mF2C

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

D2.6 mF2C Architecture (IT-1)

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 architectural definition**
Due Date:	*M6*
Submission Date:	*Date of actual submission –30/06/2017*
Version:	*0.9*
Status	*Final*

Author(s):	*Admela Jukan (TUBS)*
Reviewer(s)	*Eva Marin (UPC)* *Alexander Leckey (INTEL)*

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	25/05/2017	ToC and initial structure	Admela Jukan (TUBS)
0.2	09/06/2017	Merged contributions	All partners
0.3	10/06/2017	Integration	Admela Jukan, Jasenka Dizdarevic (TUBS)
0.4	15/06/2017	Edits	Admela Jukan, Jasenka Dizdarevic (TUBS)
0.5	20/06/2017	Harmonization	All partners
0.6	23/06/2017	First review, integrated comments	Eva Marin (UPC), Alexander Leckey (INTEL)
0.7	25/06/2017	Comments addressed	All partners
0.8	28/06/2017	Integrated version	Admela Jukan, Jasenka Dizdarevic (TUBS)
0.9	29/06/2017	Final version and quality check	Lara Lopez (ATOS)

Table of Contents

List of figures

List of tables

Executive Summary

This document, developed by the mF2C project, represents an initial version of the mF2C Architecture deliverable (iteration IT-1) that offers an overview of the mF2C architectural design.

The focus of this document is the identification of different components and interfaces within the mF2C architectural building blocks, as well as the different strategies necessary for a successful mF2C deployment. The role of each of the blocks is defined, as well as the required interfaces for the communication between blocks, comparing different solutions for the system control communication and data communication.

Furthermore, it defines different types of requirements imposed by the applications and services running in the mF2C framework analysis three use cases specified in the project proposal. The outcome of this document is a mF2C architectural design solution with a detailed functional definition and specification of the mF2C management systems and services.

1. Introduction

1.1 Introduction

The mF2C Architecture document identifies and gives a detailed description of the main architectural requirements and components of the mF2C system and services.

This document is structured in the following way:

- Section 2 describes requirements imposed by the applications and services running in the mF2C framework, as well as the technical requirements and expected challenges.
- Section 3 gives a detailed description of the architectural design and the two main building blocks, Agent Controller and Platform Manager, as well as the description of Baseline Subsystems and Communication.
- Section 4 presents the functions of the multiple modules and explores the highly innovative collaborative model envisioned for mF2C.
- Section 5 defines a development environment for mF2C user-applications.
- Section 6 concludes this document.

1.2 Purpose

The objective of this deliverable is to identify and detail the main architectural requirements and components of the mF2C platform.

1.3 Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
AWS	Amazon Web Services
BG	Business Goal
BLE	Bluetooth Low Energy
BMC	Baseboard Management Controller
CaaS	Container as a Service
CE	Core Elements
CIMI	Cloud Infrastructure Management Interface
CLI	Command Line Interface
CMDB	Configuration Management Database
D	Deliverable
DE	Decision Engine
DMTF	Distributed Management Task Force
DNS	Domain Name System
ESM	Emergency Situation Management
F2C	Fog-to-Cloud
FP7	7th Framework Programme
GDPR	General Data Protection Regulation

HDMI	High Definition Media Interface
IoT	Internet of Things
iPaaS	Integration Platform as a Service
IT-x	mF2C software release iteration
LoRa	Long Range
MDC	Mobile Device Cloud
MGC	Mobile Grid Computing
MTA	Mail Transfer Agent
NaaS	Networking as a Service
NFV	Network Functions Virtualization
OpEx	Operation Expenditure
QoB	Quality of Business
QoS	Quality of Service
REST	REpresentational State Transfer
SaaS	Software as a Service
SB	Smart Boat
SDN	Software Defined Networking
SDS	Software Defined Storage
SFHS	Smart Fog-Hub Service
SLA	Service Level Agreement
SLO	Service Level Objective
TCP	Transmission Control Protocol
VESA	Video Electronics Standards Association
VM	Virtual Machine
WP	Work Package

Table 1. Acronyms

2. Requirements and Business Goals

2.1 Context, including business and roles

The increased usage of IoT devices, grouped in fogs, has highlighted the existent breach between traditional cloud computing services offering and current IoT needs. mF2C aims to close this gap, providing a framework capable of managing resources and services in an optimal way. Technical requirements are deeply analysed to develop an extended distributed platform to integrate cloud services, IoT and fog. However, business needs and requirements must be also taken into account in order to develop a framework able to cover real stakeholders' expectations, aiming to increase industry adoption.

The process followed is structure in four main sections:

- Analyse the problem to be addressed by mF2C.
- Identify the main actor that will interact with the proposed management framework.
- Elicit potential business goals to cover expectations and needs.
- Define clearly messages for the benefits of adopting mF2C results.

2.1.1. Fog-to-Cloud problem scope

Cloud computing traditionally serves IoT customers supporting them on the storage and virtualization of their generated data, and generating value for their businesses. However, the growth of IoT is affecting the way traditional cloud architectures work. The increased amount of data to be transferred can create potential bottlenecks while increasing the latency. Furthermore, sending such a big amount of data to a cloud environment in very short periods of time can be inefficient and ineffective, apart from cumbersome and expensive. This implies that much of this data must be aggregated at the end points where data is collected. And here is where fog computing appears.

Both Fog and Cloud are concerned with compute, networking and storage; they share many of the mechanisms and attributes, such as virtualization and multi-tenancy. However, Fog computing aims to address the Latency issues detected in large IoT scenarios. This way, Fog Computing is not devised as a competitor to cloud; quite on the contrary, it is envisioned as the perfect ally for a broad spectrum of use cases and applications for which traditional Cloud Computing is not sufficient.

As in any emerging technology, multiple and compelling definitions have emerged aiming to capture all different flavours and nuances the term Fog Computing can have.

Bonomi [1] in his forefather paper proposes the following definition for Fog computing: "*Fog Computing is a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing Data Centers, typically, but not exclusively located at the edge of network*".

Definition provided by Vaquero [2] does not limit technology choices to virtualisation and adds a cooperation factor: "*Fog computing is a scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralised devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of third-parties. These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment. Users leasing part of their devices to host these services get incentives for doing so*."

Cisco [3], the pioneer on using the term defines it as: "*Fog Computing is a paradigm that extends Cloud computing and services to the edge of the network. Similar to Cloud, Fog provides data, compute, storage, and application services to end-users. The distinguishing Fog characteristics are its proximity to end-users, its dense geographical distribution, and its support for mobility*".

Recently, OpenFog Consortium[1] released a definition which describes Fog Computing as [4]: "*Fog computing is a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things. It is a:*

- *Horizontal architecture: Support multiple industry verticals and application domains, delivering intelligence and services to users and business*
- *Cloud-to-Thing continuum of services: Enable services and applications to be distributed closer to Things, and anywhere along the continuum between Cloud and Things*
- *System-level: Extend from the Things, over the network edges, through the Cloud, and across multiple protocol layers – not just radio systems, not just a specific protocol layer, not just at one part of an end-to-end system, but a system spanning between the Things and the Cloud*".

According to Cisco [3] the main characteristics of fog are as follows:

- Low latency and location awareness
- Widespread geographical distribution
- Mobility
- Very large number of nodes
- Predominant role of wireless access
- Strong presence of streaming and real-time applications
- Heterogeneity

The Fog vision was conceived to address applications and services that do not fit well with the paradigm of the centralised Cloud. They include:

- Applications that require very low and predictable latency—the Cloud frees the user from many implementation details, including the precise knowledge of where the computation or storage takes place. This freedom from choice, welcome in many circumstances becomes a liability when latency is at premium (gaming, video conferencing).
- Geo-distributed applications (pipeline monitoring, sensor networks to monitor the environment).
- Fast mobile applications (smart connected vehicle, connected rail).
- Large-scale distributed control systems (smart grid, connected rail, smart traffic light systems).

At the time of writing, controversy among differences, similarities, coincidences and divergences among use of term Edge Computing and Fog Computing continue [5], [6], [7]. For sake of simplicity, and in coherence with our initial work in the mF2C DoA, mF2C project materials refer to fog computing.

From a business perspective, fog computing complements traditional centralized (cloud) architectures in those cases where a quick decision must be taken or where there is a need that only relevant data is sent to the cloud. The combination of fog and cloud technologies leads to the development of new business models and cost-effective operations in different sectors.

Computing at the edge is expected to reduce costs and ease the interaction among different applications, paving the way for the integration of operational technologies and information technologies.

[1] OpenFog Consortium was founded by Cisco Systems, Intel, Microsoft, Princeton University, Dell and ARM Holdings in 2015, and currently agglutinates 57 members with the objective of promoting and standardizing fog computing in diverse capacities and fields.

The combination of fog and cloud computing is expected to bring IoT operations to the next level, disrupting the existing centralized business models.

In current evolution of the fog (edge) computing market we observe an increasing tension to get the major market share for IoT solutions among two different types of providers and vendors. On one hand, traditional IT/Network equipment vendors aim to cover software services in order to become Fog (Edge) Service Providers; while, on the other hand, major public Cloud vendors are pushing to distribute devices at the Edge of the network in connection to their specific Cloud services. Examples of these are summarized in the table below.

	Vendor	IoT Solution Description	Fog Solution	IoT Cloud Solution
Traditional IT/Network vendors	CISCO	IoX Application Enablement Platform offers infrastructure for application hosting and application mobility between cloud and fog (Edge) and middleware services to support app development and lifecycle management.	Fog Director enables to manage, administer, monitor, and troubleshoot fog applications running in the Cisco IOx environment IoX offers hosting capabilities across Cisco IoT network infrastructure.	
	DELL	Dell offering combines Dell Gateway Series for Edge together with Cloud Data Integration platform, Dell Boomi.	Dell Edge Gateway Series offer specific IoT hardware equipped with Intel base Atom processor offering integration with divers protocols and network access means	Dell Boomi offers a complete iPaaS Data and Services integration platform
	INTEL	Reference Architecture details and End to End IoT solution considering Edge and Cloud.	IoT Gateway in reference architecture to provide device management, Edge analytics, security and edge management. Specific product details not provided.	
Traditional Cloud and Software Vendors	Azure IoT Suite	Azure IoT Suite offers a reference architecture aggregating multiple Azure services offering device management, data collection, stream analysis, data and object storage and analytics and visualisation.	Azure has recently publicized its IoT Edge Azure in product preview. This service aims to deliver cloud capabilities to the edge, as a orchestration layer for code and services, to interact among both environments.	IoT Hub offers device message hub, Stream Analytics Object Storage, application and data visualisation means.

	AWS IoT	AWS IoT offering is structured in three main core components: Edge functionality in AWS Greengrass; Cloud IoT services,	AWS Greengrass (current limited preview) offers an Edge computing platform offering local computing thought AWS Serverless technology (AWS Lambda), messaging and data catching while providing interoperability with AWS IoT Cloud services.	AWS IoT Cloud services offers device management, message ques for data ingestion and rules engine complemented with their Data storage and computation offerings.
	Google IoT	Google offers a combination of Cloud and Device services for IoT	Although not tagged as a Edge solution, Android things is providing interesting capacities for Edge solution as mechanism to interact with Google Cloud and device connectivity.	Google Cloud solution for IoT makes specific emphasis on Data Analytics solutions as combination of existing Cloud services.
	SAP	SAP Hana Cloud Platform for the Internet of Things aims to cover both Fog(Edge) and IoT Cloud.	SAP Hana Data management at the Edge tools build upon two main components: SAP SQL Anywhere, providing lightweight database solution for Edge devices and SAP Event Stream Processor.	SAP Hana Cloud Platform offers remote device management services, message management functionalities and communication of this with persistent storage.

Table 2. Examples of current Cloud and Fog IoT solutions

2.1.2. Main actors

Prior to the identification of the main business goals that drive the mF2C architecture, we will identify the main actors that will interact with the system.

Following recommendations provided by ISO standards related to Cloud Computing [8], these actors and roles are classified in three categories: Customer, Provider and Intermediate/Partner Side.

Customer Side

- **Application Developer**, a technical person capable of developing a software application to be operated in an mF2C-powered installation.
- **IoT Platform / Solution Provider**, The provider of an end2end IoT specific solution or Platform candidate to consume as final user mF2C system and services.

Provider Side

- **Fog Service Provider,** the provider of a the provider of a single or multi-fog set of instances and services which comprise and include software, platform and/or infrastructure (compute,

storage, and networking services) operated from geographically dispersed close to the edge clusters of resources.
- **Cloud Service Provider**, the provider of a single or multi-cloud set of services which comprise and include software, platform and/or infrastructure (compute, storage, and networking services) operated from traditional data-centers in private /public or hybrid forms in an "as-a- Service Model".
- **Resource Contributor**, the provider of a resource or a set of them which does not offer it in as-a- Service" but in a contributory model.

Intermediate/Partner Side

- **Fog Equipment Provider,** an equipment provider which sells Fog and Edge specific Hardware.
- **Cloud Equipment Provider,** an equipment provider which sells Cloud capable hardware, traditionally servers.
- **Sensor Provider,** Wireless Sensor Networks hardware manufacturers.
- **Network provider,** Network services provider or operator provided in isolation to additional services commonly tagged as Cloud-services.

It has to be noted that currently in the IoT, Fog (Edge) and Cloud markets, many variations and aggregations of these Actors can be found.

2.1.3. Business Goals

This section presents the list of business goals extracted from market context information.

Field	Description
Unique ID	BG1
Short Name	Connectivity
Type	Business Goal
Description	mF2C must reduce bandwidth usage and improved 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	Extends the cloud to be closer to the "things".
Involved Stakeholders	IoT Platform / Solution Provider

Field	Description
Unique ID	BG2
Short Name	Interoperability
Type	Business Goal
Description	mF2C will ensure interoperability by using widely used de-facto market standards and 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	Ensure interoperability of the mF2C framework with existent solutions.
Involved Stakeholders	IoT Platform / Solution Provider, Fog Service Provider, Cloud Service Provider

Field	Description
Unique ID	BG3
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 business 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

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

Field	Description
Unique ID	BG5
Short Name	Extension and optimization of current solutions
Type	Business Goal
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	Move data processing to the edge.
Involved Stakeholders	IoT Platform / Solution Provider, Fog Service Provider, Cloud Service Provider,

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	Improve reliability and make the process as quickly and smoothly as possible.
Involved Stakeholders	IoT Platform / Solution Provider

2.1.4. mF2C advantage

The mF2C project framework will bring technical and research innovation in the following areas:

- **Management platform and framework:** mF2C proposes a novel management model that will leverage a stack of resources from traditional clouds to the edge.
- **Cloud:** mF2C will evolve cloud computing beyond the concept of hybrid cloud by including fog computing.
- **Fog:** mF2C model supports the creation of fog nodes for allowing the interaction of devices at the edge.
- **Security framework:** mF2C will develop mechanisms for securing information both at the edge and at cloud infrastructures.
- **Programming model:** mF2C will provide a novel programming model for developing applications to be executed in parallel in complex infrastructures.
- **Storage:** mF2C will design a new data model, linked to the programming model, for building applications to be deployed in heterogeneous environments.

2.2 Application and service requirements

To describe which kind of requirements will be imposed by the applications and services running in the mF2C framework, first we need to understand what are the requirements for the three use cases specified in the project proposal:

Emergency Situation Management in Smart City (ESM);

- Smart Boat Use Case (SB);
- Smart Fog-Hub Service (SFHS).

All the above scenarios share a common concern with the fact that nowadays centralized cloud infrastructure might not be the most appropriate solution for use cases where “real-time” data assessment together with high availability connectivity are crucial.

2.2.1. Use Cases Overview

All three use cases are real-world scenarios provided to the mF2C project for validation purposes and have been designed by the mF2C partners Worldsensing, XLAB and Tiscali.

Worldsensing’s use case, ESM, aims on leveraging mF2C’s hybrid architecture to build an alarm manager for smart infrastructures. In this solution, there are several elements which are the devices with sensor capacities LoadSensing [9], Wearables, Jammer detector) and one of the devices acting as a leader of the cluster (Gateway). Normal operation implies that the LoadSensing to send sensor data periodically through the fog device (gateway) up to the cloud (number 1 in the figure). If a sensor reports a value higher than an alarm threshold, the alarm manager will broadcast an alert via WiFi/BT to the wearables nearby and other non-wireless interfaces can be activated (number 3 in the figure). At the same time, the cloud software detects the alarm and will also trigger its alert methods. Furthermore, in order to improve the solution’s security, the alarm manager is able to detect if the LoadSensing and the Gateway aren’t able to communicate (using LoRa interface) to each other (number 1 in the figure). Whenever this lack of communication is detected, the Jammer detector has to be powered up and configured to search for jammers in the channel used for the LoRa communication (number 2 in the Figure 1). After these two steps have been done, an alarm has to be sent through WiFi/BT or other non-wireless interfaces (number 3 in the figure), these non-wireless interfaces could be a signaling light. Triggering the alarm actions using the alarm manager on the fog instead of the cloud will provide a significant advantage to the whole system.

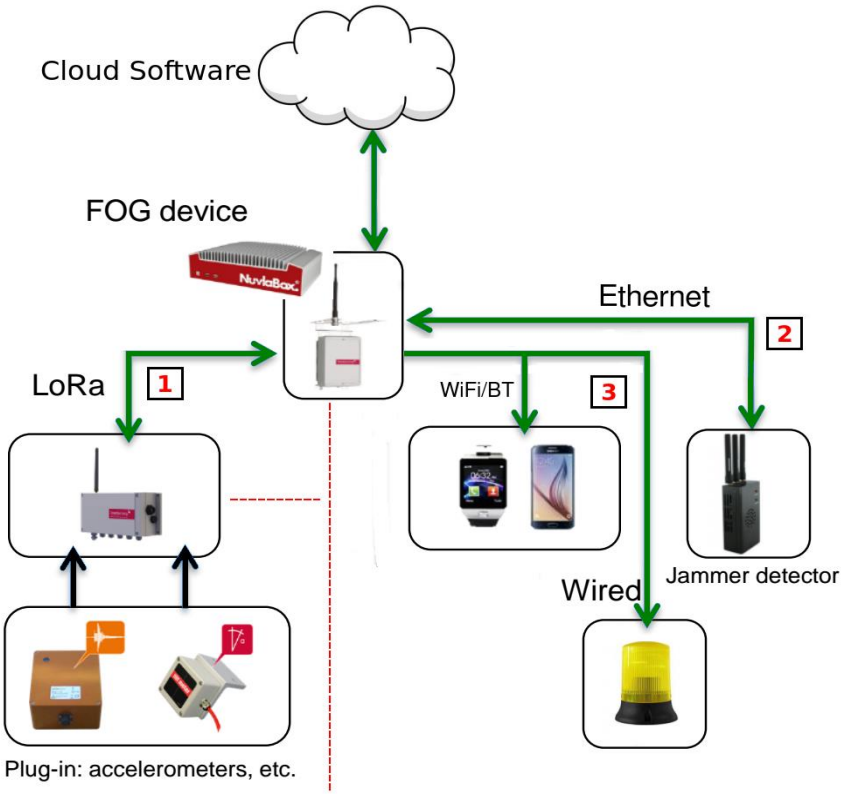

Figure 1: Technologies deployment in the Smart City scenario

XLAB's industrial participation on the project focuses on the Smart Boat use-case based on Sentinel [10], an advanced vessel monitoring device. Sentinel collects data from sensors deployed on the vessel, actuates the specific controls and provides the current state data to the vessel owner or current captain on the vessel. Sentinel was designed by default to work with unreliable networks and low power consumptions, therefore adding more functions and capabilities to the device would make it more power demanding and less attractive to wide marine market segment. Through the mF2C framework however, additional edge/fog devices, deployed on top of Sentinel, can carry the additional processing, communications and sensing power that cannot be supported directly by Sentinel or other marine IoT device. In mF2C architecture the Sentinel device will be employed as a sensor hub as it is presented in Figure 2.

