condition it becomes a leader and if there are other leaders within the same cluster, the load can be balanced among them. This also means that there are multiple backups available. The precise policies for choosing a leadership role will be defined in later phases. Regardless of the number of leaders, they will be synchronised with each other through dataClay. When considering agent's components scalability, it is important to pay special attention to three components: dataClay as a data management related component, CIMI as mF2C management API (for mF2C users and the internal mF2C components) and COMPS as the distributed execution runtime.

CIMI's scalability is completely related to the backend's (database) scalability. Horizontally, CIMI can be distributed in a cluster and load balanced to add redundancy while efficiently distributing network traffic. This is useful in our mF2C system design with multiple leaders per fog area where with the assumption that all traffic should go through the mF2C leader first, we need load balancing. Vertically, a drastic limitation of the available resources (mostly memory) might cause the CIMI server to malfunction and even shutdown. The only available solutions for vertically scaling CIMI are: 1) re-sizing (adding more power) to the device; or 2) re-implement CIMI in a less memory-demanding web framework.

COMPSs is deployed as a set of multiple agents, each one independent of the others, thus enabling horizontal scalability and load balancing and allows to start a service or an application on the most suitable one at each moment. On the other hand, COMPSs scales the execution of a single application assigning the tasks it is composed of across the available resources chosen as possible workers and assigned by the Lifecycle Manager.

Regarding the scalability of the database, dataClay is designed in such a way that every backend can operate independently of any central authority, thus avoiding bottlenecks during operation and ensuring scalability. This behaviour is confirmed by our scalability tests, performed in an in-lab cluster that would be equivalent to 512 devices connected within a fog area. The results obtained up to this number of devices show a good scalability trend and do not provide any sign of possible scalability problems if more resources where added. In fact, since only part of the data needs to be replicated and synchronized in a limited number of agents (i.e. leader and backup), and the number of communications is linear to the number of replicas, the bandwidth requirements of dataClay are very small. Thus, the scalability of dataClay depends directly on the scalability of gRPC, on which dataClay relies for the communications.

7.2. Software engineering implications

In practice, the refurbished architecture will ease the implementation of the mF2C components as it has now better identified how to separate the functional blocks which have different functions inside

the mF2C agent. Nevertheless, the architecture is a conceptual guideline, hence any future software developments might still identify other areas for improvement (components which can be merged, blocks which are still missing, etc.). In summary, this current architecture is more in line with the microservices concept which has been aimed for during the development of all mF2C components, so the expectations are that there will be no disruptive implications on software engineering for IT-2.

7.3. Adaptability in general use cases

The scenarios applicable to the mF2C infrastructure are applications using a cloud-like approach on fog to cloud infrastructures. From a developer's perspective, the services should be containerized form to be compatible with the mF2C concept, while the application management, the orchestration and deployment, is done by the mF2C platform.

There are plenty of applications types that fit to the mF2C platform; the following lists a few major ones:

1. **Data gathering apps** - applications based on gathering data from multiple resources distributed on the fog. mF2C eases the concept of gathering as it allows deploying parts of the application on the edge near the sensors. This means that gathered data can be filtered and processed near the sensor. Benefits include lower consumption of bandwidth, storage and an increase in safety and privacy for the applications.
2. **Compute apps** - some applications on mobile devices, such as mobile phones, cars, and portable computers, require more computational capacity then they have on their own. mF2C allows for fast deployment of containers on edge devices that can assist as additional workers for the application running on the device. The compute intensive jobs can be offloaded to the fog and, if necessary, also to the cloud.
3. **Geo-notification applications** - are applications that need to notify other clients about events in the fog area in near-real time. For example, if there is a car accident detected on a highway, other road users in vicinity need to be notified as quickly as possible. Sending this message to the cloud, filtering it by the zone of affected users and sending it back could result in a high latency. The hierarchical architecture of mF2C can group devices by locality and, in this case, a device only needs to notify users that have the same "parent" or "grandparent" agent.
4. **Applications requiring connection resilience** - due to the distributed and fault-tolerant architecture of the mF2C platform, the system is resilient to the failure of the connection to the cloud. Functionality available in the fog area is still available and helps sustain the period until a connection to the cloud is re-established.

