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

D3.4 Design of the mF2C Agent Controller Block (IT-2)

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

Work Package	**WP3, mF2C Agent Controller block design and implementation**
Due Date:	*M30*
Submission Date:	*28/06/2019*
Version:	*1.3*
Status	*Final*
Author(s):	*Xavi Masip (UPC), Eva Marín (UPC), Shirley Crompton (STFC), Jens Jensen (STFC), Román Sosa (ATOS), Roi Sucasas (ATOS), María Teresa García González (ATOS), Cristóvão Cordeiro (SIXSQ), Francisco Carpio (TUBS), Jasenka Dizdarevic (TUBS), Sridhar Voorakra (INTEL), Shirley Crompton (STFC)*
Reviewer(s)	*Cristóvão Cordeiro (SIXSQ)* *John Kennedy (INTEL)*

Keywords
Design, agent controller, security, authentication, event system, GUI

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

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.0	29-04-2019	ToC	Eva Marín (UPC)
0.1	07-05-2019	1st version with UPC content	Eva Marín (UPC)
0.2	17-05-2019	Provided input for all the sections	Xavi Masip (UPC), Eva Marín (UPC), Shirley Crompton (STFC), Jens Jensen (STFC), Román Sosa (ATOS), Roi Sucasas (ATOS), Cristóvão Cordeiro (SIXSQ), Francisco Carpio (TUBS), Jasenka Dizdarevic (TUBS), Sridhar Voorakra (INTEL)
0.3	23-05-2019	1st integrated version	Xavi Masip (UPC), Eva Marín (UPC)
0.4	27-05-2019	Reviewed STFC sections and comments to Agent Startup workflows	Shirley Crompton (STFC)
0.5	5-6-2019	Inputs provided by INTEL and ATOS	Roi Sucasas Font and John M. Kennedy
0.6	6-6-2019	Adjusted a few minor issues by STFC and added executive summary	Shirley Crompton (STFC) and Xavi Masip (UPC)
0.7	8-6-2019	Added changes in workflow of Normal Agent Startup	Shirley Crompton (STFC) and Eva Marín (UPC)
0.8	17-6-2019	Version revised by Cristóvão Cordeiro and addressed comments in Introduction and section 3.	Cristóvão Cordeiro (SixSQ) and Xavi Masip (UPC)
0.9	20-6-2019	Version revised by John Kennedy	John Kennedy (INTEL)
1.0	25-06-2019	Inputs provided by ATOS, STFC and UPC	Eva Marín Tordera (UPC), Shirley Crompton (STFC) and Roi Sucasas Font (ATOS)
1.1	26-06-2019	Version ready for Quality check	Eva Marín Tordera (UPC)
1.2	27/06/2019	Quality check.	María Teresa García González (ATOS)
1.3	27/06/2019	Final version	Eva Marín Tordera (UPC)

Table of Contents

List of figures

List of tables

Executive Summary

This document has been developed by the mF2C project and is intended to clearly describe the design of the mF2C Agent Controller (AC) developed in iteration 2 of the project (IT-2). This is the final design of this specific functional block.

The main objective of this document is to provide a comprehensive understanding about the Agent Controller and its main functionalities. The document starts by summarizing the final mF2C architecture proposed in D2.7 [1], for IT-2 highlighting both the main differences with the preliminary approach presented in IT-1 (D2.6 [2]) and the updated set of blocks and functionalities proposed for the AC, expressed only in two blocks, namely, the Resource Management and the User Management. In addition, some of the subcomponents of these main blocks have been simplified (such as categorization), moved to the Platform Manager (such as QoS enforcement) or collapsed into another block (such as monitoring in categorization), etc. The main functionalities of the AC are illustrated through the updated (for IT-2) workflows, as the main foundation for the continuous development and later implementation. Furthermore, it is worth mentioning that all interface descriptions have been shifted to WP4, to be reported in D4.8.

Finally, in parallel with D4.4, aimed at describing the Platform Manager (PM) functionalities, some of the transversal blocks in the mF2C agent, specifically the Event Manager, the Security block and the Dashboard/GUI interface, are also described in this deliverable, while the APIs block and the Data Management block are described in deliverable D4.4.

The outcome of this document is a detailed final design of the Agent Controller, including the update to the approaches presented in IT-1 for the different functionalities (resources discovery and identification, policies, SLA, sharing model, etc.), including illustrative workflows that will also be essential for the next stages of the improved development and implementation of all the AC's blocks, taking as a starting point the design and implementation in IT-1, and considering the updated designs presented in this deliverable.