Smart Boat Example

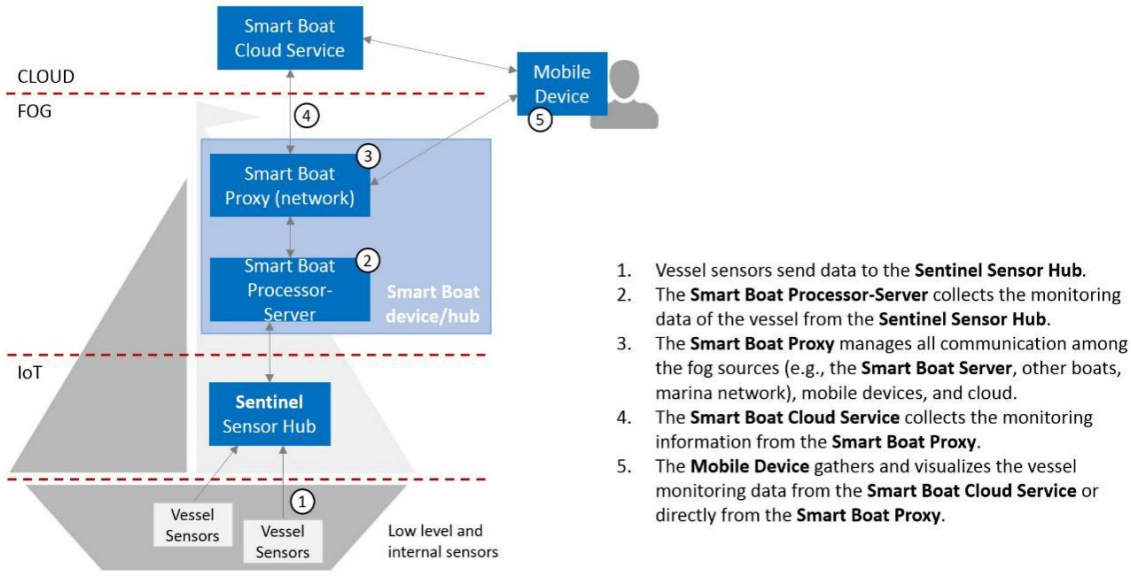

Figure 2: Sentinel as a sensor hub for Smart Boats

Finally, Tiscali's use case is completely experimental and aims at extending the concept of "cloud hub" to a new concept of "fog hub", driven by real market needs. In this scenario Engineering believes that value is generated mostly at the business services level, especially in spaces with recurring concentrations of people and objects that communicate and interact, typical of airports, railway stations, seaports, shopping centres, hospitals, sports facilities, large parking areas, or domestic scenarios with a communal clustering level. Thus the use case is based on setting up a Fog Hub in the Cagliari Elmas airport, with the ability to track the presence of people and other objects in the field, create engagement between them, and developing value added services on top, such as suggestions on shops nearby based on preferences or suggestions on when use airport services to minimize queues or delays.

This use case intends to show the project functionalities and added-values by setting up a scenario developed in the context of an airport, with the objective of proving the great potential value and different business opportunities that can be created in physical environments with a great concentration of smart objects and a Smart Fog Hub Service.

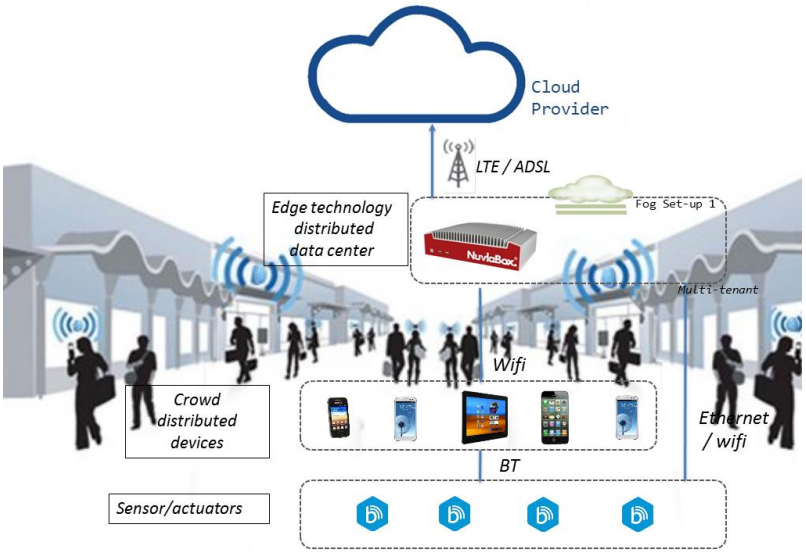

Figure 3: Smart Fog Hub in an Airport

2.2.2. Overall Requirements

For a successful integration with the final mF2C framework, each use case presents a set of requirements that must be addressed in order to meet the proposed business and technical goals.

2.2.2.1 Business goals and requirements

Emergency Situation Management in Smart City: Worldsensing is interested in both the scalable and flexible approach brought by the mF2C resources management and its suitability to deploy services out of a datacentre, and therefore move the computation close to the origin of data. The synergies and the knowledge obtained in the proposed use case will enable more ambitious developments; allowing access to wider markets that are not currently under exploration by Worldsensing due to the limited availability of cost-effective technology in this context.

Smart Boat Use Case: The goal is to provide the possibility of processing and cleaning raw data from the sensors on the boat, including the smartphone and tablet devices. With this expansion user on the boat will be able to receive already enriched data on the smartphone and the link to the cloud will be used only to submit clean and useful data. If the other boats will be in the vicinity, they could share the data and processing power to reach more complex goals.

Smart Fog-Hub Service: Tiscali is looking at the IoT evolution as a potential arena where current cloud offering could be enriched and differentiated, so is interested in setting up (Fog) Hubs in such scenarios to interact with all the objects within the scope of coverage, and to operate "in-proximity" marketing efforts, applying predictive algorithms to track (in an anonymized form) movements, choices and decisions of persons nearby, or even extend the hub with devices (e.g. beacons) capable of sending input (e.g. customized advertising) and determine the effectiveness of the specific campaign in terms of attention/visits rather than conversion (purchasing products / services). Potentially this model could be further extended by making different fogs, perhaps 5-15 km from each other, communicate, and by combining the results in terms of behavioural predictions in adjacent fogs.

2.2.1.2 Technical goals and requirements

Emergency Situation Management in Smart City: The developed management system will facilitate scaling and lowering infrastructure cost, especially for the envisioned smaller-scale deployments that today are too expensive due to the need for over-dimensioned dedicated infrastructure. The mF2C solution will also enable the introduction of new computing paradigms, including edge-technology

computing approach, where computation is moved close to the "edges" of the Internet to reduce risks of failure and delay, and improve reliability in the sense that data is not lost while the edge device is temporarily disconnected. This approach will enable cities to install fog computing infrastructure locally and enable new real-time services. More concretely the mF2C benefits in the context of managing Emergency Situations in smart cities may be summarized as:

- Increasing service reliability by 30% and Quality of service (QoS) by 10%
- Decreasing delays by 30%
- Decreasing operational costs (OPEX) by 10%

Smart Boat Use Case: Sentinel must be expanded with additional hardware and software. The additional Fog layer would be expanded with additional multi-purpose processing devices (like Raspberry PI, Intel Joule or NuvlaBox [11]) which can handle a connection to (BLE communication) Sentinel and additional sensors, like video and sound input, etc. Another important add-on is a gateway capable of WiFi and 3G/4G communication. The additional processing power opens new possibilities, like image or sound recognition and weather prediction calculations. The gateway can be used to maintain the cheapest and most appropriate communication link to the Cloud or other Fog devices with ad-hoc network clustering nearby boats.

Smart Fog-Hub Service (SFHS): The resulting infrastructure will be based on standard components and protocols, and will be sized according to the data volumes, remaining open to use different devices that will be made available by the Project partners. The additional workload on the networking elements will be managed with SDN/NFV to provide bandwidth optimization and low latency, while from a security and privacy perspective Fog and Cloud should be able to use different policies, with anonymization of data when requested. The edge fog element will be configured with some resiliency capabilities, at least for stored data and fast reboot/recovery.

2.2.3. Expected Challenges

Use case	
Emergency Situation Management in Smart City	- Connectivity reliability and availability (packet loss ratio, etc.) Define and measure the parameters to compare the visibility and complexity of the mF2C devices and processes with respect to cloud applications, in terms of delay, reliability, etc.
Smart Boat Use Case	– Create the connection for Sentinel to access all data it can collect – Designing a mF2C powered edge device to manage Sentinel data, sensors and communication gateway – Develop a communication proxy capable of maintaining a connection to the cloud while providing the local network for all additional devices and the ad-hoc capabilities for similar devices in the vicinity – Related to Marine environment, devices must be able to shutdown, suspend or operate in low energy mode
Smart Fog-Hub Service	– Proximity marketing and enhanced user engagement - The strict interaction of huge amount of users enables a much more effective, customized offering and advertising, based on user preferences and behaviour. A continuous refinement in the proposition can be applied. – Data collection and analysis - Collecting lot of data from objects on the move can enable running machine learning algorithms to extract user profiles and demands, but also trace trends or identify new required services. – Social integration - Offering connections and interactive applications gives way for connected users to share or offer some resources,

	under user defined access rules. – Improved performance - Optimal load balance between cloud and fog nodes with low latency – Improved data Privacy and security - Management of user personal data is done at edge level, separated from the cloud, encryption of storage and anonymization techniques are applied before moving data to the cloud, thus reducing the risk of disclosure on data

Table 3. Expected challenges

2.3 Technical requirements

A set of technical requirements must be addressed for a successful development of the mF2C architecture:

- A **Coordinated orchestration** is required to generate individual service workflows, to map one of these service workflows into the devices best suited for the requested service, 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 physical allocated resources. To achieve that, a **semantic adaptation** between the attributes of a service and the capacities offered by the devices is required.
- A coordinated orchestration requires that the proposed mF2C platform must enable a timely creation of **dynamically provisioned F2C infrastructure** (on-demand), automatic **discovery** of resources, sharing all or part of those resources between users, 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 on different physical resources, and even dynamically managing schedules based on runtime conditions. Due to the complexity of the infrastructure, a **high-performance service execution** is necessary through parallel computing.
- Appropriate **runtime system** for this new service execution paradigm shall be considered to optimize the characteristics of the resources brought by the combination of fog and cloud computing. The **service parallelization** must be the runtime system according to the resources availability and characteristics and the service requirements. Strategies based on distributed or centralized services atomization may be considered.
- **Security and privacy** must be addressed at the beginning during the design of each block within the architecture. For that reason, instead of describing security and privacy features for each individual block, a separate section will address those questions for the whole architecture.

3. Architectural Design

This section shows the global design of the system, indicating what the main functionalities are that will be deployed for IT-1. For each individual block of the system, we describe its role and relation within the global architecture, as well as, the required interfaces for the communication between blocks. Regarding communication, we compare different solutions for the system control communication and data communication, and which communication protocols are appropriate for the proposed system. Finally, we show the subsystems required to build the proposed system for all layers covered in the architecture.

3.1 mF2C system description

The mF2C system proposes a coordinated management solution leveraging all existing and potentially available resources, from the edge up to the cloud, when executing a service. Fig. 4 shows a typical scenario where a service is executed using the set of available resources that are made available in a distributed fashion. As it can be seen, heterogeneous devices, computing systems and end-systems are included from sensors and mini-cloud system over to the traditional cloud. When these heterogeneous resources are included into the mF2C system, they are expected to be properly managed according to the service requirements and user needs. To that end, the mF2C system proposes on a layered architecture (described next) where the resources are categorized according to a certain policy and the agent entity deploys the management functionalities in every component within the system.

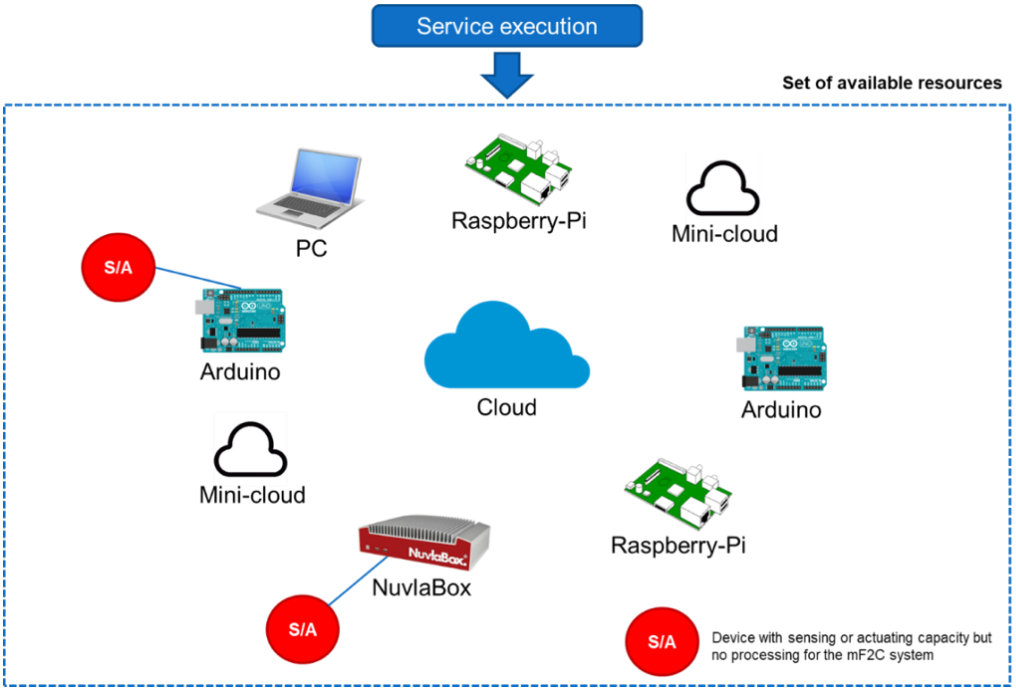

Figure 4: Service execution into the set of available heterogeneous and distributed resources

Fig. 5 is intended to show the two main characteristics of the layered architecture built in mF2C system. We see how different layers are set (from layer 0 at cloud to Layer N+2 at the level closer to the edge) and the fact that an agent is deployed on all components. Fig. 5 also shows edge devices (in red, sensors and actuators with no smart capacities) connected to some nodes in the whole architecture. It is worth mentioning that edge devices may be connected to different nodes located at different layers. Although the connectivity links among nodes will certainly depend on the real connectivity on place, we assume a hierarchical topology for the control functionalities to enable a better deployment of the proposed mF2C management solution. Indeed, Figure 5 depicts such a hierarchical topology for control functions.

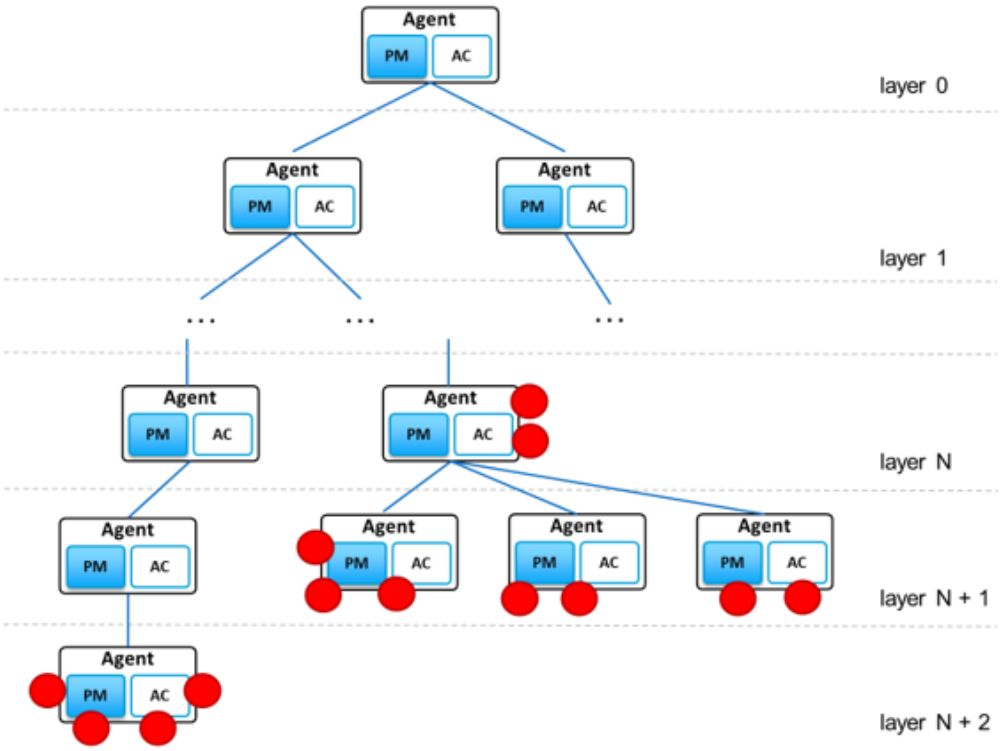

Figure 5: mF2C layered architecture

In the proposed architecture all information regarding the edge devices (sensors, actuators, etc.) is gathered, processed and distributed by the node connecting all edge devices. The policy to identify how and why an edge device is connected to a particular node is still an open issue in the mF2C architecture. Similarly, several nodes are clustered under the control of one node that is defined as the leader which according to the hierarchical model receives aggregated information about the connected devices. Again, the policy to identify the clustering strategy and the leadership role is to be defined.