These concepts are only a subset of fundamentals that can be applied to a large set of applications, ranging from industrial grade, based on the concept of Industry 4.0, where the edge and fog zones have specialised and critical goals, to the everyday consumer applications on the people's mobile phones. However, the details between the cloud and fog to cloud are different, but the users interact only with the application on top of the platform and, for them, the mF2C architecture and the platform are transparent---a black-box. The only difference with the cloud that users can sense is an increase in performance.

7.4. Performance

In mF2C, limitations on the achievement of the optimal performance can be due to the nature of the platform, where dynamicity plays an important role that is in principle contradictory to high performance. In high performance computing, the lack of resources is not an issue, and the availability of a reliable infrastructure can be assumed to a certain extent. In contrast, in fog-to-cloud environments, these assumptions are no longer valid.

However, there are some commonalities between both areas. The main shared characteristic is the need for scalability, that is, being able to take advantage of a virtually unlimited set of computational resources. This has been achieved in mF2C by making DER to rely on the COMPSs programming framework, capable of distributing execution among resources to achieve parallelism. COMPSs has been adopted to implement applications for HPC platforms, demonstrating scalability and improvement in the overall performance of the original code. Real time behaviour is being evaluated also thanks to the participation of the COMPSs team in the CLASS project.

The second one is reducing the amount of communications as much as possible, since they are an important and well-known source of overhead. In the design of the mF2C architecture, the minimization of communications between agents, and especially to the cloud, has been considered as an essential requirement. Thanks to the hierarchical architecture, only some specific agents (the ones in Layer 1) communicate with the cloud, while the rest only communicate with those agents that are closer, thus avoiding expensive communication roundtrips.

Another relevant design choice in mF2C is that agents can operate as much as possible even when they lose connectivity. This implies being able to access the data they need without having to communicate with others, which, as well as supporting dynamicity, has a positive impact on performance. The Data Management functionality takes care of this by relying on dataClay, which makes the appropriate data available to the leader so that it has a global view of its cluster without requesting any data to other agents.

Additionally, in the design of the components, communications have been removed from the critical path of service execution. In particular, by appropriately coordinating the Lifecycle and the Landscaper and Recommender, a service execution request is assigned to the best set of available resources that are capable of performing the execution, thus minimizing the probabilities of having to look for additional resources at runtime and avoiding costly communications to coordinate agents during execution. However, failures occurring during service execution due to an agent leaving unexpectedly cannot be avoided and will cause additional communications, but this issue is inherent to the dynamic nature of the mF2C platform.

Finally, another common source of overhead is the access to disk. It has been tackled in the Data Management component by relying on dataClay, initially designed for HPC and being adapted to fog-to-cloud requirements in this project. DataClay minimizes the impact of disk latencies by operating in memory and accessing the storage device only when data does not fit in the cache. For the operation of the infrastructure, an agent only deals with a small amount of data and, thus, most data accesses will happen in real time since data is already in memory in the format in which it will be consumed. Regarding application data, the extent to which real-time data access can be achieved will depend on the amount of data and how the application manages it. But in any case, disk latencies are always avoided as long as the data used at a given time fits in the memory of the device.

8. Conclusions

This deliverable describes the second version of the mF2C architecture (aligned to iteration 2 of the mF2C project). It presents an updated and modified mF2C architectural design based on the challenges and experiences reported during the deployment of the IT-1 mF2C architecture and the development of its components.

The resulting deliverable offers a modified architectural solution, introducing two main mF2C architectural entities - mF2C agent and mF2C microagent along with different and modified mF2C components and interfaces. The set of requirements from IT-1 imposed by the applications and services running in the mF2C framework has been analysed and updated where necessary.