1. Introduction

1.1. Introduction

The final design for the mF2C architecture in IT-2 has been described in depth in D2.7 [1], leveraging the preliminary design for IT-1 reported in D2.6 [2]. Although the architectural view does not substantially change in key aspects (i.e., a layered and hierarchical architecture, the agent as the key deployment component, etc.), some improvements have been added to that preliminary design, some of them motivated by the mandatory process of updating the list of assumptions considered in the IT-1 design. In short, two main architectural improvements require specific attention, both related to the agent, as introduced next. On one hand, we assume the fact that the agent will not run on every device, meaning to say, in other words, that there will be devices willing to participate in the m2C system without sufficient capacity to support the agent needs. In order to sort out this problem, thus facilitating less powerful devices also joining an mF2C system, in IT-2 we proposed to deploy a light version of the agent, referred to as a microagent, certainly having a small set of the functionalities envisioned for the agent, although enough to make the device mF2C capable. The design of the microagent is described in detail in D4.4.

On the other hand, we consider a new order in the set of functionalities to be included in the agent. This effect motivates two changes in the agent components design. First, some functionalities moved from one component (Agent Controller) to another one (Platform Manager). Second, new components in the agent block have been designed.

The main high-level changes in the agent design for IT-2 regarding IT1 implementation are:

- The Service Management block included in IT-1 design within the Agent Controller has been shifted to the Platform Manager. The rationale behind this change is motivated by considering the Service Management tasks as global tasks not only related to the particular device.
- The QoS enforcement within the User Management block has been shifted to the Service Management now in the Platform Manager, since we decided to put all QoS related functionality into a single block.
- The Data Management in the Resource Management block has been shifted to a new external block referred to as Data Management,
- Three new blocks are added to the Agent, extending the two blocks, i.e. Agent Controller and Platform Manager, considered in IT-1 design. The main rationale justifying the need for the three newly-added blocks boils down to simplifying the design while also guaranteeing a correctly functional interface as well as transversal performance when need. The three blocks are:
    - Event Manager and GUI:
        - Event Manager: Responsible for guaranteeing all internal components to have the required information about events happening within the system.
        - GUI: Responsible for facilitating a correct system visualization at user level, through the added dashboard aimed at easing users and operator's management.
    - Security: Including all specific components to provide security by design to the mF2C system.
    - Data Management and APIs:
        - Data Management: Including dataClay as the database management system used in mF2C, CIMI as the interface used in the agent to manage information
        - APIs: the set of APIs to enable the deployment on different environments.

Assuming that some of the functionalities of the agent are neither within the Agent Controller nor within the Platform Manager, the project has decided to report the first two blocks (i.e. Event

manager, GUI and security) in D3.4 while keeping the one remaining (Data Management and APIs) in D4.4.

1.2. Purpose

The objective of this deliverable is to provide a final description of the different blocks and components as envisioned for the Agent Controller (AC) block in IT-2, including the architectural refinements of the design proposed in IT-1. This is the final Agent Controller design deliverable, so along with D4.4 must give a comprehensive description on what the complete agent is. The refinements of the global agent do not only affect the internal components of the Agent Controller, but also drove to add 5 new components in the agent, two of them to be reported in this deliverable. So, Section 2 starts describing the agent architecture for IT-2 emphasizing the changes and upgrades to the architecture proposed in IT-1. Then Section 3 describes the Resource Management block going deep into each one of its components, while Section 4 reports on the User Management block and its components. The new security block is described in Section 5, Section 6 presents the Event System and Section 7 the GUI & Dashboard.

As done in D3.3 [3], for the sake of better understanding about the operation of the different proposed blocks, this deliverable also includes some workflows describing the functionalities of the sub-blocks.

1.3. Glossary of Acronyms

Acronym	Definition
AC	Agent Controller
ALP	Automatic Leader Promotion
ALSP	Automatic Leader Selection Policy
API	Application Program Interface
CA	Certification Authority
CAU	Control Area Unit
CIMI	Cloud Infrastructure Management Interface
COMPSs	COMP Superscalar framework
CSR	Certificate Signing Request
CURL	Client URL Request Library
deviceID	unique device IDentifier
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
IDkey	unique user IDentifier
IdP	Identity Provider
IT-1	Iteration 1
IT-2	Iteration 2
JOSE4j	Javascript Object Signing and Encryption for Java
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JWE	JSON (JavaScript Object Notation) Web Encryption (RFC 7516)
JWS	JSON Web Signature (RFC 7515)
JWT	JSON Web Token (RFC 7519)
LC	Launching Control
LCM	Lifecycle Manager
LDR	Leader Discretionary Requirements
LEA	Leader Election Algorithm
LMR	Leader Mandatory Requirements
MAC	Media Access Control
OpenVPN	Open source VPN
PKI	Public Key Infrastructure

Acronym	Definition
PLP	Passive Leader Promotion
PLSP	Passive Leader Selection Policy
PM	Platform Manager
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request For Comments
SLA	Service Level Agreement
SSE	Server Side Event
UM	User Management
VPN	Virtual Private Network
VSIE	Vendor-Specific Information Element

Table 1. Acronyms

2. Summary of mF2C architecture for IT-2

The layered architectural design proposed for mF2C has been described in deliverable D2.7 [1] as well as the agent architecture design. Differently from IT-1, where the agent was only divided into two main blocks, the Platform Manager and the Agent Controller, for IT-2 some additional functionalities are also represented in the architecture as part of the agent description.