Another remarkable characteristic of the mF2C system focuses on resource virtualization. Indeed, mF2C will manage resources only considering their capacities and characteristics through categorization and classification mechanisms as well as through abstraction or virtualization strategies, for example, when enabling a sensor with different sensing capacities to be used by different services simultaneously.

The execution of services is designed in the following fashion. When a node receives a request for executing a service (indistinctively on the layer), the Agent on the node must decide whether the required resources are allocated: in the node itself or in any of its connected devices/nodes. If so, the service is executed, otherwise the request is forwarded either up in the hierarchy or horizontally in the same layer, to find out the proper resources. This process ends when either the required services are found –the service can be executed–, or when the system runs out of free resources.

3.1.1.mF2C architecture for IT-1

The mF2C project development has been defined following an iterative approach, thus setting two iterations, the first, IT-1, ending in M18 –providing a first approach to the whole system and including the main architectural functionalities–, and the second, IT-2, in M36 –as a final implementation of the mF2C system. In this Deliverable, we position the main assumptions for IT-1, considering the timeline defined in the project and the main challenges to be considered for

developing a novel architecture as well as to deploy the solution in a preliminary testing and validation phase into the three real-world use cases.

To that end, Fig. 6, depicts the functional control architecture we identify for IT-1, mainly considering the following assumptions:

- Only one cloud is considered
- Only three layers are considered
- There is no horizontal communication among nodes, so service requests are only vertically forwarded
- Only one backup node is considered for robustness
- Only one node acts as leader at any given time
- Resource virtualization is not considered
- Resources information will be stored in a database and its management will follow a simple approach just to verify the system works
- Mobility is only considered at the edge
- Functionalities to be included will be a subset of the total identified, sufficient to show mF2C system performance and benefits.

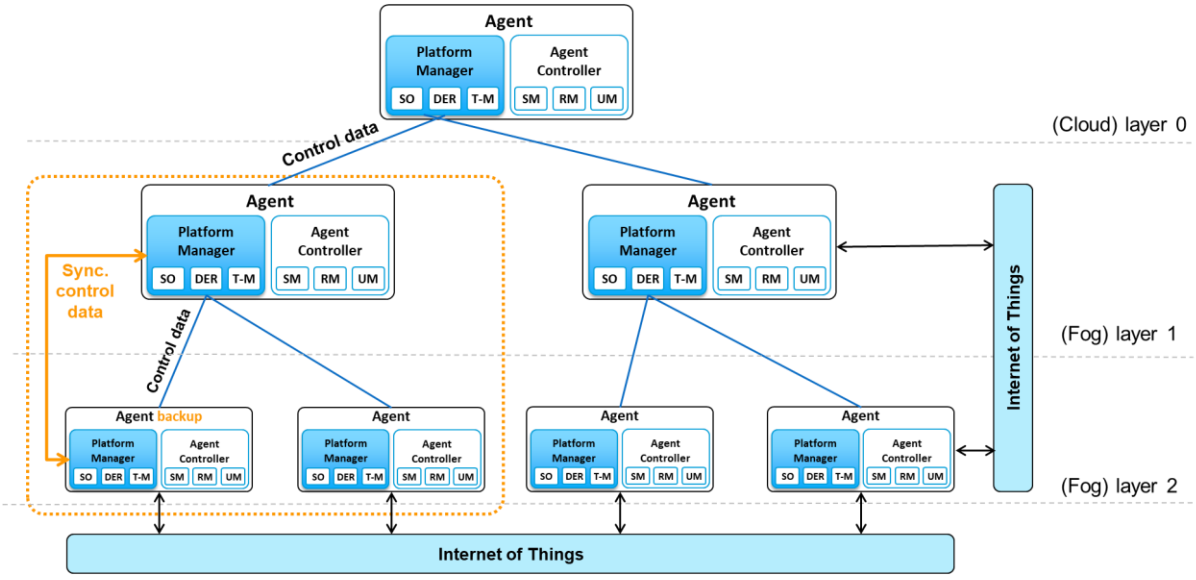

Figure 6: mF2C architecture for IT-1

3.2 Main building blocks

Once the overview of the system is defined, we follow with the description of the agent detailing each individual block and all the related functionalities. It is worth mentioning that, for IT-1, we are constraining the functionalities of the system, but the design of the blocks is not limited to this first version and they can be extended without affecting the global design.

3.2.1. Agent module

The main entity within the mF2C architecture is the agent, deployed in all the resources with computing capacity. The agent structure is the same independently of which layer it is located, for IT-1. The Fig. 7 shows the two main building blocks that comprise the agent entity: **the Platform Manager** and **the Agent Controller**.

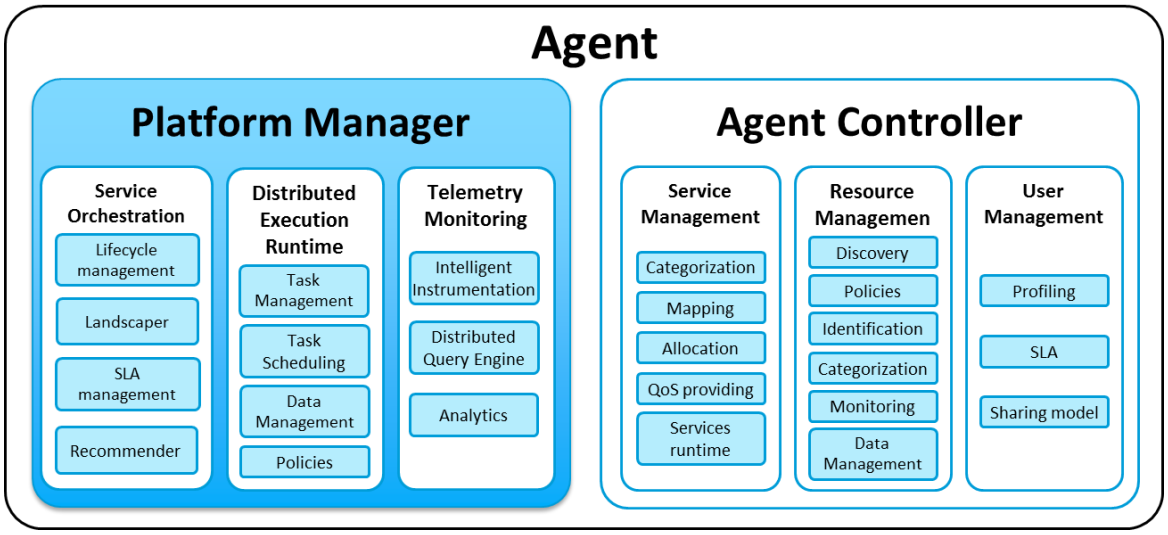

Figure 7: Agent architecture

The Platform Manager is the block responsible for the orchestration of services based on the compute, storage and network resources and using a full-stack monitoring system, which receives telemetry data from different sources. This block also includes the distributed execution runtime, which coordinates the execution of end-user applications within the mf2C infrastructure. The Platform Manager is defined as a global entity that works as a controller, when it is managing agents in lower layers, or that acts as a receiver of control data, when it is being managed by agents from upper layers. Alternatively, the Agent Controller only manages the local resources, services and users of each individual agent. Then, this information is shared with the platform manager, abstracting this information as global resources. This procedure is explained with an example in the Fig. 8.

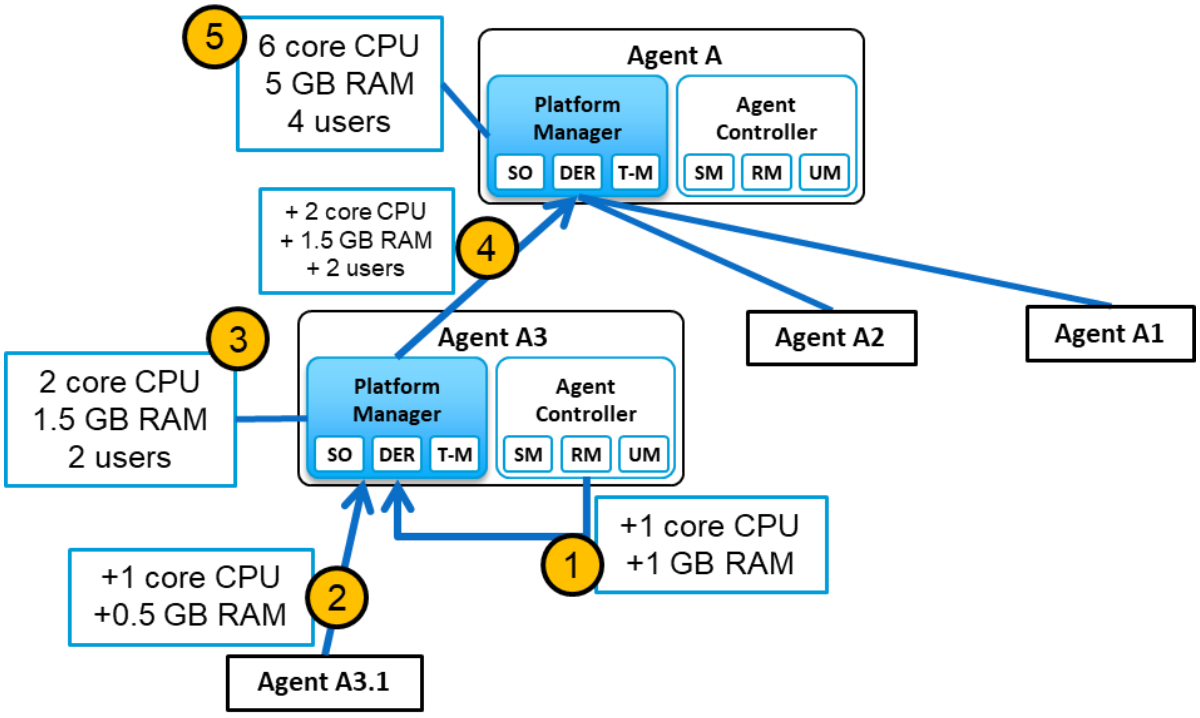

Figure 8: Communication between Platform Manager and Agent Controller

In this example, the agent A is already managing the agents A1 and A2, and a new agent (agent 3) is entering into the system. As step 1, the Agent Controller will notify to the platform manager that this device will share 1 core cpu and 1 GB of RAM with the system. Then, a new agent A3.1, from a lower layer, is also joining the system through the agent A3 sharing 1 core cpu and 0.5 GB of RAM. The platform manager receives and saves this information (step 3) and informs to the upper layer (step 4). Finally, the agent A updates its information, adding the new available resources to the system. This is a simple example about the expected operation between the agent controller and the platform manager, and is not defining the final real operation. In the following sections, we detail the specific functionalities of each block and the required interfaces.

3.2.2. Agent Controller

This section outlines the main components for the Agent Controller which, as previously described, deals with local resources, services and users. We categorize the set of functionalities for the Agent Controller into three main blocks, the Services, the Resources and the Users. The first will mainly encompass all functionalities dealing with resources management, for example including discovery, naming, categorization, etc. The second will include functionalities related to services, such as orchestration, runtime model, etc. Finally, the third one will include functionalities, such as QoS, SLA or user profiling, highly related to the user expectations, demands and characteristics.

It is also mandatory to define the set of interfaces connecting the different blocks within the Agent Controller, but also those connecting the Agent Controller to the Platform Manager. At the time of this deliverable, such a need is identified and left for work to be done in the deployment of both components.

The next subsections will go deeply into each one of the functionalities for each block. However, it is worth mentioning that this deliverable is aimed at identify components, blocks and conceptual definitions of the functionalities envisioned to build the Agent Controller. Thus, at this stage, the next subsections are only intended to that objective and a more elaborated and deep description is left for further deliverables.

3.2.2.1 Service management

Mapping

The service management block in the agent controller will be responsible for the mapping of local services. Since an agent can be used to execute different services due to the variety of resources such as, storage, cpu, sensors, etc., this block has to allocate global service requests coming from the platform manager into the local resources. For instance, a car running an mF2C agent requires the traffic status for the highway, then, other cars on the same highway will receive this service request from their platform manager and the platform manager will map this request to their local service management blocks.

Allocation

The allocation block within the service management will be notified of the incoming service and will communicate to the resource management block in order to provide the service though the available resources. Following the previous example, the allocation block will ask for the GPS location of the car to the resource management.

Categorization

The categorization of services block classifies the different type of services based on their specific characteristics. The functionality of this block will work in cooperation with the categorization block inside the resource management. In the previous example, this block will determine that the user is requiring for the current traffic conditions.

QoS providing

The service management block also needs to check and ensures that the service meets the requirements specified by the user and the resource management module. Following the previous example, this block will check if the precision of the GPS location provided is good enough to consider it or not.

Service Runtime

The service runtime block is controlling the service execution of the local resources. This block knows the current status of each service: in queue, delayed, executed, on execution, etc., and how is the execution order based on specific priorities in case to be applied.

3.2.2.2 Resource management including semantic adaptation

Discovery (registration, detection)

In order to ensure a seamless execution of fog applications, idle resources, located at the different layers of the F2C hierarchy, have to be efficiently discovered. This is achieved by the resource discovery module, which hosts mechanisms allowing the agent to become aware of nearby resources capable of contributing to the mF2C system and to register them upon mutual agreement. It also allows the agent to announce its presence and characteristics in order to be discoverable by other nodes willing to join the system either as contributors or as clients.

This process will be implemented with the following considerations in mind. First, given the heterogeneous nature of the mF2C landscape, it will have to detect all sorts of resources regardless of their communication technologies, vendors, hardware platforms etc. Second, it has to provide mechanisms for timely reactions to dynamic changes in topology, either because nodes offering resources may be moving from one area to another or because of intermittent connectivity issues. This will ensure that the agent always has an accurate and up-to-date view on resources within its range. In addition, the discovery process should have a light footprint mainly on devices with moderate to limited energy, processing power and storage capacities. Scalability is also an important aspect to consider, either in terms of increasing number of resources capable of joining the system, or in terms of user demands for these resources. This is one of the aspects where the decentralized hierarchical topology of mF2C is most beneficial. Finally, given that there is no single trusted organization owning and controlling the resources, mutual trust has to be established between the involved parties to realize the envisioned potential of mF2C.

Besides considerations related to the characteristics of the resources, the main reasons driving the need for fog computing should also be kept in mind during the discovery phase. These include the need to save bandwidth and to reduce the network traffic, which will be addressed by designing an efficient discovery message exchange strategy. Furthermore, given that fog applications generally have real-time constraints and strict QoS attributes, the discovery mechanism will guarantee that the requested results are provided in time. In particular, higher priorities will be assigned to discovery requests coming from critical and safety-related applications, as opposed to requests related to less urgent ones.

Resource identification and naming

The resource identification refers to the adoption of mechanisms for the unambiguous identification of physical and virtual objects, such as computers, sensors, actuators, networking devices, smartphones, tablets, wearables, virtual machines and more. For this reason, the adoption of a naming technique that allows to strategically assigning unique names to those entities is envisioned as a key perquisite for the development, deployment and operation of the mF2C platform [12].

In [12] and [13] an extensive list of naming schemes that have already been proposed and used is presented and explained. Nevertheless, none of them was designed to be applied to an mF2C

context, what makes its possible adoption pretty complex, basically because any existing strategy meets the mF2C requirements, like mobility support, interoperability, uniqueness, etc.

In order to implement this mandatory functionality, emergent naming approaches can be used to create an identification strategy for the edge devices. One of these approaches is the hash-based technique, which consist of the transformation of a variable size string into a meaningless fixed length string that is suitable to be used as unique name for the object (physical or virtual).

Resource categorization, including database design

Resource categorization stands for classifying resources according to their inherent characteristics. This functionality is crucial to guarantee an optimal matching between resources capacities and services demands. Once resources are categorized, it is also necessary to design a database to store the resources related information. The database should contain the information about networking status, processing and storage capacity, resource id, information about the resource allocation, as well as information about the energy consumption.

Assuming the IT-1 scenario, the mF2C control architecture is built on a three layered architecture. The lowest layer at the bottom, referred to as Layer 2, contains all edge devices, that is, nodes with the lowest computing capacity, such as smart phones, single computer boards (Raspberries, etc.). We consider that IoT devices such as sensors and actuators, are characteristics of these computer devices, and do not constitute per se an additional layer. On the top of this layer, referred to as Layer 1, there will be devices with medium computing capacity, such as PCs, routers, gateways, and even mini clouds. Finally, on the top of this layer, we have the cloud layer, referred to as Layer 0. In this hierarchical architecture, it is very important to categorize and classify the distinct resources according to their characteristics for each layer. In the categorization of resources the following characteristics for resources may be considered (at this preliminary stage):

- *Communication & Networking* – Networking Type, Communication Time, Communication Enablers
- *Storage Capacity* – Small, Medium, High
- *Processing* – Processing Capacity
- *Platform* – Open Source or Not
- *Mobility* - Static or Mobile
- *Memory* – Size of the Memory
- *Security* – It can be categorised as Hardware and Software Security
- *Life Span* – Duration to die out of a resource
- *Services offered by the resources* – Type of Services, Duration of offering the services
- *Cost* – For Processing, Networking, Deployment, using the services
- A*utonomous* – Resources have autonomous characteristics or not
- *Power Consumption* – Source of Power, rating for power consumption

Some of these characteristics are more suitable for resources at the Edge Layer (Layer 2), for example, mobility or power consumption, but all resources from all layers must be categorized according to similar characteristics. Additionally, for the Cloud Layer, we should also consider the previously mentioned characteristics, as well as some other specific cloud data, such as the *cloud federation information*.

After categorizing the resources, it is necessary to keep, to maintain and to preserve the resources information for further processing or future analysis. To that end, a database must be designed. When designing several challenges come up, such as security, real time access, reliability and so on. Another constraint to capture in the database is the capacity of the devices. Devices in the lowest layer may have lower storage and processing capabilities than devices in a higher layer. Taking into account this inherent characteristic in the designed architecture, it would be necessary to design some light-weight and real time accessible database in the lowest layers.

Policies

The policies are a set of rules that will define the way in which the devices in the mF2C system are grouped and organized. These policies should define the different roles existing in mF2C system, describe the functions of each role and regulate the role assignment to the fog nodes.

Those policies can be set to prioritize one or more desirable characteristic of the nodes acting as leader or sub-leader. For example, if one of the requirements of the leader node is a long availability time, a policy discarding all the mobile nodes for this specific role could be a good start. For instance, it may be defined by a policy that a leader node needs to have a certain amount of compute and storage resources, as well as good energy management in case of wireless devices.

The main aspects that the policies should rule are:

- The role assignment.
- The associated functions of every role.
- The maintenance tasks.
- The nodes aggrupation.
- The way how the nodes interact among them.