Same three use cases defined in the project proposals and tested during the first deployment phase, where the main challenges with the architecture have been reported, are being used to modify the requirements imposed by the applications and services for second iteration of the project. The idea of the global design of the system with a layered architecture remains the same during IT-2, as explained in Section 3, but with a more focus on the implication that dynamicity and mobility of the mF2C agents could have for the architecture. Section 4 of this document gives a detailed description of the agent entity, with the definition of each individual block of the system, emphasising the new components and the ones whose functionality have been changed. Sections 5 and 6 describe characteristics of scaled up and scaled down versions of the standard agent, that is cloud agent and microagent. The last section defines architecture scalability consideration, analysis of different applications types that fit to the mF2C platform in terms of general use cases, software engineering implications and mF2C performance limitations.

References

[1] mF2C Consortium, “D2.6 mF2C Architecture (IT-1),” June 2017. [Online]. Available: http://www.mf2c-project.eu/d2-6-m6/.

[2] mF2C Consortium , “D5.1 mF2C reference architecture (integration IT-1),” June 2018. [Online]. Available: http://mf2c-project.eu/d5-1-mf2c-reference-architecture-integration-it-1/.

[3] mF2C Consortium, “D2.4 mF2C Security/Privacy Requirements and Features (IT-1),” May 2017. [Online]. Available: http://www.mf2c-project.eu/d2-4-m4/.

[4] mF2C Consortium, “D2.5 mF2C Security/Privacy Requirements and Features (IT-2),” January 2019. [Online]. Available: http://www.mf2c-project.eu/d2-5-mf2c-security-privacy-requirements-and-features-it-2/.

[5] mF2C Consortium, “D6.2 mF2C annual report on dissemination and standardization (IT-1),” December 2017. [Online]. Available: http://www.mf2c-project.eu/d6-2-m12/.

[6] mF2C Consortium, “D5.3 mF2C solution demonstration and field trail results (IT-1),” June 2018. [Online]. Available: http://www.mf2c-project.eu/d5-3-mf2c-solution-demonstration-and-field-trial-results-validation-it-1/.

[7] mF2C Consortium, “D3.5 mF2C Agent Controller Block integration (IT-1),” November 2017. [Online]. Available: http://mf2c-project.eu/d3-5-m12/.

[8] Distributed Management Task Force, Inc. (DMTF), “Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol,” 2016. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0.pdf.

[9] A. Q. D. G. A. B. J. J. C. T. C.] J. Martí, “dataClay: A distributed data store for effective inter-player data sharing,” *Journal of Systems and Software,* vol. 131, pp. 129-145, 2017.

[10] mF2C Consortium, “D3.1 Security and privacy aspects for the mF2C Controller Block (IT-1),” July 2017. [Online]. Available: http;//mf2c-project.eu/d2-1-m6/.

[11] “Traefik proxy,” [Online]. Available: https://docs.traefik.io/.

[12] mF2C Consortium, “D4.3 Design of the mF2C Platform block components and microagents (IT-1),” September 2017. [Online]. Available: http://mf2c-project.eu/d4-3-m9/.

[13] I. T. U. S. s. (ITU-T), “Recommendation X.509,” [Online]. Available: https://www.itu.int/rec/T-REC-X.509.

[14] SixSq, “Multi-Cloud, Hybrid-Cloud and Edge Application Management Service,” [Online]. Available: https://sixsq.com/products-and-services/nuvla/overview.

[15] mF2C. [Online].

[16] “[RFC7515] JSON Web Signature (JWS), an IETF standard for the signing arbitrary data using JSON-based structure,” [Online]. Available: https://tools.ietf.org/html/rfc7515. [Accessed 15 November 2018].

[17] “[RFC7516] JSON Web Encryption (JWE), an IETF standard for the exchange of encrypted data based on JSON data structure and Base64 encoding,” [Online]. Available: https://tools.ietf.org/html/rfc7513. [Accessed 15 November 2018].

[18] mF2C Consortium, “D2.4 mF2C Security/Privacy Requirements and Features (IT-1),” April 2017. [Online]. Available: http://www.mf2c-project.eu/wp-content/uploads/2017/05/mF2C-D2.4-Security-Privacy-Requirements-and-Features-IT1.pdf.