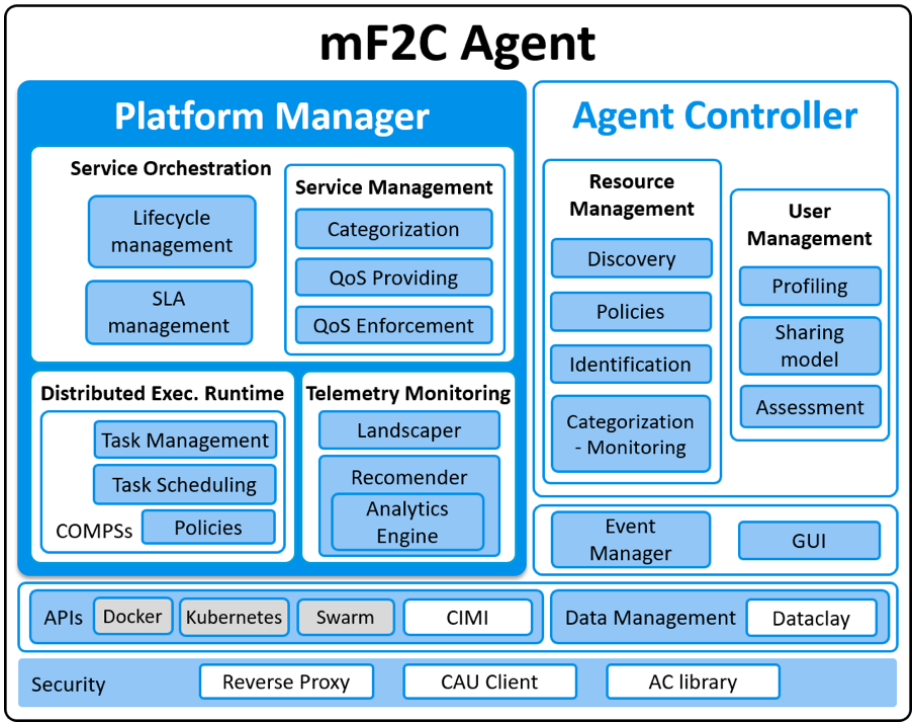

Figure 1. mF2C agent for IT-2.

In the Platform Manager, the Service Management block now is part of the Service Orchestration, while in IT-1 it was in the Agent controller. Telemetry Monitoring and Distributed Execution Runtime remain the same. On the other hand, the Agent Controller, whose functionalities will be explained in more detail in Subsection 2.1, now only has the Resource Management and User Management. The Data Management block is now an independent module used by both Platform Manager and Agent Controller. Some of the functionalities that have been previously part of either AC or PM during IT-1 have now been positioned as separate components in the new iteration [1]. These include Data Management, Security, Event Manager, GUI and an API, and can be seen in Figure 1.

In IT-1 some of the Data Management functionalities were assigned to the AC and others to the PM. Since the Data Management was required by all mF2C agent components for accessing data it made sense for this component to evolve to a transversal atomic component outside of PM and AC.

Another transversal component is Security, whose different security functionalities cross cut both the AC and PM. It includes the enhanced CAU client which serves as an Agent's local gateway to the CAU middleware, the Reverse Proxy and the refactored AC library. The three blocks working together offer security at different levels of the mF2C software stack:

- the Reverse Proxy vets requests from outside the Agent according to predefined security rules and re-directs the filtered traffic to the appropriate Agent blocks to help protect their endpoints
- the AC Library provides functionalities for securing messages in line with the mF2C data security policy (see D3.2, [4]) as well as the creation of signed identity JWT to facilitate Agent authentication, and

- the CAU client serves as an Agent's gateway to the loosely coupled CAU middleware which acts as the mF2C identity manager. The two together helps operate the PKI trust model within mF2C, e.g. obtaining X.509 certificates to uniquely identify Agents, leveraging the credentials tied to the certificate for Agent authentication, key exchanges, data security and message non-repudiation, etc.