In [14] and [15] the authors present two resource discovery approaches and in their works they include some policies for the node aggrupation and the role assignment that may be used in the mF2C. In any case, those rules should be refined to make them match the paradigm requirements.

Data management

Data management is the module responsible for organizing all data resources from the mF2C system, and provide an easy-to-access interface for accessing such data. For this reason, this module must know which devices can provide any data resource to the system, and maintain a context-based directory structure that facilitates data search and access. The interface provided will allow data requests by type of data (for instance, in IoT, what type of sensor feature), location (a specific place or an area), and time (a specific time, a time range, or real-time streaming).

Resource monitoring

In mF2C, the resource monitoring will be a utility, intended to facilitate the collection of information about the current status of the existing resources. For example, in case of a node acting as agent (not being a leader), the resource monitoring should collect the information about the status of the IoT resources (sensors and actuators) directly connected to the agent. On the other hand, in the case of an agent acting as leader, it should monitor and collect information about the rest of the agents under its control, as well as the possible IoT devices directly connected to it.

Some of the main characteristics an agent must monitor are the availability, the status and the reliability of their connected resources. In mF2C, this module should work in a real time basis, to find out which resources are available but also identify those running differently than expected (malfunction, etc.). By monitoring the resources, we can also reduce the overhead of the database by cleaning up the unnecessary, died out or malfunctioned resources.

The resource management monitoring module is also responsible for monitoring the resources (CPU, RAM, etc.) used by the mF2C services running in the device. The information collected by this component will be used later by the user management QoS enforcement module.

3.2.2.3 User management

This module provides all the functionality related to the management of the profiles, the quality of service enforcement and the sharing model of the users who have access to the mF2C system and the various services running on top. This includes aspects related to the definition of rules for the shareable resources residing in the devices (CPU, disk, battery, etc.) and the user roles definition.

Finally, an appropriate interface should enable access to other Agent Controller modules in order to enable the quality of service enforcement (profile, and shared resources' rules), and the access to all the information required by the Platform Manager, among other functionalities.

Profiling

The profiling module is responsible for the definition and the management of the user's roles and permissions, such as how to join an mF2C cluster, and the list of mF2C services with permission to run in the device. The characteristics that should be defined and managed by this module are the following:

- Turn on / off the mF2C functionalities: manually or based on rules (e.g. turn off the mF2C functionalities when the device's battery is too low).
- How to join the mF2C system: as a contributor or as a client
- Permissions:
  - mF2C services allowed to make use of the device´s resources
  - mF2C services allowed to be used by the user
- Definition of the maximum number of mF2C services to run in the device.

Sharing model

This module is responsible for the definition of the device's shareable resources. The user should be able to define the amount of physical resources that should be given to the mF2C system. The resources to be shared are the following: CPU, RAM, disk and bandwidth. Some kind of rules or constraints should also be defined in order to establish not only the amount of resources to be shared, but also the conditions from which these resources should be increased, decreased or not shared at all. For example, if the device's battery is low then the amount of shared resources should be decreased or stopped. This includes the following sharing resources definitions:

- Maximum CPU percentage usage that should be dedicated to the mF2C services.
- Maximum RAM usage.
- Maximum disk usage.
- Maximum amount of bandwidth usage.
- Battery usage limits. Definition of the battery levels at which the shared resources should be lowered.

Finally, in order to encourage users to share their resources to the mF2C system, this module should also allow the definition of reward mechanisms for these resource contributions, like some kind of service execution credits, economic rewards, etc.

SLA/QoS enforcement

This module checks and enforces that the mF2C services running on the device meet the user profile and sharing model rules. On one hand it should be able to get and analyse the data related to the use of the device's resources by these mF2C services. It will also track the availability of these resources in order to assess the reliability of them. In order to achieve this, it should get access to the data related to the connections, disconnections, average connection times, battery status, etc. Then, on the other hand it should be able to invoke the required actions that ensure the implementation of the defined rules and the user profile's defined properties.

To make these functionalities possible, the User Management module should provide an interface to other modules, including the Platform manager, can access information related to the user's profile and shared resources. And finally, this module should also be able to connect to the other Agent Controller modules, in order to get all the resources information required to enforce the QoS, and to invoke the required actions when needed.

3.2.2.4 Interfaces

Given the different modules described for the Agent Controller architecture, it is necessary to define a set of interfaces for each module that will allow the interaction with them and the rest of the system. Next, the functions that need to be supported through interfaces identified for every module are described:

Service Management

- **Add / Remove Service**: firstly, there must be a way to allocate new services and remove others in order to provide proper service management.
- **Get / Set Service Category**: all services will be categorized and it is required to manage these categories.
- **Set Service Resources**: every service will need a minimum of resources in order to run properly. This interface will allow the mapping between services and resources. There could also be recommended and maximum resource usage setting.
- **Get / Set Service QoS**: several QoS parameters (e.g. latency) can be defined and taken into account by the Service Manager for the proper behaviour of the services.
- **Run Service**: a way to schedule between services is needed in order to run each one with the proper QoS.

Resource Management

- **Register Agent Resources**: announce resources availability for an agent. This interface will allow an agent to join the mF2C system.
- **Get Agent Resources**: this request will be mainly called for detection of the resources that can be shared by an agent.
- **Get / Set Agent Role**: the main roles identified so far are leader or sub-leader. In this case, the role will be assigned to an agent and retrieved from it.
- **Get / Set Agent Role functions**: allows the configuration of the agent's role functions.
- **Get / Set Resource Id**: every resource will be identified by unique ids and there must be a way to manage them.
- **Get / Set Resource Category**: communication, storage capacity, memory and processing are some of the categories of the resources. Every resource needs to be categorized using this interface.
- **Get Resource State**: resources' characteristics, such as status, availability and reliability need to be maintained in order to define the state of the resource in real time. This interface aims to provide a way to retrieve the resource state.
- **Get Resource Data**: data resources will provide data which can be requested by type, location or time.

User Management

- **Create / Delete Profile**: user profiles are to be created, modified and deleted on demand. The type of profile needs to be specified. So far there are client or contributor profile types.
- **Get / Set Profile Service Permissions**: this interface is used to configure how services are allowed to use device's resources.
- **Get / Set Profile User Service Permissions**: user permissions to access different services need to be defined too.
- **Get / Set Profile Max Services**: the profile will define a maximum number of services that can run on the device.
- **Get / Set Profile Resource Shareable Limit**: every resource (CPU processing, memory, etc.) will define shearing maximum limits on a given profile. This interface will allow the system to manage those limits for any given resource.

- **Get / Set Profile Limit Policies**: the limitation of the resource sharing must be defined as a policy within the profile. For example, stop sharing when running on low battery.
- Enable / Disable Sharing functionalities**: mF2C functionalities for sharing resources need to be enabled or disabled depending on the profile defined and its policies.**

3.2.3. Platform Manager

The Platform Manager is the block responsible for the orchestration of services based on the compute, storage and network resources and using a full-stack monitoring system, which receives telemetry data from different sources. This block is also responsible for coordinating the distributed execution of services and applications within the mF2C infrastructure.

The Platform Manager is divided into three main components according to these responsibilities: Service Orchestration, Distributed Execution Runtime, and Telemetry. Each of the components and their specific functionalities are detailed in the following subsections. The last subsection explains how a generic interface for orchestration and deployment of services in different kinds of devices is provided, thus abstracting users from the underlying resource diversity.

3.2.3.1 Service Orchestration including Landscaping module including SLA control

The *Service Orchestration* component will be responsible for allocating services to the most suitable resources producing optimal performance and efficient use of those resources. Service placements will be dependent on the analysis of historical invocations of similar service descriptors, and configuration of available resources in real time, including SLA, and QoS. *Lifecycle Management* will control the different phases of a Service being executed. The physical and logical infrastructure topology of the orchestrated apps/service on these resources will be stored in the *Landscaper* component. *SLA Management* will guarantee end users expectations regarding service performance and quality. Finally, the *Recommender* will store recipes and heuristics resulting from the analytics performed.

Lifecycle Management

Lifecycle management enables control of diverse phases in the status of a Service to be executed in mF2C infrastructure. These phases (deploy, start, stop, etc.) correspond to the Service Management interfaces defined in previous sections (2.2.2.4).

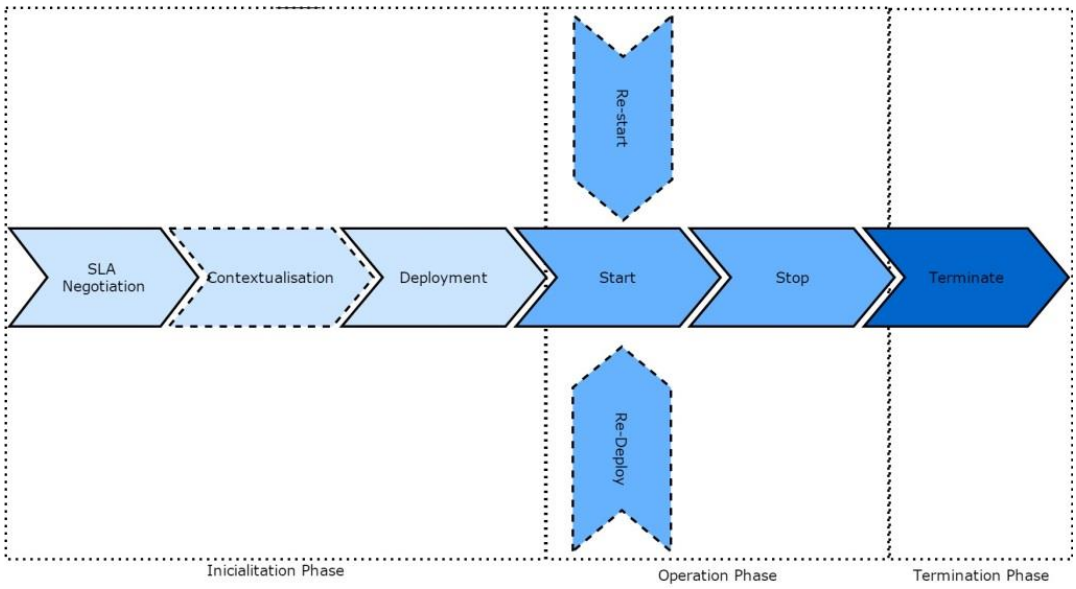

Figure 9: Phases and operations in service lifecycle

The service lifecycle considers three distinguished phases which determine different operations over a service:

Initialization phase: aggregates the necessary steps for service deployment, namely negotiation with available mF2C Agent installations (through the Leader) to get appropriate deployment candidates; contextualization as an optimal step that performs necessary configurations to adequate the service components to the candidate Agent infrastructures and actual deployment into the target installations.

Operation phase: once the service is deployed, it can be started and stopped by the user. Optionally the user can request redeployment of the service or due to SLA Enactment or scalability operation, and then it can be redeployed.

Termination phase: is the final step after stopping the service, and consists in releasing all resources the service had made use of.

These three phases result in the following Service operations and states workflows in the service lifecycle:

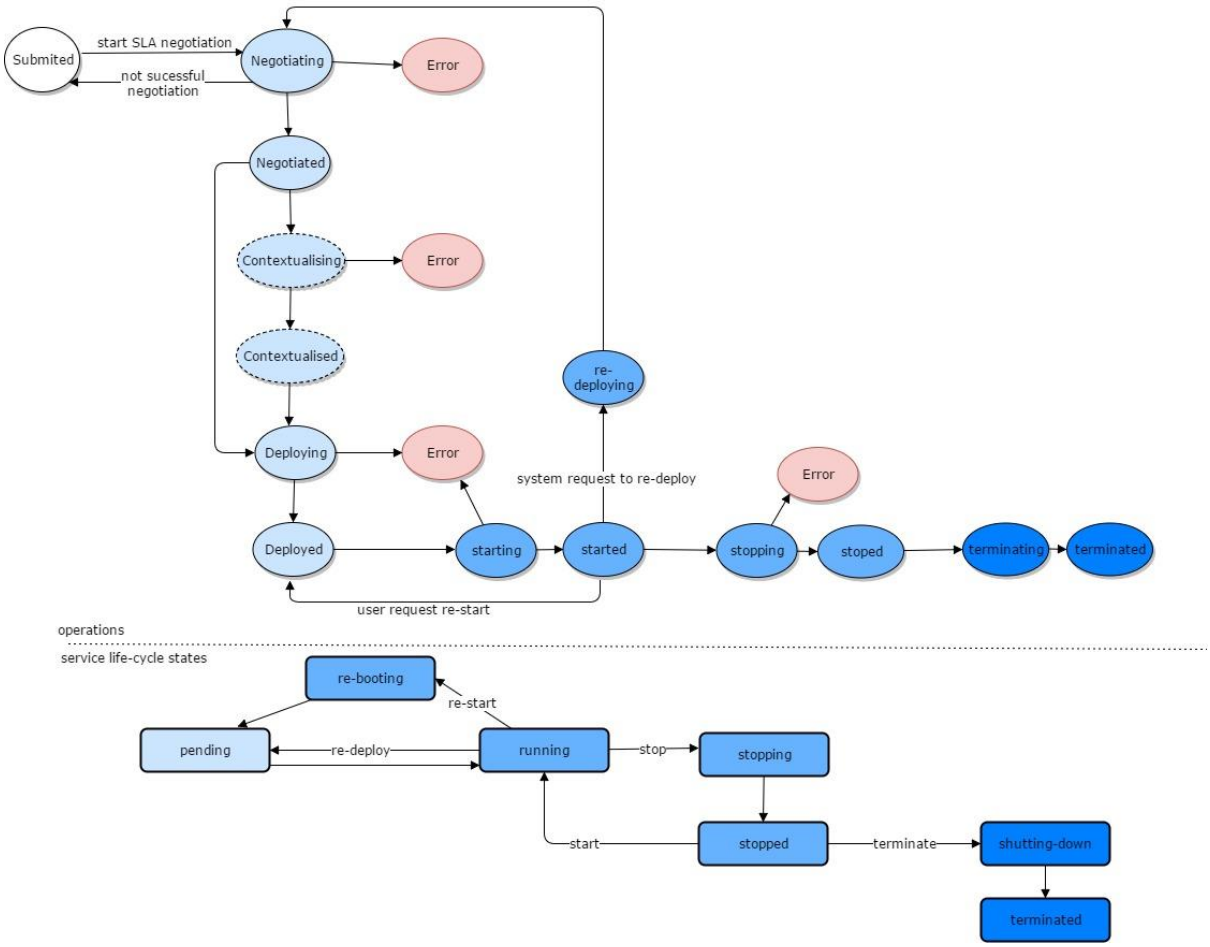

Figure 10: Service Operations and life-cycle states

Landscaper

The *Landscaper* Module expresses both the physical and logical infrastructure topology of the different layers of the F2C hierarchy, including the cloud, fogs and IoT devices. It will be more advanced than a standard Configuration Management Database (CMDB), storing both functional and configuration information for each F2C resource element. By storing the discovered resources in a

graph database allows us to generate a series of planes or graphs representing the physical, logical and various service layers. Historical values will also be retained so that a landscape snapshot of the cloud, fog or any permutation of the F2C hierarchy can be generated for any given date/time.

By mapping telemetry data from a time series database, causalities can be inferred for a deployed service stack. As the landscape will store historical values, it will allow service's performance to be analysed against its deployment configuration to identify optimizations and improve service assurance.

As the landscape is represented as a graph, we can optimize (sub) graphs of the overall landscape as shown in Figure 11.

- The ***Service Layer*** captures the resource elements making up the services currently deployed, and therefore the application workload.
- All Software Defined entities (SDx) reside in the ***Virtual Layer***. These include Containers, Virtual Machines (VM's), Software Defined Networking (SDN) and Software Defined Storage (SDS) entities.
- In the ***Physical Layer***, these are mapped to the actual physical resources of the compute node such as memory, disks, NICs, CPUs, etc.

The landscape module stretches a Software Defined Infrastructure end-to-end from the IoT device all the way to a service in the data centre.

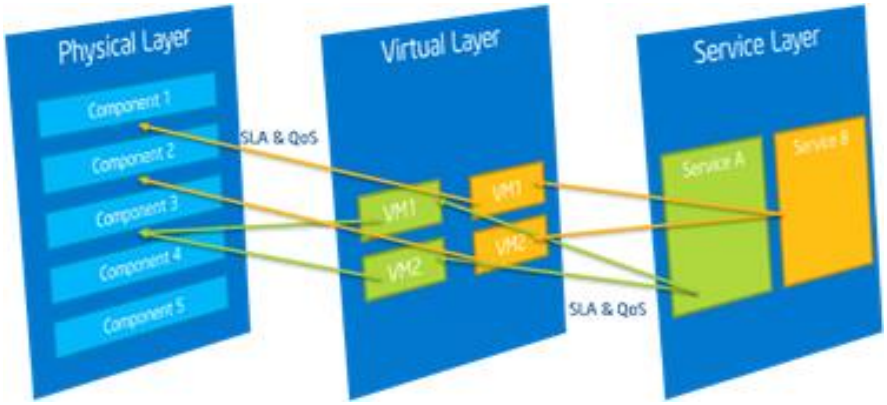

Figure 11: Different layers of the landscape: Physical/Virtual/Service

Examples of Information models that can now be mapped include layers across the stack showing:

- One service running across multiple VMs
- Placement of the VMs on different hardware infrastructure
- Placement of the storage
- SDN entities (vRouters, vSwitches, Controllers supporting VLANs and hence the connectivity)

SLA Management

In the mF2C context, we can find two different types of applications: the set of applications and tools that compose the **mF2C system** and the the (deployed) **user services** running on it. In both cases, but especially in the second case, the mF2C solution needs to guarantee their customers and final users their expectations with regard to the service's performance and quality.

Service-level agreements (SLAs) typically provide guarantees in terms of Quality of Service (**QoS**) and Quality of Business (**QoB**). While the QoS consists of a set of criteria for an end user, like the availability of the applications, services and systems, the performance of them, etc., the

enforcement of business oriented policies (QoB) represents a constraint on a metric that impact on the business of the application, and the business actions to apply in case of violation.

The SLA control module, based on the Atos SLA Framework described in section 2.3.1, will be responsible for doing these tasks and its main responsibilities are: generating and storing the Web Services Agreement Specifications (**WS-Agreements**[2]) templates and agreements, assessing that all the agreements (SLA guarantees) are respected by evaluating the business rules, and taking the business actions defined in these agreements.

These templates or agreements contain functional and non-functional terms that describe the service being delivered. In mF2C, we are mostly interested in non-functional terms (guarantee terms), where a Service Level Objective (**SLO**) is defined as a constraint on a metric, and a list of business values describing the result of not fulfilling an objective. For example, a QoB rule could be defined (for a specific mF2C service running in the mF2C solution) in the following way:

- A **constraint over a metric** (SLO) provided by external sensors or monitoring tools: *response time < 2500ms*.

A non-fulfilled constraint is considered a breach.

- A **time window** where the number of breaches must be below a threshold (e.g. 20 breaches /day). If this time window is violated, a QoB violation is raised, and the business values take place:
  - Discounts, down rating, other recovery actions, etc.
- The **business action** that takes place in the case of violation: this action may also be defined inside a time window. For example, 5 violations of a specific constraint in a day are penalized with some kind of discount, and 20 violations in a day are penalized with a trial of a service during one month.

In mF2C, we identify two levels of SLA, depending of the involved parties of the agreement:

- [user service – end users]

  This level describes the service offered by the user services providers to their end users. In this SLA level, there is one template per full application. When generating the agreement from the template, the customer party in the agreement may be not an actual customer, but a generic one. This makes sense on applications that offer a free service to their users and do not require a previous registration.

- [mF2C system - user services]

  This level describes the service offered by the mF2C system to the user services providers. In this SLA level, there is one agreement for each mF2C Provider. The SLA control module will put the focus on this level.

Finally, this module will connect to the Telemetry/Monitoring module, in order to get all the information required by the SLA tool.

Recommender

The main function of the *Analytics* module in the *Telemetry/Monitoring* component, described in section 2.2.3.2, is applying machine learning and data mining algorithms to the telemetry captured and mapping it to the landscape data sources. This actuates and feeds *Lifecycle Management*. The results of these analytics are heuristics and models which will take the form of decision trees. They

[2] A **WS-Agreement** proposes a standard for describing SLAs to specify "which" service will offer and "how" it is offered. Or in other words, according to the Open Grid Forum, WS-Agreement is a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer

are a mathematical expression by nature and specific to the use case we try to solve and level of detail required. The *Recommender* module will store these models.

To actuate and feed *Lifecycle Management*, we derive recipes. Recipes express simple decision flows that a scheduler can then perform. There is an expectation that recipes will change over time due to changes in the landscape. This drives the need to store historical landscapes over time, and just like the heuristics and models be retrained.

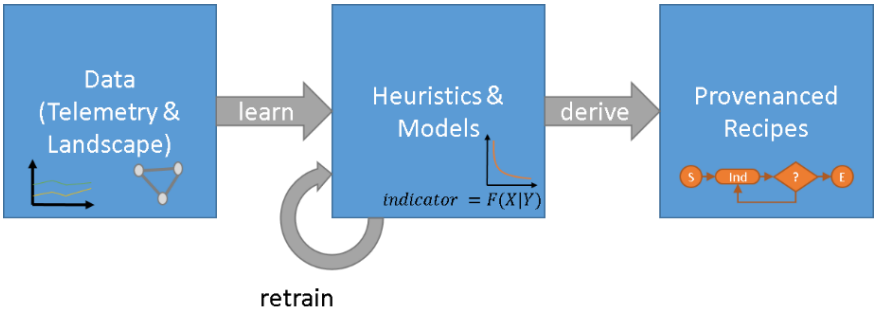

Figure 12: Lifecycle of creating models and recipes

3.2.3.2 Distributed Execution Runtime

The distributed execution runtime participates in the operation phase of the services executed in the platform, and is responsible for coordinating the execution of end-user services or applications in the mF2C infrastructure. The project will follow a task-based approach, where applications are composed of tasks. A task is a piece of the application that implies some computation, and receives a set of inputs and generates a given output.

The runtime will take care of the whole application life-cycle: task creation and data-dependence detection, task scheduling and synchronization taking into account the available resources and the location of the data, and data transfer (if required).

Task Management

This functionality is responsible for detecting the applications' tasks and find data dependences between them. A data dependency between two tasks exists when one task consumes an output of another task. With this information, the runtime builds a data-dependence task graph or workflow, which is an abstract representation of the application behaviour. This task-graph exposes which tasks can be executed in parallel and which ones should wait for others to complete before being executed. This process is illustrated in steps (1)-(2) in Figure 13.

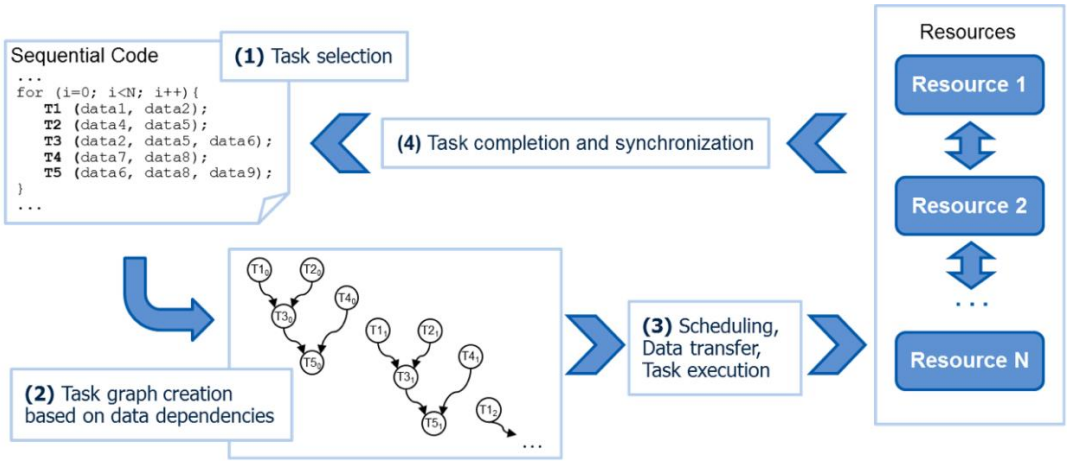

Figure 13: Distributed execution runtime overview

Task Scheduling

The runtime schedules the different tasks in the distributed computing platform, being able to select the more adequate resources. As well as the dependencies shown in the task-graph, additional information to take into account when scheduling the tasks is where the data is located and which resources can perform the specific task (due to hardware or software requirements). Once the decisions on scheduling are taken, the runtime requests the execution of tasks in the appropriate resources and takes care of monitoring the completion of the tasks retrieving the generated data in order to keep the sequential consistency of the application flow (steps (3)-(4) in Figure 13).

Data Management

The runtime delegates storage and data management tasks to a distributed storage platform that handles data in the form of objects, either persistent or volatile. This storage platform provides information to the runtime about where data objects are located (i.e. a data directory), so that the runtime can decide to execute each task in the most appropriate location to avoid data transfers as much as possible.

In case all the objects required by the task are not in the same location, the storage platform accesses them without requiring any action from the runtime. However, the platform provides a set of functionalities to replicate, move, or version data in different locations in case it is required by the runtime for performance and scalability issues.

Policies

The policies in the execution environment define how tasks are assigned to resources, and how data is distributed in the storage platform. These policies help to identify the most appropriate resource to execute a task, taking into account its current state and the data it contains. Also, data policies such as replication or distribution policies will be taken into account to improve performance and reliability. The runtime will prioritize policies according to the current state of the execution.

3.2.3.3 Telemetry/Monitoring including SLA control and logging

Intelligent Instrumentation

This module will provide the telemetry collectors that capture the raw data. By increasing the resolution and simultaneously monitoring the entire system stack to generate and derive metrics (including domain or service specific) in a combined namespace will provide context. These include:

- Offline (pointing the fire-hose at a means to extract patterns and relationships)
- Online (rapid response to the API style queries that help guide runtime scheduling and placement decisions)

- In-band (OS, middleware and application metrics)
- Out-of-band (BMC, rack controller counters)

Instrumentation will be carried out by the telemetry framework and a collection of customized modules (plug-ins) that will allow the instrumentation of the "full stack" including: Benchmark (e.g. per query latency), Middleware/Database, system metrics (e.g. per core, per socket, system utilization), hardware counters (e.g. Processor Counter Monitor) and environment (e.g. temperature, power consumption).

The telemetry framework will allow complex flows of telemetry that can be easily constructed and managed. Data can be captured from hardware, both in-band and out-of-band - many modern servers include dedicated management systems independent to the main CPU's. Collectors will be software probes to collect any metric from any software source: host operating system, hypervisor, guest operating system, middleware, or hosted application.

Captured data can be passed through local filters or aggregators that analyse and perform some action on the data. The results of one probe, e.g., battery life, may impact the collection of a separate probe, e.g., Disk I/O, on the same device with one metric potentially causing the throttling of the other.

Processed data will then be published to arbitrary destinations. Endpoints can include SQL and NoSQL databases, message queues, or analytics engines.

Distributed Query Engine

As collectors are capturing telemetry at various locations in the hierarchy of the mF2C system, some services may require aggregation at certain levels. The *Distributed Query Engine* would allow for the querying of telemetry data through either local storage or the ability to provide access to the data through remote querying.

Local storage would be populated based by aggregation services that are constantly processing the data into usable high level values. However, on when anomalies are detected, these events may require more granular data so the query engine should broker the querying of the original raw data from original sources.

Analytics

Data will flow through the *Analytics* module from the multiple instrumentation sources allowing it to perform a number of tasks. An extract of the Landscape will be queried and mapped to the telemetry allowing the different analytics algorithms to contextualize the landscape and make informed decisions. The derived heuristics and models will then be fed back to the Recommender system. This allows the potential to have the recipes evolve over time. Analytics may suggest recommendations to actuate directly on currently running services and immediately trigger change requests to the current system.

As there is more than one way to realize the actuation triggers and feeds, the analytics system should support:

- Streaming analytics: The analytics are run on a stream of data (including Log data) and actuation triggers send immediately, eg, anomaly detection.
- Offline – Analytics are run on large dataset, so runtime might be lengthy. This would lead to the creation of heuristics and models using e.g. decision trees.
- Complex Event Processing – This is analysing real-time events in the system and derives heuristics and models from them.
- Continuous analytics – can be applied to combinations of these processes. This allows for extremely long running analytics services.

On-demand – This infers whenever required, a data centre owner should be able to trigger an analysis cycle, e.g., to create a report for capacity planning.

3.2.3.4 Interfaces

According to the agent architecture design and the functionalities detailed in previous sections, the need for several interfaces between the different blocks and components can be identified. The concrete design of these interfaces will be described in next deliverables, namely D3.3 and D4.3, which will detail the design of the mF2C Agent Controller and Platform Manager blocks, respectively.

At the current time, for each Platform Manager component, we identify which are the main interfaces of the Agent Controller used by the component, and also the functionalities that each Platform Manager component needs to expose in its interface. These interfaces will be used by other Platform Manager components, either in the same or in other agents, since the Platform Manager is the block through which different agents interact.

Service orchestration

This component uses the *Service Management* and *Resource Management* interfaces of the Agent Controller block for the *Lifecycle management* (initialization and termination phases). The *Landscaper* functionality also uses the same interfaces to build its historical view of services and resource usage.

The functionalities exposed in the *Service Orchestration* interface need to provide access to the *SLA Management*, the *Landscaper* and the *Recommender* so that other agents can perform their *Lifecycle Management* functionality.

The orchestration and deployment of new services and applications into the mF2C framework shall be a seamless and transparent process, whereby users are abstracted from the underlying resource diversity through a multi-cluster (a set of agents) interface. There are already some existing tools that fit into this scenario, providing a single point of access to a multi-cloud environment, like SlipStream [16].

As an illustrative example using the Slipstream solution, described in the Baseline Subsystems section, Figure 14 tentatively shows how cloud orchestration and management solutions can be extended to bridge standard cloud or data centre environment, with fog or edge computing environment.

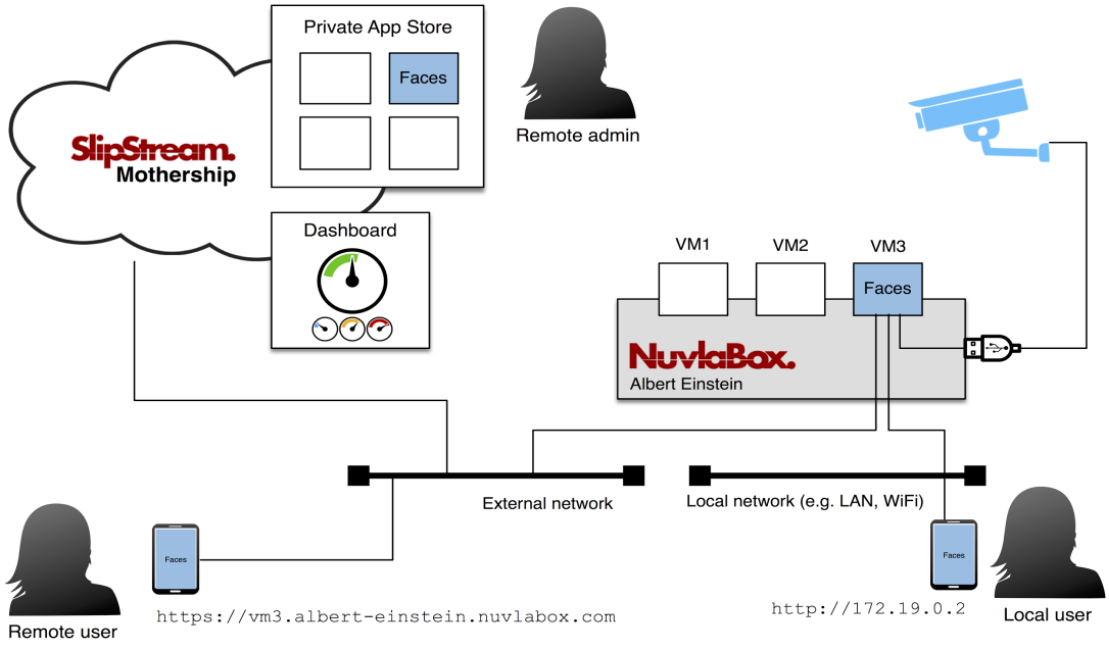

Figure 14: Using SlipStream to interface cloud based and edge based resources

Distributed Execution Runtime

This component will use the *Resource Management* interfaces in the Agent Controller to obtain information from the resources where the tasks are to be executed.

It also offers an interface for the *Service Orchestration* component in the same or another agent, which will be used in the operation phase of the service lifecycle. This interface will provide functionalities to trigger the execution of tasks.

The *Distributed Execution Runtime* also needs to interact with the same component in other agents. For this reason, this component needs to expose functions to obtain the location of data, to reallocate, and to replicate data for the scheduling functionality. On the other hand, it needs to provide a means to get information on the finished tasks, and eventually get their results.

Telemetry/Monitoring

This component needs to communicate with the *Service Orchestration* to obtain landscape information to perform its *Analytics* functionality. This functionality also requires including the possibility to update landscape information in the *Service Orchestration* interface.

This block also needs to provide an interface that will be used by the *Service Orchestration*, which will allow feeding the *Recommender* with the results of the *Analytics* functionality to feed the *Lifecycle Management*.

3.2.4. mF2C Client including Trigger

The mF2C Client is the software suite that the end-user will download in the form of mobile application, desktop application, virtual machine, and install in order to join to an existing mF2C system or deploying a new one. The Client app does not contain the required mF2C software to run, but it will download and install the agent instance. Once the agent is running on the device, the user will decide what set of resources to share with the system and what services requires. The Fig. 15 shows an example of this procedure.

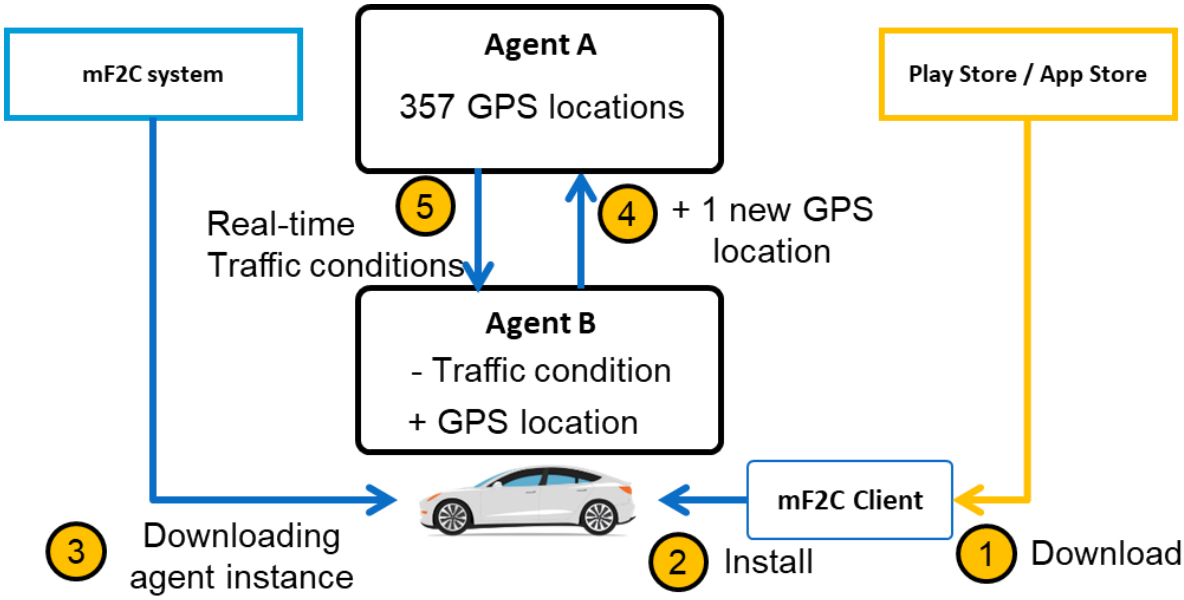

Figure 15: mF2C Client installation

In this example, the car, using Android Auto or Apple CarPlay, will download and install the mF2C client from the Play Store or the App Store. Then, the app will download from the mF2C system the agent instance necessary to deploy the service. In this example, the user will require for the current traffic conditions and in exchange will offer his actual GPS location to the system. The parent agent receives this request with the new location and provides service to the agent sending the real time traffic conditions.

3.3 Baseline Subsystems

This section introduces all the subsystems that will be used in every layer for the development of the mF2C project.

3.3.1. Cloud based

One of the tools that will serve as a baseline subsystem for the mF2C project is the ATOS SLA Framework.

Name	Description
SLA Framework	The SLA Framework is an implementation of an SLA system, compliant with WS-Agreement.

SLA Framework

The SLA Framework is a tool responsible for generating and storing the formal documents describing electronic agreements (WS-Agreements) between the involved parties: customers, application providers, cloud providers, etc. The resulting SLAs describe the service that is delivered, the functional and non-functional properties of the resource, and the duties of each party involved. At runtime, this tool is also in charge of supervising that all the agreements are respected.