Two additional transversal modules for the mF2C agent are Event Manager and GUI (Graphical User Interface). The Event Manager, which is introduced in IT-2, allows for any of the modules in the mF2C agent to subscribe to the events. The GUI or dashboard is an update of the previous dashboard in IT-1, which was only used for registration and launching services and located in the cloud. In IT-2 GUI is enriched with new functionalities, such as the monitoring of the mF2C system and is installed in every mF2C agent, not limited to the cloud.

Design of Security, Event Manager and GUI modules will be explained in more detail in Subsections 5, 6 and 7, respectively.

The remaining two transversal blocks (APIs and Data Management) are described in the D4.4 deliverable dedicated to the design of Platform Manager.

2.1. Survey of main Agent Controller Functionalities

This section updates for IT-2 the main components for the Agent Controller as proposed in deliverable D2.6 [2], designed in deliverable D3.3, [3] and integrated in deliverable D3.5, [5]. The set of functionalities for the Agent Controller was split into three main blocks, Resources, Services, and Users in the design for IT-1. Following the integration and validation in IT-1, the consortium decided to move the Service Block to the Platform manager module. As established in the first design of the architecture, the Agent Controller is in charge of local decisions, and all the smartness of the system is also located at the Platform Manager. Furthermore, control communication is always done among Platform Managers. For these reasons, and taking into account that the Service Management block includes QoS provisioning and QoS enforcement, that cannot be only handled locally - they need a global view of services and resources to be executed, the Service Management block will be in Platform Manager in the design for IT-2.

The next figure, Figure 2, shows the blocks of the Agent Controller for IT-2:

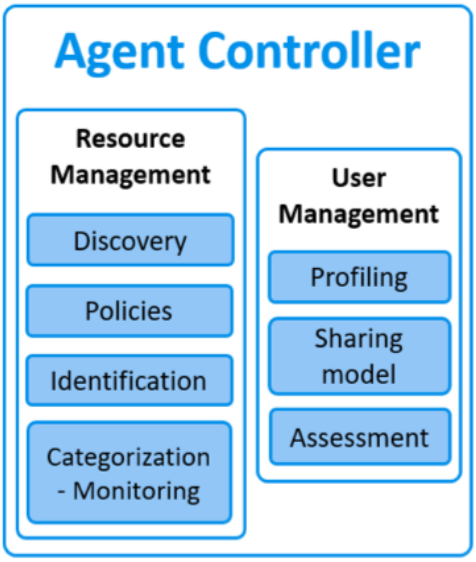

Figure 2. mF2C agent controller

For IT-2 the two main blocks of the Agent Controller are the Resource Management block and the User Management block. Regarding the resources, the main functionality of the Agent Controller is to synchronise information in the mF2C agent database, related to the resources, including the resource discovery, identification and categorization of them, and the policies to be applied.

The User Management block in the Agent Controller has all the functionalities related to the management of the profiles, Sharing model and assessment. It is worth mentioning that in IT-1 in this User Management block we had the QoS enforcement which has moved to the Service Management block in the Platform Manager, whereas we have added the Assessment module, that checks that an mF2C device does not share more resources than defined by the user.

3. Resource Management Design

3.1. Registration

The registration is the first logical step required to join to the mF2C network, as already explained in D3.3. The process is the same as it was implemented in IT-1, and such registration is hosted in the cloud of the service provider and requires the user to enter his/her email and to pick a username and a password. After a successful registration (see Figure 3), the user must validate the newly created account by following a link sent to their email.