The ATOS SLA tool is focused on the enforcement of business oriented policies (QoB: Quality of Business), which represent a constraint on a metric that impact on the business of the application, and the business actions to apply in case of violation. To fulfil this task, the SLA Framework relies on external monitoring tools which should assess the QoS. As a result of the enforcement process, the component stores the produced QoS and QoB violations and penalties, maintains the fulfilment state of each agreement and notifies the results to other components.

Figure 16 depicts the current SLA Framework architecture:

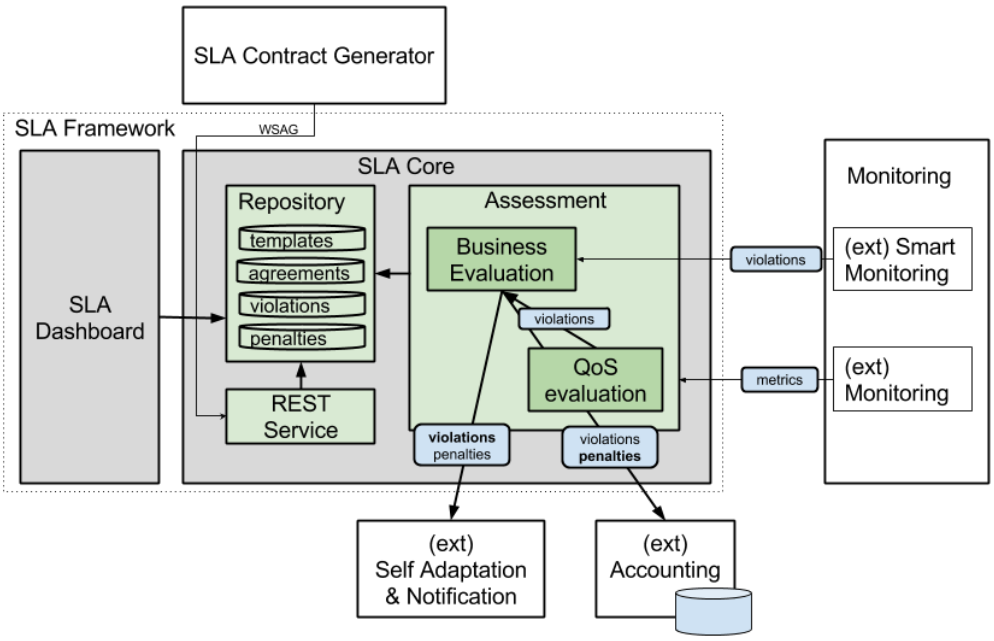

Figure 16: SLA Framework architecture

These are the main features of the SLA Framework tool:

- *WS-Agreement compliance:* The tool follows, in its implementation, the concepts and XML structure for agreements and templates that are defined in the WS-Agreement specification. In this specification, a template describes a service offer, and an agreement describes an instantiation of a template associated to a particular customer. According to WS-Agreement:
  - A template is a document used by the service provider to advertise the types of offers it is willing to accept.
  - An agreement defines a dynamically-established and dynamically-managed relationship between a provider and a customer, where the object of this relationship is the delivery of a service by the provider to the customer.
  - WS-Agreement structure:

  - **Context**: information about agreement parties, the agreement's lifetime, and the template reference
  - **Service Description Terms**: functional description of the service to provide (domain-specific description)
  - **Service References**: refer to existing services (domain-specific description)
  - **Service Properties**: define variables in the context of an agreement.
  - **Guarantee Terms**: define how guarantees are assessed and which compensation methods apply in case of a meeting or violating the service guarantees
    - Service Level Objective (QoS)
    - Penalties, Rewards
- XML and JSON representation.
- *One shot negotiation:* A simple negotiation, where a service provider automatically accepts a customer offer, is implemented. The development of a real negotiation process is outside the scope of the SLA tool.
- *REST interface:* The SLA tool exposes the basic functionalities of handling providers, templates and agreements, and searching of violations and penalties, through a REST interface. It also offers the possibility to push events, such as violations and penalties, to subscribed components.
- *QoS & Business Values enforcement:* The tool enforces the QoB rules defined in the agreements. The QoB rules rely on QoS rules to create complex metrics focused on business accounting.
- Notifications to external parties.
- Configurable and extendable: Monitoring adapter, document parsers, etc.

Apart from the SLA Framework, some of the resulting components of the EU-FP7 project ASCETiC [17], which investigated the optimization of designing, developing, deploying, and running software in Clouds with respect to energy efficiency and minimise the carbon footprint, will also be used as baseline subsystems within this project. These components are the Application Manager and the Application Monitor:

Name	Description
Application Manager	This application manages the lifecycle of an application deployed over diverse IaaS providers.
Application Monitor	The Application Monitor collects, stores and processes a flexible data set of service-level metrics.

Application Manager

The Application Manager component manages the lifecycle of applications deployed over diverse IaaS providers. These applications are sets of interconnected virtual machines, which are deployed according to a set of defined deployment requirements. This tool offers the following features:

- *Composite appliances:* Manage virtual appliances formed by a set of Cloud Resources that are interconnected between them.
- *Application Lifecycle.* It manages and coordinates the complete life cycle of an application: deployment, operation and undeployment; according to performance and energy constraints and goals at different levels: provider, application, VM.
- Support Elasticity.
- *Graphical and REST Interface:* This tool offers both a REST and GUI interface
- *Vertical Elasticity:* It supports vertical elasticity of VMs as self-adaptation action.
- *Multi-provider reallocation:* It supports moving partial or total application deployment to a different IaaS provider.

Application Monitor

The Application Monitor collects, stores and processes a flexible data set of service-level metrics. It also allows aggregating and retrieving them using a simple query language. Finally, it provides a GUI for the metrics visualization.

The application monitor is the primary source for information about the performance of the deployed application. The SLA tool could connect to this tool, in order to utilize this to indicate when SLA conformance isn't being met. This enables it to notify any SLA violations.

3.3.2. Fog based

SlipStream [18] is a multi-cloud application management platform and brokerage system, available under both open source (community) and proprietary (enterprise) versions. The solution provides a complete environment for capturing, deploying and managing any applications, in any cloud. High-level features provided by SlipStream's SaaS deployment, Nuvla, through a rich web UI, include a single Dashboard, AppStore, workspace for private development, unified usage monitoring and advanced auto-scaling.

This solution has been extended to manage edge computing appliances, such as the NuvlaBox, to build hybrid environments for distributed applications, creating a micro-cloud appliance for reliable computing.

The brokerage, the ability to communicate with several infrastructures in a hybrid environment, is facilitated by SlipStream's connector architecture. A SlipStream connector is a programmatic bridge that translates user requests into the destination infrastructure's native API. At the moment, SlipStream already supports a wide range of IaaS providers and due to its highly versatile connector architecture; new connectors can be easily created and plugged in. This is a clear added value for the mF2C framework where there is the need for having a generic interface to interact with the mF2C agents within the control plane, which in practice will consist of different kinds of devices with various specifications, for which different types of agent controllers will have to be implemented. Alongside with each one of these agent controller types, a SlipStream connector will also be created, easing the horizontal and vertical interaction within and among the different framework layers. Since mF2C agents can also be equipped with a platform manager instance, running SlipStream, the implementation of a new connector that exposes other instances of SlipStream will be exploited, allowing for a hierarchical and distributed management of the applications running in the infrastructure.

Another type of SlipStream connectors that is currently being developed addresses the orchestration of container based applications in existing Container as a Service (CaaS) infrastructures. As an example, this connector will enable the orchestration and deployment of containers in less powerful and smaller mF2C devices (like NuvlaBox Nano) which might not be suitable for hosting a virtualized environment but can run a CaaS cluster (based on mainstream container orchestration engines like Kubernetes, Docker Swarm, etc.).

For advanced usage, SlipStream also provides a REST API [18] and a command line interface [19] where some extra functionalities (like application scaling) are allowed. Each and every feature available via the web client or the command-line client is available via the REST API. SlipStream is also converging towards the DMTF/CIMI standard for the API, to deliver higher interoperability and compliance with the mF2C Agent Controller interface.

As a fog based baseline subsystem also the NuvlaBox [11] smart-clouds shall be included. NuvlaBox is SixSq's smart cloud-in-a-box appliance that delivers a simple and secure private cloud solution packed in a small standalone device. It is a plug & play device with remote management and application update capabilities that can be easily deployed by a non-technical operator. Once powered-up, the box can create its own local Wi-Fi network for neighbouring devices to use, while performing the work of multiple physical servers through its virtualization capabilities, handling several VMs running in parallel.

The hardware platform is a branded version of the fanless design by Logic Supply, certified for operations of up to 50° C and is dust resistant. Currently there are two NuvlaBox form factors: NuvlaBox v2 and NuvlaBox Mini. Due to its size, the latter suits better as a fog based device, with

4GB of RAM, 128 GB SSD, 4 USB ports, 60W external power supply, VGA/HDMI, one RJ-45 network interface, VESA holes, an Intel Celeron processor and the ability to run up to 4 VMs in parallel thanks to its OpenNebula based cloud layer and SlipStream, as shown in Figure 17.

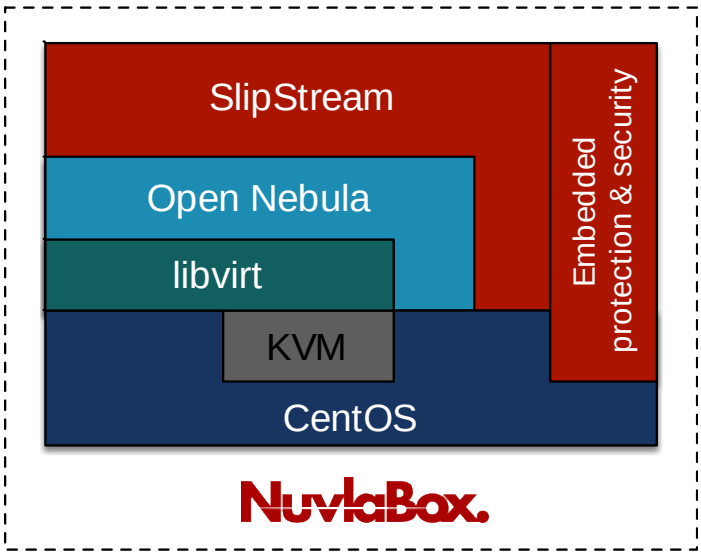

Figure 17: NuvlaBox - software specification

Currently in progress is also a third NuvlaBox form factor: the NuvlaBox Nano. This pocket sized NuvlaBox will be based on tiny computers like Raspberry Pi (or Intel Joule) and will offer a more flexible and fog-like solution at a much cheaper price tag and much lower power consumption. A possible software specification for the NuvlaBox Nano relies on container orchestration, having it working as a CaaS, running container applications.

3.3.3. Network management based

The proposed subsystem for Network management must be based on the latest advances, with the creation of complex applications using high level programming languages, to be executed on very different devices taking advantage of the abstraction of resources provided by the Operating Systems.

Software Defined Networking (SDN) and Network Function Virtualization (NFV) seem the perfect candidate, as SDN offers the decoupling of data and control planes in network devices enabling their independent development and a centralized view of the network. NFV proposes the migration from traditional network equipment, like firewalls and load balancers, to software packages that can be instantiated in a virtualized infrastructure. Both are complementary and could be integrated to provide an open network environment for developers.

OpenStack is proposed as the baseline of the software framework, as it is a very well-known and mature open source project to manage cloud platforms through a set of services, like Nova, Neutron, Keystone, Glance, Swift, Telemetry, etc., which can be integrated with SDN and NFV or other external sources.

OpenDaylight is a collaborative open source project, whose goal is to promote software-defined networking (SDN). OpenDaylight supports OpenFlow, the standard communication protocol in SDN networks. The software distribution includes an open controller, a virtual overlay network, protocol plug-ins and switch device enhancements.

Neutron is the OpenStack component focused on delivering networking-as-a-service (NaaS) in virtual computing environments. Neutron can be configured to work with OpenDaylight, delegating OpenDaylight to manage open vSwitch and provide SDN Controller capabilities.

The integration between Opendaylight and Openstack is based on the following:

- OpenDaylight exposes a single common OpenStack Service Northbound, where the API matches Neutron API, with multiple implementations of Neutron networks in OpenDaylight
- OpenDaylight Openstack Neutron plugin simply passes through, thus simplifying the Openstack plugin, and moving the complexity to OpenDaylight.

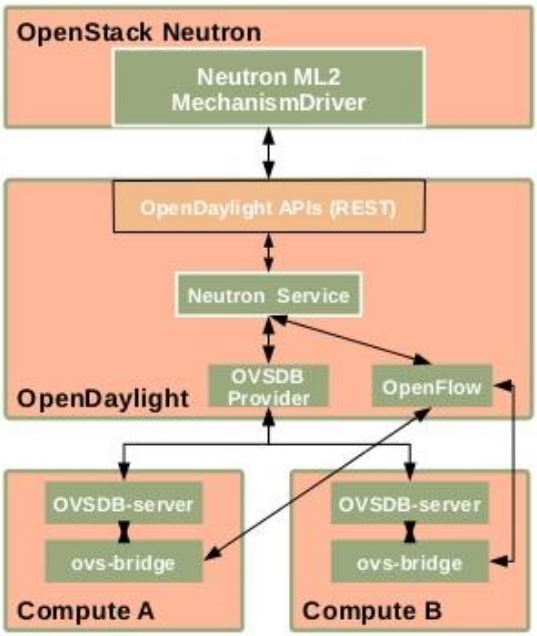

Figure 18: Neutron and OpenDaylight interaction

3.3.4. Distributed execution framework

COMP Superscalar (COMPSs) is a programming framework that aims to ease the development of parallel applications to run atop distributed infrastructures. For that purpose, it offers a sequential, infrastructure-agnostic way of programming that abstracts coders from the parallelization and distribution concerns. COMPSs considers applications as composites of invocations to pieces of software encapsulated as methods called Core Elements (CE). To manage the parallelism inherent in the application, the framework instruments the application and replaces CE invocations by calls to a runtime system to execute them atop the infrastructure.

The COMPSs runtime orchestrates the execution of CE invocations (tasks) fully exploiting the available computing resources (local devices or remote nodes) guaranteeing the sequential consistency. Applications share computing resources and, potentially, data values. To decide which resources host the execution of a task, the runtime is based on the concept of *Computing Platform*: a logical grouping of computing resources capable of running tasks. The decision is made on the *Decision Engine (DE)*, which is agnostic to the actual computing devices supporting the platform and the details to interact with them. The DE requests to each of the available platforms –configured by the user beforehand– a forecast of the expected end time, energy consumption and economic cost of the execution. According to a configurable heuristic, the DE picks the best platform to run the task and requests its execution; the selected platform is responsible for monitoring the data dependencies of the task and scheduling the execution of the task on its resources. Currently, there exist three different implementations of Computing Platform according to the nature of the computing devices composing it. CPU Platform manages the execution of tasks implemented as regular methods on the multiple cores of the device CPU. GPU Platform executes tasks implemented as OpenCL code on the embedded GPU. Finally, the third implementation, Remote Platform, offloads the execution of methods to remote resources. For the runtime to properly exploit Fog-Cloud environments, users can instantiate four platforms: a CPU Platform, a GPU Platform and two Remote Platforms: the Fog Platform encapsulating the low-latency remote resources (using West-

bound interface) and the Cloud Platform representing those VM instances deployed on Cloud Providers (using North-bound interface).

COMPSs is integrated with dataClay [20] in order to allow applications and services to access data transparently regardless of its location and of its format, thus hiding the complexities of data management. dataClay is a distributed object storage platform that allows applications to access data under a single data model independently of whether it is persistent or not, of where it is located, and of the technology used to store the data. This is achieved by means of object-orientation and encapsulation, that is, every manipulation of a data object is done through the invocation of a method associated to it, which is executed within the storage platform to avoid data transfers and hide the actual structure, state and location of the object.

dataClay enables sharing objects between different applications, with the possibility of offering a different set of methods to access and manipulate the data to each of them, according to the permissions each application has on each piece of data.

The integration of COMPSs and dataClay is done by means of an API offered by dataClay, which implements the set of functionalities required by COMPSs to get information about the locations of the objects, to invoke the execution of tasks, and to replicate or re-distribute objects in case it is required by the runtime for performance and scalability reasons.

3.4 Communication

The main objective of this subsection focuses on describing the communication flow between the different components of the system. To that end, we start by differentiating the control system communication among all agents prior to service execution, referred to as mF2C System Communication –described in subsection 2.3.1–, from the data flow communication produced during the service execution, referred to as Application Data Communication –described in subsection 2.3.2. A key difference between both communication flow schemes falls into the following assumptions we consider for the preliminary deployment strategy driving IT-1:

1. mF2C system communication will be hierarchical, i.e., the communication among all agents will follow a hierarchical topology.
2. Application data communication will be flat, i.e., the communication between the different agents (mF2C resources) will be flat, only limited by the real connectivity among them.

Fig. 9 depicts both topologies. Indeed, Fig. 9 a) presents an example putting together distinct devices forming part of the mF2C system, with red lines showing data communication among them. On the other hand, Fig. 9 b) shows the system (i.e., control) topology, organized into three hierarchical layers. The proposed hierarchical topology is configured once the devices are grouped, and the agents to act as leaders for each group (cluster) and layer, are all selected. It is worth mentioning that once the leaders' selection is done, despite the fact that all devices have the capacity to be leaders, only the selected one for each group and layer will act as leader. That said, this strategy is slightly tuned to make the whole architecture robust. Indeed, subsection 2.3.3 dealing with system resilience and fault tolerance, introduces the idea that for fault tolerant purposes, the grouped resources do not have only one leader, but (potentially) many. In that subsection, we start by assuming two potential leaders, where the second one, referred to as backup, may become leader in case of a leader failure. The policy to select the leader, the backup and how the roles swapping are done is yet to be defined.

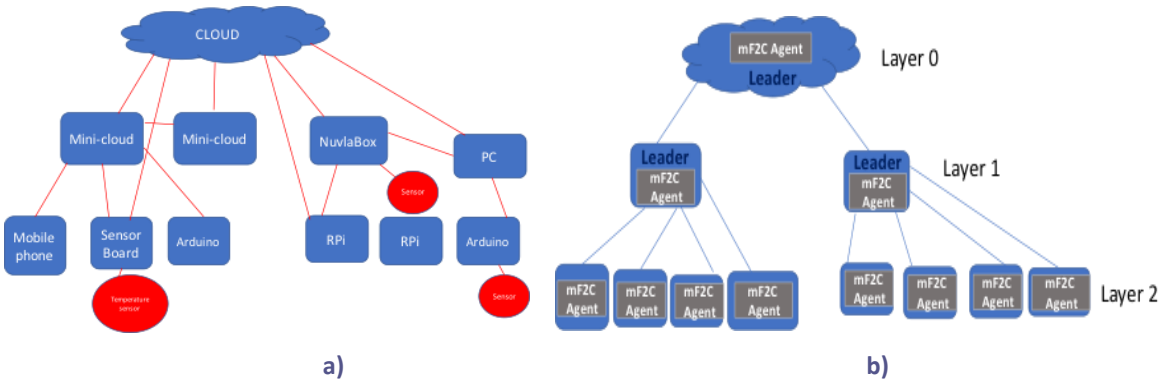

Figure 19: Communication: a) Flat topology, b) mF2C system topology

3.4.1. mF2C system communication

In the Fog-to-Cloud architecture the agents are instances of Remote Platforms as described in the previous section.

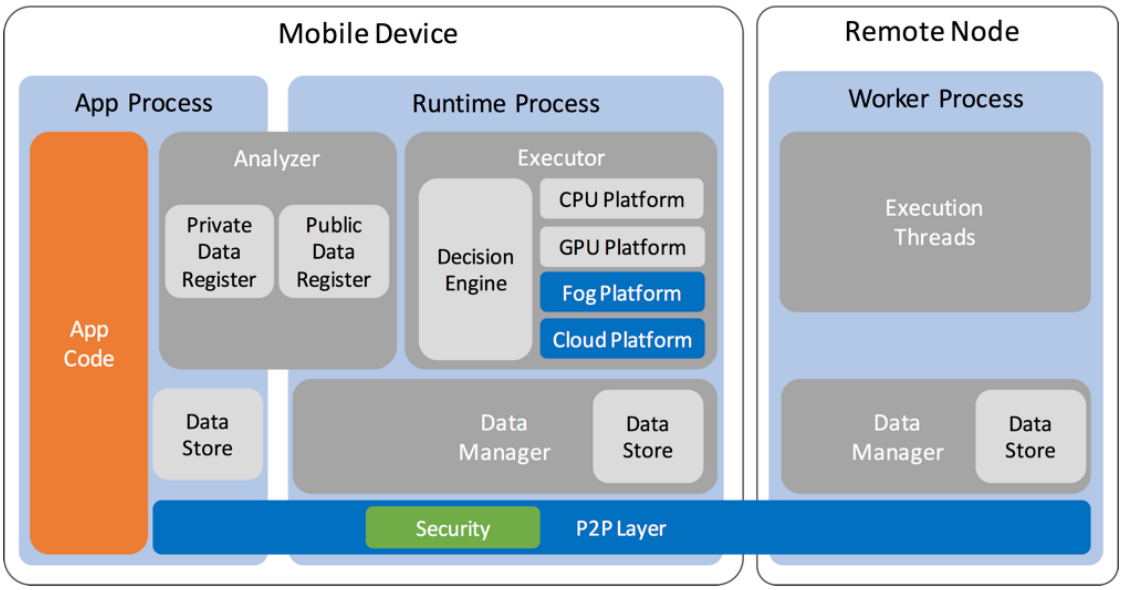

Figure 20: Distributed execution architecture

To release the device (a mobile for example) from the computational load of orchestrating the remote resources, Remote Platforms are organized as a peer-to-peer network. Each node of the network runs an execution process persistently listening to the network for task submissions; these processes are able to autonomously handle the execution of the task on the local computing devices. To ease the management of data dependencies, the remote nodes subscribe for and publish information and values of the datums accessed by the tasks on the Data Manager component, whose content –either information or values– is consistently distributed across the whole infrastructure. The local instance of the Data Manager is responsible for fetching the value from any hosting remote node.

An important feature of the communication between the leading agent and the others is the tolerance to network disruption typical of Fog devices. In the case of a leader running the application and that loses the connection with the rest of the infrastructure, the leader sends a message to every node upon the reconnection indicating its new address. Upon its reception, remote nodes update every reference to the leader node with the new IP and restart any interrupted transaction – transfer of a value or submission of internal master command. On long-lasting disruptions, worker nodes should keep progressing on the computation despite the isolation. In the case of reconnection, remote nodes autonomy reduces the impact of the network failure on the

performance of the application. Upon the broadcast reconnection notification, Data Manager instances synchronize their content, thereby all the components of the infrastructure become aware of the progress made by the other part. On the other end, the leader device should produce the expected result, even if the network connection is never re-established. Therefore, the leader may need to run all the pending tasks, even those already offloaded. Probably, some input values for a pending task are the output of an offloaded one and they are not likely to be on the master; hence, the value must be computed locally by running the producing task. This mechanism results in a backtracking process that only stops when all the input data required by a task exists in the device.

3.4.2. Application Data communication

As described above, the Application Data Communication model will be flat, meaning that all agents can communicate between each other, depending of course on the connectivity limitations among them. A representation of this flat model is shown in Figure 21, where the three layers are represented on the same plane and any agent is able to communicate with any other agent. The communication is represented by the blue lines and the black points represent the intersection point between the layers.

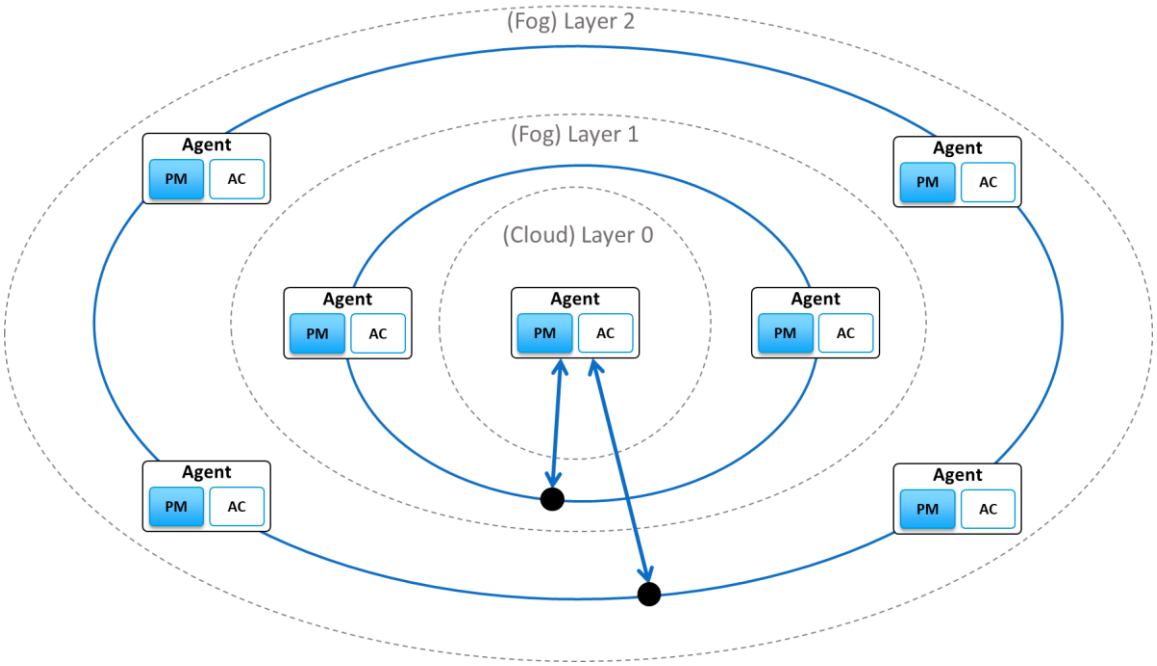

Figure 21: Flat data communication model

Application data is referred to as any kind of data that can be accessed and/or generated by a task (can be an application, a service or any other type of computational job) during its execution lifetime. To profit from the distributed model envisioned by the mF2C system, data will be distributed throughout multiple agents. This distributed data management approach arises not only as an obvious framework commodity but also as a requirement due to the intrinsic resource variability of the Fog devices. As an example, one can consider a smartphone as a mF2C agent which is completely capable of running an application, but that is also subject to external disruptions due to other applications sharing the same physical resources. Within the scope of data management, these external factors can highly affect the storage capabilities of that device and therefore justify the need for having a stage in and out processes that allow mF2C applications to both read and write data internally and externally to the agents.

Thanks to the Data Manager component referred above in 2.2.2.2, a data catalogue is broadcasted and made available to all agents in the infrastructure, hence allowing the agents in the lower layers to dynamically discover where data is located. It is the Data Manager's job to make sure the

broadcasted metadata contains not only the information about data location (or locations if data replications mechanisms are put in place) but also about the Access Control List (users' rights and permissions) and data retention and resilience policies applied on each data object.

Such a distributed data management model implies the existence of a common mF2C data access protocol that generalizes the data communication within the overall infrastructure. This uniform interface is provided by a distributed object storage platform that offers a global view of all the data in the infrastructure, and abstracts away physical details on how and where the data is stored. Each data object, be it a single object or a collection of objects, has an identifier that uniquely identifies it within the platform. This identifier allows looking-up in the catalogue for an object's metadata, such as its current location, or accessing it to perform some action, wherever it is located. In this way, tasks running on the mF2C infrastructure do not have to explicitly deal with the physical location of the data or the heterogeneity of the underlying resources.

Thus, when expecting a data object or a dataset (set of data objects), applications can make use of their agent's data catalogue to discover the closest (to reduce the round-trip time and network saturation) storage element containing the desired content, independently of whether the closest is its neighbouring worker, a leading agent or even a cloud provider. If the user/worker running the application has sufficient privilege for the requested data (according to the data catalogue) and it can be trusted (via authentication) with the storage element(s) hosting the data object(s), then data communication can occur. A clear limitation of this remote data communication functionality is the availability and reliability of the network. If agents lose connectivity to each other, then data communication is compromised. In this scenario, the application should try to make use of its agent's local storage as much as possible while waiting for the system's communication to be re-established.

3.4.3. Resilience/security/fault tolerance

While generally desirable features, resilience and fault tolerance are also security features, as they help ensuring *availability* and *reliability* of services (denial of service attacks were extensively discussed in D2.4 [21]). Programming for resilience can also help service availability, e.g. during upgrades. (We use "program" here as a generic term, covering software engineering for mF2C components as well as applications and agents.) Building in fail-over (e.g. redundancies) and fail-safe (e.g. self-checking) functionalities in components could ensure reliability or continuity of services.

In terms of communication, we need resilience in *programs* in order to ensure that they are available to send/receive data; we also need resilience in *data* to ensure that it is available (there is little point in securely communicating a message if it is compromised when it is stored); and obviously we need resilience features in the communications protocols themselves.

Programs (libraries, components, agents, applications)

Arguably there are commonalities between secure and fault tolerant programming, such as:

- Programs shall validate input, both from the network and their own operating environment. Communications should always be treated with some degree of suspicion; for example a legitimate and honest client may accidentally send a message which is too large, and crash the server.
- Always ensure the system is in a consistent state. The Apache (and to some extent Tomcat) web servers are good examples of mature servers which are good at handling high availability communications, by virtue of a thread pool or forking on handling incoming requests.
- Provide structured logging messages to facilitate fault detection and diagnosis. Section 2.2.3.3 gives more details on ensuring quality of service through monitoring.

- Check error codes from the calls and handle them appropriately - for example, a DNS lookup that has been successful all the way through testing suddenly fails, leaving the communications subsystem hanging or behaving strangely.
- Ensure that control is in a known state, e.g. avoid blocking on "slow" system calls.
- Quotas, limits, virtualization, groups, etc. can help constrain programs and guard against abnormal states; some operating systems and microcontrollers have watchdog features.
- Aim to run processes - particularly external ones - with the least privilege required.
- Simplicity - simple and well-commented code tends to be easier to secure, check, and make resilient.
- Built-in fault tolerance logic for mission-critical components. For example, it may make sense to have specific components that cache data and retry sending data when connections reappear.

Given the diverse nature of the program elements, there will be advice and best practices for specific operating systems and microcontrollers, and to specific programming languages.

Data

"Data resilience" can include different aspects. Measures to deal with these, unsurprisingly, can offer different levels of protection; for example, protecting against accidental damage is generally easier than protecting against a malicious adversary.

- Immediate availability - ensuring that the data remains available at any given time. This is typically solved by having multiple copies, and perhaps an automatic system to resolve and fetch the "nearest" copy.
- Long term availability - ensuring that data is archived and backed up, so it can be recovered in case of a disaster.
- Durability - which the data doesn't change over time (at least not unintentionally). Combining backups with integrity checking is a simple way of implementing resilience.

Communication protocols

Resilience of communication means that the communications protocols or networks offer features to help ensure message delivery or to determine whether delivery has failed. For comparison, email is sent over the Internet over TCP packets which include a sequence number and each packet is acknowledged by the receiver. The sender will retry sending a packet if it does not get the acknowledgement and the server will discover duplicate packets via the sequence number and discard them. Moreover, if the receiving Mail Transfer Agent (MTA) is unavailable or sends a "temporarily unavailable" message, the sending MTA will typically keep the message in its queue and try sending it again later, until it either gets a "success" message through the SMTP protocol, or a permanent failure message, or it fails to deliver the message for a set period of time. Note also that the sending MTA will generally notify the person who sent the mail of the delivery failure. However, the resilience comes at a cost. The default TCP mechanism takes time to recover from failure, particularly when ramping up high performance transfers, and queueing and retrying messages also requires resources.

In the IoT world, MQTT offers message delivery: the sender can choose, for each message, whether the message needs to be delivered at most once (so sent, and hoping for the best), at least once (so err on the side of retry), or precisely once (requiring a more complex send/acknowledgement pattern).

Conversely, on private low-noise networks, packet/message loss is unlikely, in which case high performance transfers with sufficiently high communication resilience can be achieved with UDP.

Also, stateless services are easier to replicate and thus make more resilient.

More generally, mF2C could use data and security policies to specify an appropriate level of resilience, such as backups, at-rest encryption, and qualities of service for message delivery etc., in the three aspects discussed above to help deliver communication resilience.

4. Functions of multiple module

This section describes key platform functionalities that cross-cut individual mF2C components. These requirements principally concern non-functional aspect of the architecture such as security, resilience, and performance, which are quality measures that will help demonstrate the robustness and usability of mF2C to potential users. In the following subsections, we describe how the mF2C architecture could address them. Last but not least, we will also explore the collaboration model underpinning the philosophy of mF2C: the creation of ad-hoc computing capacities on a mutual, voluntary and on-demand basis, and outline the roles of mF2C components in delivering the model.

4.1 Fault Tolerance and Resilience

Due to the highly dynamic and volatile scenarios that mF2C need to support, fault tolerance and resilience have to be key features of the proposed architecture. Moreover, it must be part of the architecture design from the ground up and should not be treated as supplementary features.

Depending on the use case, resilience will generally be more crucial in order to maintain connectivity among the agents than in other cases. For instance, let's consider a scenario with three layers, such as cloud (top layer), laptops (middle layer) and smartphones (lower layer). In this case, the low mobility and high resilience of laptops makes fault tolerance a less important factor to consider. On the other hand, in a highly dynamic scenario, like cars in a highway using the mF2C platform connected to the cloud, the volatility and speed of the cars force the mF2C platform to take the resilience and fault tolerance as a key feature. Fig. 2 shows this scenario and the backup procedure in case of the leader failure.

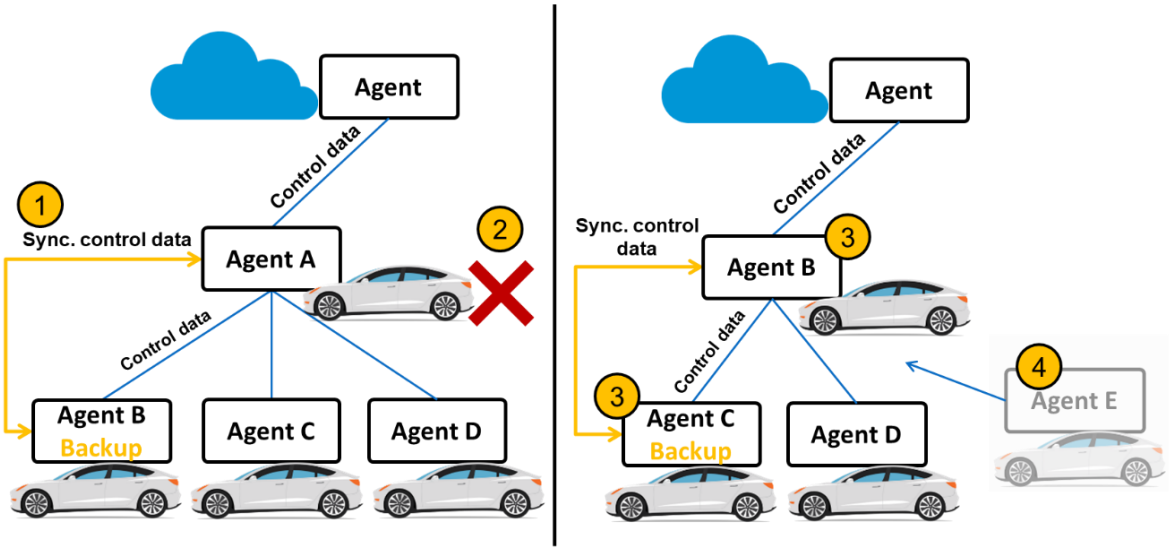

Figure 22: Fault Tolerance and Resilience in a Highly Dynamic Scenario

On the left side of the figure, we see three cars (Agent B, C and D) in the lowest layer connected to a car in the middle layer (Agent A). Agent A is the leading agent that has knowledge of Agents B, C and D and is working as a leader. However, this knowledge is not confined to Agent A, but is also replicated in Agent B who is working as a backup. Any modification on the control data in Agent A will be notified to Agent B. It should be noted that this knowledge is not part of the control that Agent A is applying to Agent B, but is completely independent.

Following with this scenario, if the car representing Agent A disappears (step 2), Agent B will automatically take over as the leader agent in the middle layer and assume control of the remaining agents (step 3) and a new agent will become the backup agent. In this case, Agent B will start the synchronization process sharing all its knowledge with Agent C. If new cars join the system (step 4), the agent (B) will do the registration and update the backup agent of the changes.

While this is just an example about how the mF2C architecture could build-in resilience, we are not constraining the number of backup agents as it will depend on the specific application scenario. In the highly dynamic scenario presented above, more than one backup will be necessary; while in more static scenarios such as an aggregator of low-risk information, only one backup will be sufficient. Running a large number of backups will clearly provide more resilience but this will incur increased communication overhead as more agents will need to synchronize control data. Therefore, there is a performance trade-off between the number of agents working as backups and the level of resilience required. Consequently, the mF2C architecture should allow application owners to configure the number of backup agents required in line with their application scenario.

4.2 QoS and Performance

The mF2C control architecture employs a distributed and hierarchical control topology aiming at mapping the service requests into the most suitable resources for provisioning services with low delay and network load. Therefore, distinct control topologies must be assessed: regarding number of controllers, controller capacity, and number of control layers, in order to find the optimal (or near-optimal) control topology for achieving the expected QoS.