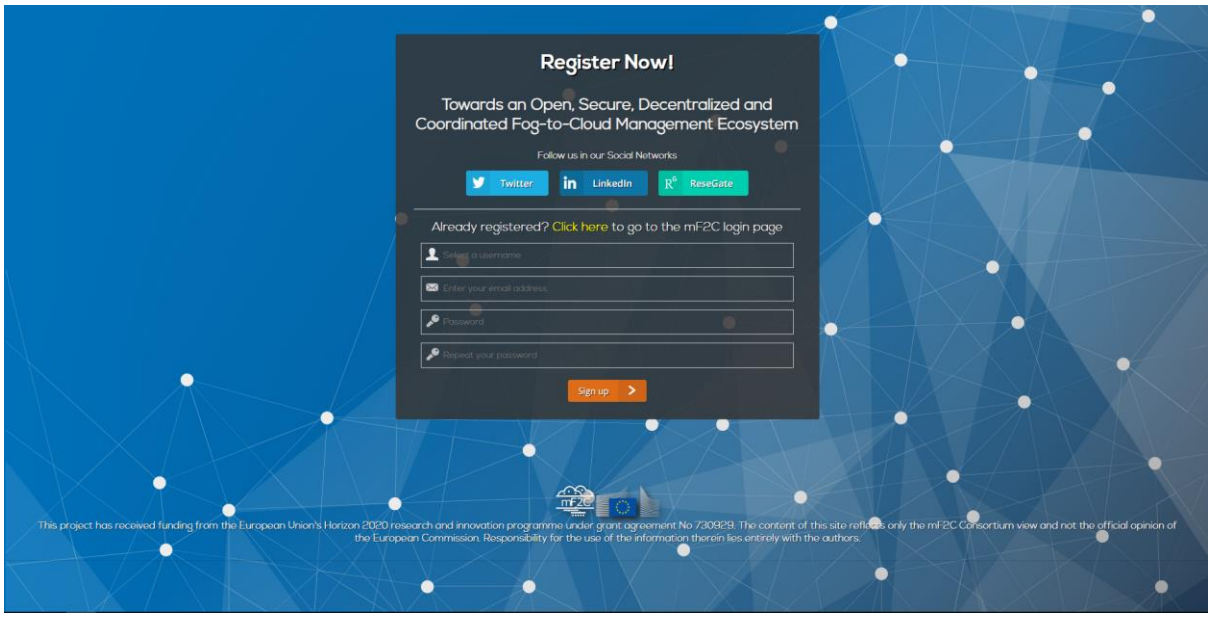

Figure 3. Registration site (Figure 7 in D2.7 [1]).

Once the account has been validated, the user will be able to login into the mF2C frontend and to download the agent appropriate for their device (see Figure 4). The registration process ends once the user is able to login into the system and thus, a unique IDKey (aka, user identifier) has been assigned to the newly created account.

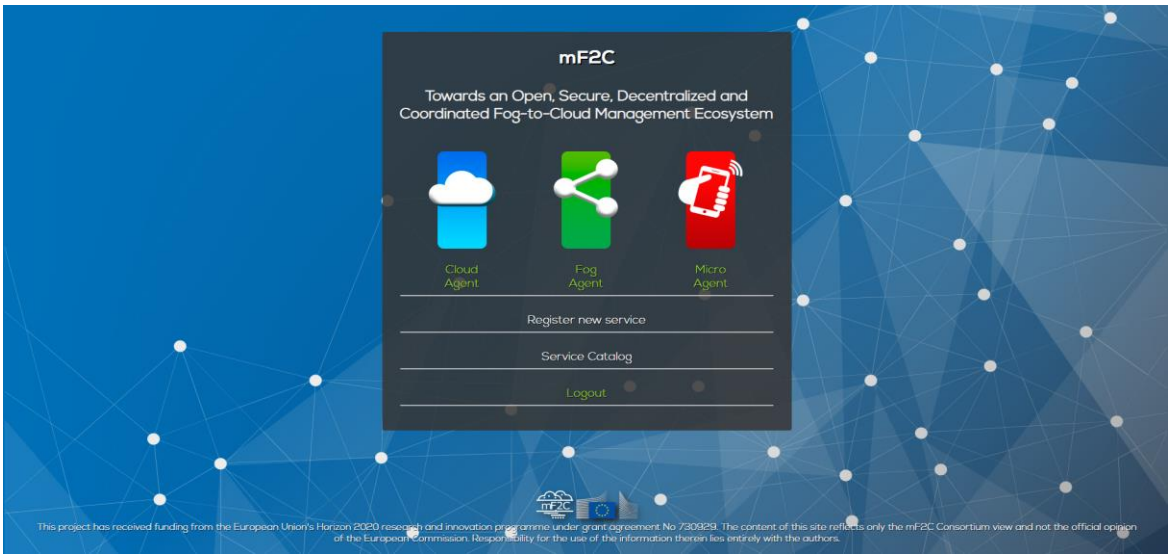

Figure 4. Download page updated for IT-2

3.2. Identification

In the mF2C network, the identification is an essential characteristic that allows to assign and manage unique identifiers for each device that is connected to the network. The mF2C system requires the devices to be properly identified in order to allow them to access to the available pool of resources (compute, storage, services, data, etc.)

Like the registration procedure, the identification process in IT-2 remains essentially the same as IT-1. Once the user executes the agent for the first time, the identification module connects to a WebService (WS) in charge of assigning a unique identifier to each device (deviceID).

In the case where the agent has been downloaded using the dashboard, the IDKey included in the docker-compose file is sent to the WS, otherwise (for example download source from GitHub without IDkey), authentication using the user credentials (username and password) is performed.

3.3. Discovery

The discovery mechanism in IT2 follows the same design presented in IT1. In a nutshell, leaders leverage their Wi-Fi interface to broadcast beacons containing mF2C-specific Vendor-Specific Information Elements (VSIEs). On the other hand, in order to detect the presence of nearby leaders, regular agents perform a Wi-Fi scan looking for beacons containing mF2C VSIEs.