Furthermore, in the mF2C platform, QoS provisioning is not limited to the parameters employed in traditional networking. In addition to latency, jitter, bandwidth, among others, various features may also impact on QoS in the collaborative scenario envisioned by mF2C. Some of these are briefly described as follows:

- *Energy consumption*: high energy consumption may result in the rapid unavailability of resources dependent of battery.
- *Mobility*: in order to cope with various issues arising from edge devices mobility, strategies for QoS-aware hand-off, migration, and checkpoints must be employed.
- *Data quality*: the employment of highly distributed and heterogeneous resources may affect data quality, which may include device location, data age, and device accuracy.
- *Security:* the resources sharing scenario envisioned by mF2C introduces new security concerns, including privacy, authentication, secure communication, and access controls, etc.
- *User profiling*: contracts between users and companies will yield distinct SLAs demanding distinct QoS provisioning.
- *Resource management*: besides the real-time management of traditional network resources, expected QoS provisioning in the mF2C platform requires the management of edge-resources by means of monitoring to avoid unavailability, as well as load balancing to avoid either over or under utilization.

These are only some of the parameters that may be employed for QoS measuring. However, it is worth mentioning that QoS requirements must be service-aware, in other words, distinct applications shall employ distinct sets of parameters as well as distinct weights for each one.

4.3 Security and Privacy

Given that mF2C provides PaaS for hosting client applications which may have differing security and privacy needs, a one-size fit all solution is not realistic, as in the case of ensuing communication resilience (Section 2.4.3) and QoS (Section 3.2). Thus, the mF2C architecture needs to support a flexible approach through the implementation of security policy (see D3.1 [22]) and it follows that:

- There must be an (or a few) authority service(s) which can grant limited purpose identity tokens to services (e.g. restricted to IP address or domain name, like X.509 certificates that can be used only as a server endpoint.)
    - It may make sense to have authorisation tokens with limited scope for clients as well.

    - It may need to provide a token translation service for different credentials, dependent on the policy on acceptable credential types
- There must be a mechanism to classify and tag data, services and components according to the level of security required/offered and a common vocabulary to encapsulate the information
- It must be possible to obtain secure storage/database services for agents in the fog/cloud
- A complete suite of security tools is deployed. Where this is not possible, e.g. at an edge device, the implementation of the security policy must be protected by some other means.

As in the case of QoS, it should be possible for end-users to specify parameters in the security policy to constrain the behaviour of critical data privacy functionalities such as:

- GDPR: users can see what their data permissions are, can choose how to share data, and have the right to be forgotten – see D2.4 Annex 7 and D3.1.
- Enabling users to select whom to share their data with and for which purpose.
    - It follows that there must be discovery facilities for peers and services
- Inter-operation with non-EU privacy regimes to at least a minimal level.
    - It follows that devices and systems should have some awareness of geo-location. This is required for correct operation of privacy regulations.
- The tracking of the location of persistent data. This is required for GDPR audit, financial audit and compliance with Legal Discovery actions. See D2.4 for a discussion.

For policy enforcement, mF2C components need to have some awareness of the provenance of:

- Data
- Device
- Human actor.

This is required for defense against imposter attacks and man-in-the-middle attacks (see D2.4 [21]).

Other key security-related factors to consider include:

- Dynamism: Flux is assumed for mF2C at all levels and all locations. This could mean power-on/off of a device, the installation of a new application, migrating to a different network, subnet or judicial region which entails a change of privacy regulations etc. When the *status quo* changes, whether physically or logically, there is scope for someone with malicious intent to break into the system. To prevent this implies the tight integration of security components and this in turn implies centralized coordination of security responses across mF2C platform through centralized monitoring of breaches and implementation of security policy. Mobile systems might be best dealt with as a primary actor versus the communications cell it connects to being dealt with as the secondary actor. The significance of this is that the mobile system moves from cell to cell rather than a list of cells each receiving a temporary visit from one mobile system (also see next section for a discussion of the effects of devices mobility).
- Interoperation: mF2C should support gateways between dissimilar architectural components as this is where security can be applied most effectively.
- Cooperation: integration of security systems at the most inclusive level possible. This usually means within the Cloud.

Finally, taking into account the need to centralize and coordinate the enforcement and monitoring of security policy, we propose the mF2C security architecture as illustrated in Figure 23. An in-depth analysis of the design is given in D3.1 [22].

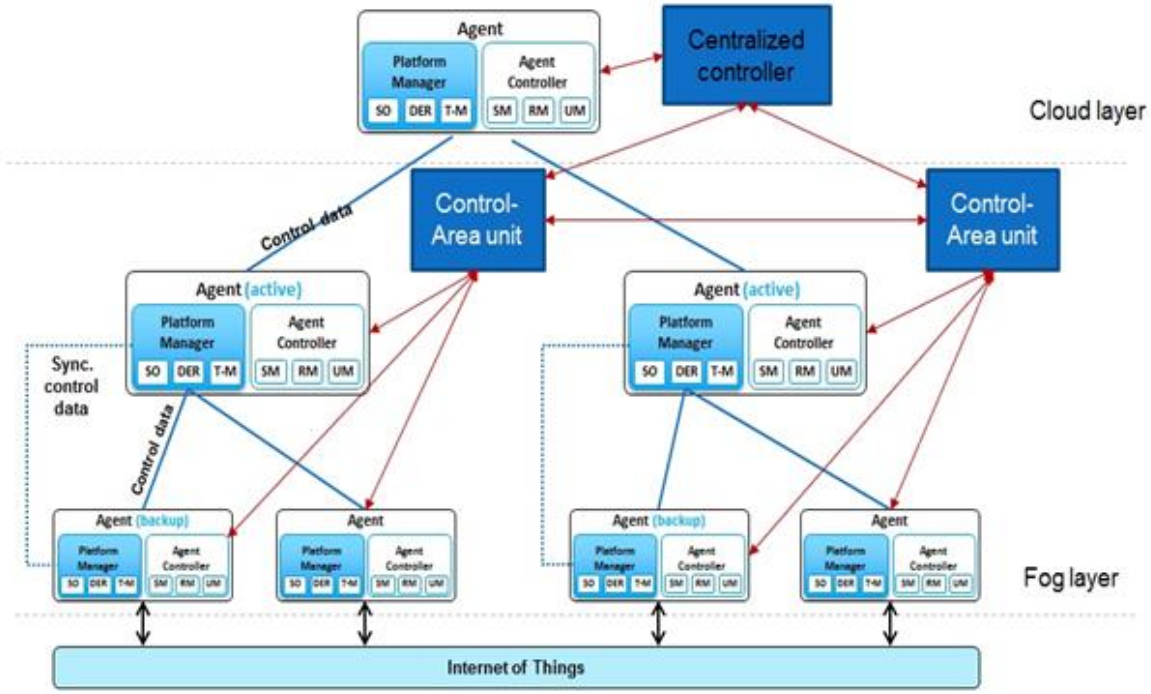

Figure 23: Preliminary security architecture

4.4 Collaborative model

This section preliminary explores the highly innovative collaborative model envisioned for mF2C, and outlines new sharing-based collaborative policies to be deployed to devices registered in an mF2C capable scenario. The main rationale for the proposed model leverages users' devices sharing and clustering policies to set-up on-demand, ad-hoc computing capacities to be offered to other users and services. While the envisioned collaborative model is definitely aligned to current economy models evolution (e.g. Airbnb, Uber, etc.), some technical as well as business-related challenges could hinder its future deployment. Both challenges, for instance, pose hard constraints on resources belonging to the lowest layer of the hierarchy (Layer 2), due to their limited capacity in terms of CPU, storage and even worse for mobile devices, when considering battery, switch on/off policies, etc.

Thus, we need a solution to overcome these potential concerns. On one hand, the envisioned collaborative model will require the deployment of new business strategies, providing users with incentives to encourage their willingness to participate in the collaborative model. To that end, novel QoS strategies and SLAs among collaborators and the mF2C system should be set. On the other hand, this collaborative model should consider the specific constraints linked to resources and services, in particular device mobility. The effects of mobility of devices will impact in particular on several functionalities of the mF2C solution, such as device discovery, identification or clustering, the latter is critical for the collaborative model envisioned: since the number of clustered devices will depend on their mobility pattern. In summary, the capacities to be provided by a group of devices participating in the collaborative model will depend on both the agreement in place between each user and the mF2C system. The agreement will highlight the resources that each user wants to share, based on incentives and/or other parameters, as well as the amount of connected devices per cluster depending on the mobility pattern.

To address the first issue, i.e., the capacities to be shared by user's devices, the project will propose policies similar to those proposed in volunteer computing [23]. Indeed, applying the volunteering concept, devices would voluntarily share (i.e., offer) its resources to the mF2C system, explicitly

setting the fraction out of the total of available resources to be shared with (i.e. offered to) the system. Beyond this static sharing policy, i.e. determining the static fraction of available resource (CPU, storage) the user wants to share, there is room to apply dynamic user's preferences [24], such as:

- Limits of the CPU usage. Only as an example, the user may limit the use of his/her resource:
    - When the resource is idle or used less than a percentage
    - Enabling only a limited number of mF2C services to run in his/her device
    - Enabling only specific mF2C services to run in his/her device
- Limits on the amount of battery. Only as an example, the user can limit the amount of resources shared, depending on the available amount of battery, which also can be related with the temperature reached out by the device.
- Limits the amount of bandwidth mF2C services are using, etc.

To address the second challenge, i.e. effects of mobility, new policies must be designed. As a preliminary attempt, such policies may benefit from the ideas proposed in Mobile Grid Computing (MDC). According to [25], a paper surveying MDC contributions, the more the mobile nodes added to a grid, the larger the monitoring overhead. This effect is mainly due to the fact that nodes on the move must be monitored more frequently. In addition to this overhead effect, other aspects must be especially addressed when considering nodes mobility, such as offloading policies, security, connectivity, variable bandwidth, energy-aware computing and quality of service, among others. Some of these aspects are addressed in [26].

After this brief summary highlighting main challenges coming up when considering resources sharing in the mF2C system, we identify which are the architecture blocks related to these challenges, mainly focused on the resource management, service management and user management in the agent controller:

- Resource management
    - Discovery
    - Policies
    - Categorization
    - Monitoring
- User management
    - SLA
    - Profiling
    - Sharing model
- Service management
    - Categorization
    - Allocation

4.5 Summary

mF2C proposes on-demand ad-hoc computing facilities offered to other users and services through a collaborative model (Section 3.4). For the least powerful devices (e.g. sensors, Raspberry PI etc.), unrestricted sharing could soon overload their capability. As a safeguard, the concept of volunteer computing requires that the volunteered system publishes a contract specifying what and how it is prepared to share as well as on what conditions. This idea of applying contracts to constrain mF2C system behaviour forms a recurring theme. Section 3.1 introduced the concept of allowing applications to improve resilience through configuring more back up agents but which would

negatively impact on performance. In Section 3.2, SLA is the vehicle for defining QoS parameters and performance goals. At runtime, the mF2C control architecture maps the service requests onto the most suitable resources as per the SLA requirement for network, storage, CPU capacity or security etc.

Similarly, in Section 3.3, we proposed a centralized and coordinated approach to the enforcement and monitoring of security policy covering generic trust and data privacy. So far, we have only outlined the direction that the mF2C architecture should evolve to support the platform-wide challenges outlined in this Section. Our next task is to drill down into the specific functionalities required for implementing these contracts and managing their life-cycles, for example, the vocabulary, protocol for the contracts and the components needed to deliver the supporting functionalities such as logging and housekeeping, monitoring and enforcement of breaches etc.

5. Development environment

An objective for the development environment for mF2C user applications is that low-level details are hidden from the developer, thus making the programming of fog-cloud applications as simple as possible. To this end we will use the COMPSs programming model, since it prevents the programmer from explicitly dealing with distributed computing issues on the heterogeneous resources that conform the mF2C infrastructure. In this way, technical details such as communications, networking, resource management or data management are hidden from developers, who can just focus on the functionality they want to implement.

The COMPSs programming model focuses on simplicity and is based on the idea of programming sequential and executing distributed and in parallel. Sequential programming is much simpler than parallel/distributed programming, and enables the application developer to focus on the semantics of the problem. The parallelism in COMPSs is exploited at task level, and for this the COMPSs syntax provides a simple interface to identify tasks. Depending on the programming language (Python, Java or C/C++) tasks are identified by the programmers in a slightly different way, but basically the programmer need to identify those methods or functions of the application that should be tasks and indicate the directionality of their parameters. The directionality of the parameters can be “input” when a parameter is read; “output” when a parameter is written or “inout” when is read and written by the task. This information is used by the runtime to determine the data dependences. This syntax is completed with a small synchronization API, only needed for Python and C/C++ bindings.

The COMPSs development environment includes a graphical Integrated Development Environment (IDE) that helps on the deployment of applications on distributed computing platforms; a runtime monitor, which enables to track the execution of the applications; and a performance analysis toolset, able to analyse in detail the behaviour with post-mortem tracefiles.

Persistent data is integrated in the programming model following the same philosophy of making applications unaware of low level details.

The first step for application developers is to create the set of Java or Python classes that represent persistent data in their applications, and upload them to the platform. Then, in order to code the logic of the application, they first need to retrieve from the platform a set of classes representing the original ones, and use them as a regular library. These classes, called stubs, also include new methods that allow making objects persistent, or deleting persistent objects in the distributed platform. Stubs allow programming the applications without dealing with client-server communications or interactions with a database or a file system, abstracting applications from the required data transformations between the different data models and interfaces.

6. Conclusion

This deliverable describes the initial version of the mF2C architecture (aligned to iteration 1 of the mF2C project). The second version will be submitted later in year 2. The different mF2C components and interfaces, as well as different types of requirements imposed by the applications and services running in the mF2C framework analysed and defined in this document result in a proposed architectural solution.

Section 1 analyses the reasons for developing an extended distributed platform that will integrate cloud services, IoT and fog, with also providing a short overview of the current solutions. It identifies the main actors that will interact with the system, responsible for fulfilling requirements of mF2C system and service along with the main business and technical goals that drive the mF2C architecture. For the better understanding of the requirements imposed by the applications and services we used three use cases specified in the project proposal.

The global design of the system with a detailed description of each individual block of the system, as well as the required interfaces for the communication between blocks is shown in Section 2. The mF2C platform proposes a layered architecture where the resources are categorized according to a certain policy and the agent entity deploys the management functionalities in every component within the system. It also includes the communication subsection focused on describing the communication flow between the different components of the system.

Key platform functionalities such as security, resilience, and performance that cross-cut individual mF2C components and a way how the mF2C architecture could address them are described in Section 3. Finally, the Section 4 presents the development environment for mF2C user applications with a goal of making the programming of fog-cloud applications as simple as possible

References

[1] F. M. R. Z. J. &. A. S. Bonomi, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 13-16. DOI:10.1145/2342509.2342513*, 2012.

[2] L. M. &. R.-M. L. Vaquero, "Finding your Way in the Fog," in *ACM SIGCOMM Computer Communication Review, 44(5), 27-32. DOI:10.1145/2677046.2677052*, 2014.

[3] "Fog Computing, Ecosystem, Architecture and Applications," [Online]. Available: http://www.cisco.com/web/about/ac50/ac207/crc_new/university/RFP/rfp13078.html.

[4] O. Consortium, "Definition of Fog Computing," [Online]. Available: https://www.openfogconsortium.org/resources/#definition-of-fog-computing.

[5] P. V. a. Y. Simmhan, "Demiystifying Fog Computing: Characterizing Architectures, Applications and Abstractions," 2017, pp. 1-23.

[6] "How does fog computing diverge from edge computing?," [Online]. Available: https://readwrite.com/2016/08/05/fog-computing-different-edge-computing-pl1/.

[7] "Fog Computing vs. Edge Computing: What's the Difference?," [Online]. Available: https://foghorn-systems.com/fog-computing-vs-edge-computing/.

[8] [Online]. Available: https://www.iso.org/standard/60545.html.

[9] [Online]. Available: http://www.worldsensing.com/product/loadsensing/?/solutions/industrial/technology/loadsensing-en.html.

[10] [Online]. Available: www.sentinel.hr.

[11] [Online]. Available: http://media.sixsq.com/nuvlabox-product-description.

[12] A. G. X. M.-B. E. M.-T. a. J. G. V. B. Souza, "Towards a Fog-to-Cloud Control Topology for QoS-Aware End-to-End Communication," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, Vilanova i la Geltrú, 2017.

[13] D. P. A. a. G. Fedak, "The computational and storage potential of volunteer computing," in *CCGRID'06*, IEEE Computer Society, 2006, pp. 73-80.

[14] R. J. A. Bichawat, "A Survey in Mobile Grid Computing," in *Int. J. of Recent Trends in Engineering and Technology*, 2010, pp. Vol.4, No.2.

[15] D. P. A. J. M. V. D. Kondo, "Performance Evaluation of Scheduling Policies for Volunteer Computing," in *Third IEEE International Conference on e-Science and Grid Computing*, Bangalore,

] India, 2007.

[16 [Online]. Available: http://sixsq.com/products/slipstream/.
]

[17 [Online]. Available: http://www.ascetic-project.eu/.
]

[18 [Online]. Available: http://ssapi.sixsq.com/.
]

[19 [Online]. Available: http://ssdocs.sixsq.com/en/latest/.
]

[20 A. Q. D. G. A. B. J. C. T. C. J. Marti, "dataClay: a distributed data store for effective inter-player
] data sharing," *Journal of Systems and Software,* p. DOI:10.1016/j.jss.2017.05.080, 2017 (to appear).

[21 "mF2C D2.4 Security and Privacy Requirements and Features," [Online]. Available:
] http://www.mf2c-project.eu/d2-4-m4/.

[22 "mF2C D3.1 Security and Privacy Aspeects of the mF2C Agent Controller," [Online].
]

[23 E. R. C. o. t. I. o. Things, "EU-China Joint White Paper on Internet-of-Things Identification," 01
] Oct 2014.

[24 Y. L. a. R. Jain, "Naming in the Internet of Things," 20 Dec 2013.
]

[25 [Online]. Available: http://blogging.avnet.com/ts/advantage/2016705/why-the-internet-of-
] things-needs-fog-computing/#_ftn1.

[26 P. A. a. D. Vidyarthi, "Resource discovery in Mobile Cloud Computing: A clustering based
] approach," in *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)*, 2015, pp. 1-6.

[27 R. C. L. L. a. G. M. Z. Li, "Dynamic Resource Discovery Based on Preference and Movement
] Pattern Similarity for Large-Scale Social Internet of Things," *IEEE Internet of Things Journal,* Vols. 3, no.4, pp. 581-589, 2016.