3.4. Categorization

Compared to IT-1, in this iteration (IT-2) the resource categorization module has been modified in several ways. Firstly, in this second iteration, we added a new functionality for the resource-categorization module, Monitoring. Now the categorization module is not only categorizing the resources according to the captured resource information but also, by continuously collecting the resource information, this module is also monitoring the current resource availability. On the other hand, and to ease the designing and implementation process, we decided to reduce the previous resource categorization proposed in IT-1. So, for this IT-2, we consider the following categories:

- the device specification (i.e., Hardware_Specs, Software_Specs, Power_Info, Network_Info)
- information of attached IoTs (i.e., Sensor_Info, Actuator_Info)
- and the agent-type information (i.e., cloud, fog, and micro-agent).

In Figure 5, we depict the details of the existing categorization module.

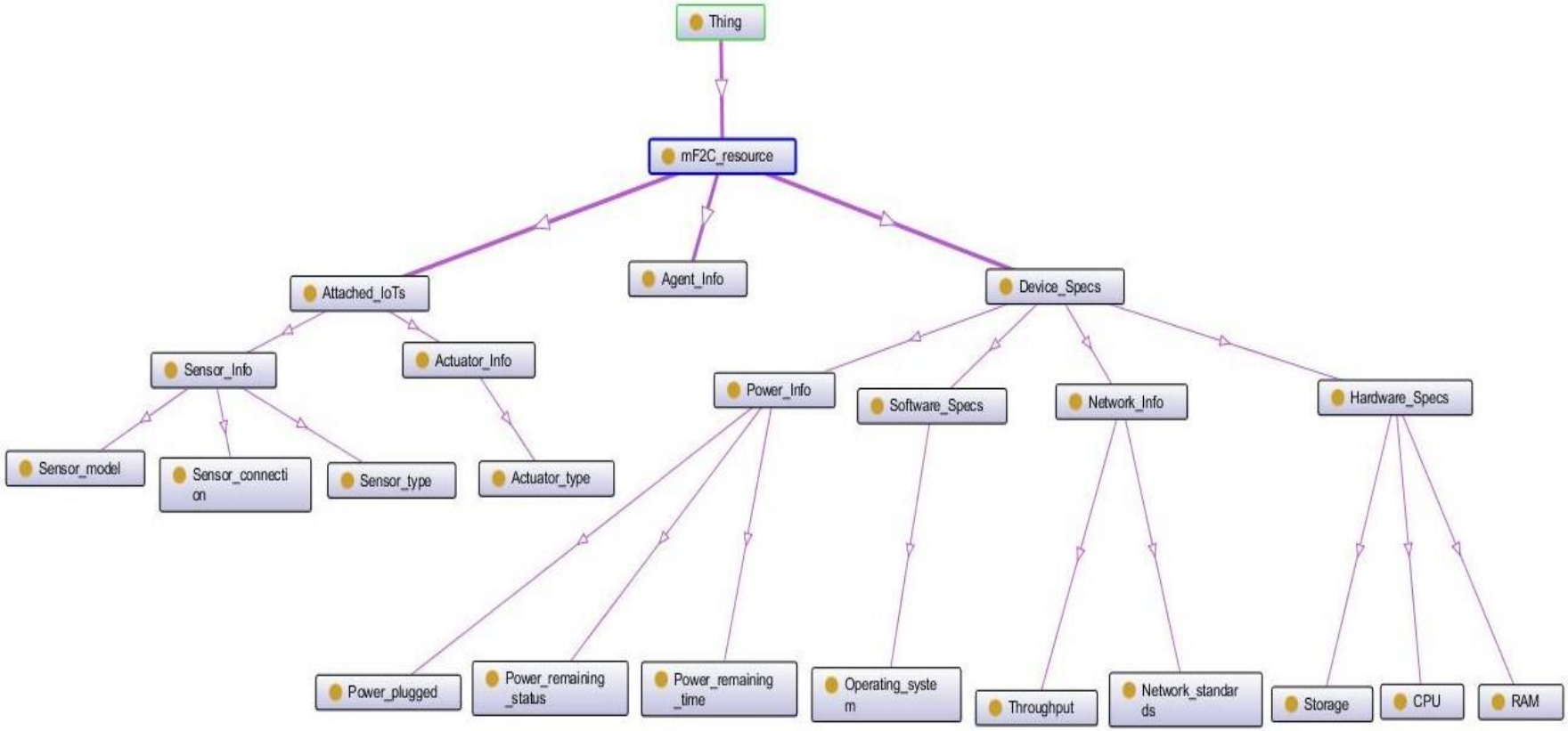

Figure 5. Class Diagram of mF2C resource categorization

3.5. Policies

The Policies module in IT-1 defined a set of rules and parameters regarding the clustering, discovery, aggregation, synchronization, and the leader selection and protection. Moreover, the module included the mechanisms to enforce some of the defined policies (i.e. election of the backup). In IT-2 the leader selection and protection policies have been improved by adding new functionalities:

- **Leader selection policy**, defined as the policy to select which device is the leader among a cluster of devices (or the leader of the agents in the area), was a simple approach consisting of selecting the leader randomly but manually the first time. In IT-2, this policy now includes the possibility of selecting a leader manually, such as was done in IT-1, or automatically by one the selected policies:
  The Passive Leader Selection Policy (PLSP) establishes if the device acts as a Leader of the area only if it fits the requirements to be a leader (defined by policies). Alternatively, the Automatic Leader Selection Policy (ALSP) will promote the device to leader when any of the policies, PLSP or ALSP, are set (ALSP extends PLSP). ALSP establishes that when there is no leader nearby or reachable, if the device is capable to be a leader it becomes one. The algorithms to enforce the PLSP and ALSP inside the module are the Passive Leader Promotion (PLP) and the Automatic Leader Promotion (ALP) respectively. To illustrate these algorithms, Figure 6 shows where and how it works. In the left branch, we show how the PLSP works, if a device fulfils the mandatory requirements to be a leader (imLeader?), it becomes a leader. Whereas in the right branch, we show the ALSP, where first of all the device starts scanning to detect if there is already another reachable leader (the policy also can set the number of reattempts). In the case of not detecting any leader and if the device has the mandatory requirements, it can become the leader of the cluster.

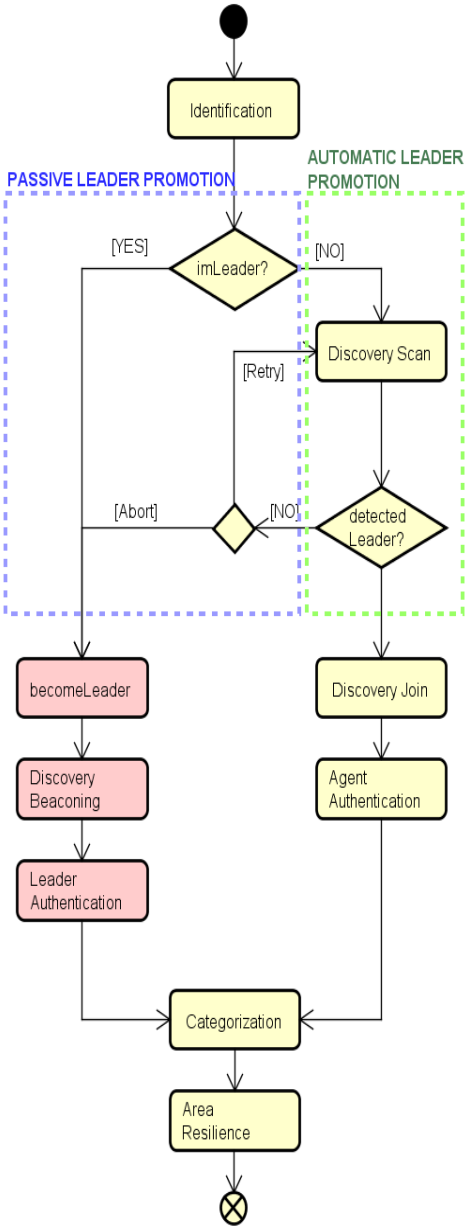

Figure 6. Workflow of leader selection policies

In addition, two new sets of policies have been added to filter if a device is suitable to be a leader and among them which is the best for that duty. Leader Mandatory Requirements (LMR or 1st Order Requirements) are the set of parameters required to be a leader. If any of the requirements are not satisfied, the device is not eligible to be a leader. When a device is suitable, we can rank it using the Leader Discretionary Requirements (LDR or 2nd Order Requirements), that give a score for each of the different aspects defined by the current policy of the system. The algorithm inside the module that implements this feature is the Leader Election Algorithm (LEA).

- **Leader protection policies** consist of selecting an agent acting as a backup for the leader. For IT-2, these sets of policies also include the policy to select the most suitable device in the area to be the backup, instead of the random selection done in IT-1. Moreover, the mechanism for checking if there is a backup present and the backup is detecting is now included in the module.

- **Scanning policies** (in discovery), from a research perspective, two potential policies aiming to optimize the scan process have been investigated. In fact, since the scan relies on the use of wireless resources, it can lead to a high energy consumption for battery-powered mobile devices. This energy consumption is completely wasteful, especially in areas with no leader coverage. Therefore, a first approach to address this issue has been proposed in [6]. In this approach, the previous leader would provide the agent with information about potential leaders that may be encountered along the agent's moving direction. This information is embedded into the discovery beacon, in the same way as other discovery information. Leveraging this information, the scan will only be enabled when a leader is about to be encountered, thus resulting in energy savings in non-covered areas. While this approach relies on topology-awareness, the second approach for optimizing the scan process relies on context-awareness [7]. More specifically, the cellular context of the agent is used, i.e. its serving cell ID and the corresponding signal strength. As a result, if the current agent context has been previously associated with the presence of a certain leader, then, when that context is observed again, the scan will be enabled in order not to miss the opportunity to associate with it. On the contrary, if the agent's context has been correlated with the absence of a leader in the past, then the scan will be disabled, thus improving the energy Core Resource Management operation.

3.6. Core Resource Management operation

3.6.1. Registration and Identification

Figure 7 and Figure 8 show the process of registering a user, downloading the mF2C agent, and initializing the agent including the identification. The specific steps shown in Figure 7 consider the case when the mF2C agent download is not done from the registration page, and the steps are:

1. The user registers himself/herself in the mF2C system providing an email, username and password. Then, the user receives the confirmation email with the link to download the mF2C agent.
2. The user initiates the download, obtaining the mF2C agent installer from GitHub.
3. The user downloads the agent from the repository.
4. The agent initiates the execution.
5. The identification module is executed

5.1 The identification module sends the user and the password to the Webservices to obtain both the IDkey (associated to the user) and the deviceID (associated to the device).

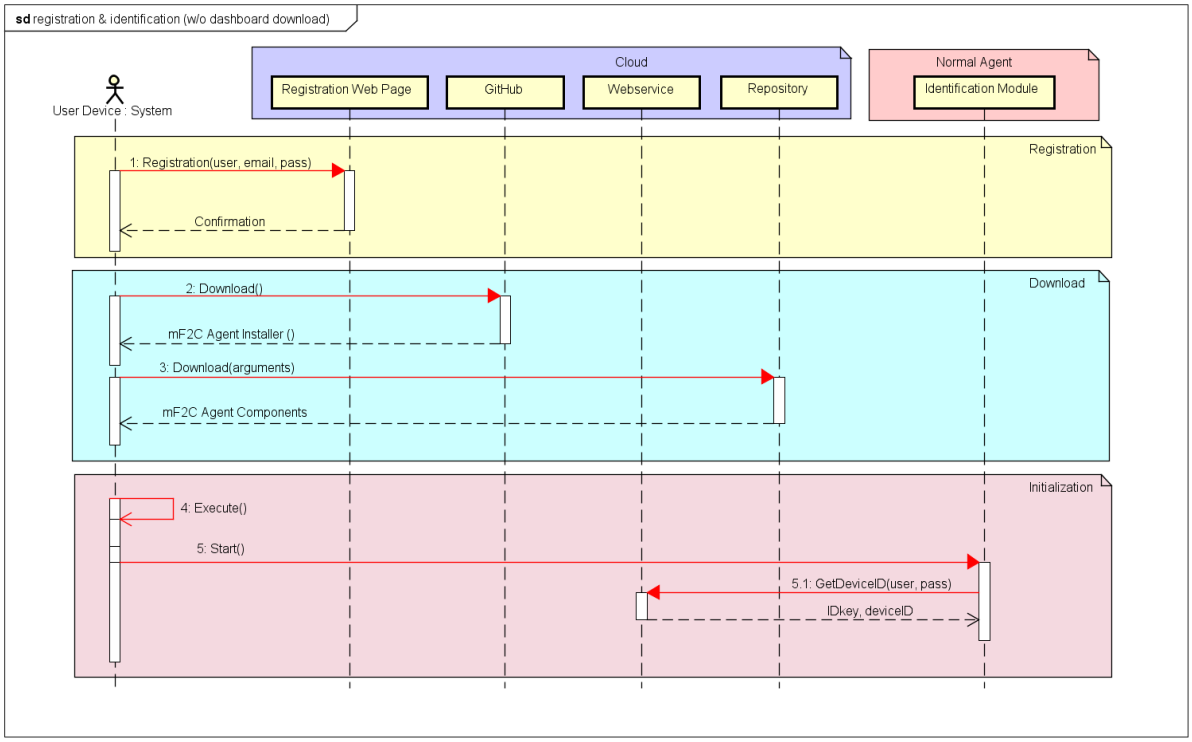

Figure 7. Registration and download using GitHub

The specific steps in Figure 8 consider the case where the download of the agent is initiated in the registration page in the cloud; and in this case when obtaining the mF2C installer from cloud, in step 2, the user also receives the IDkey. Then in step 5.1 the user doesn't need to send the username and the password, but they send the IDkey to the WebServices to obtain the deviceID.

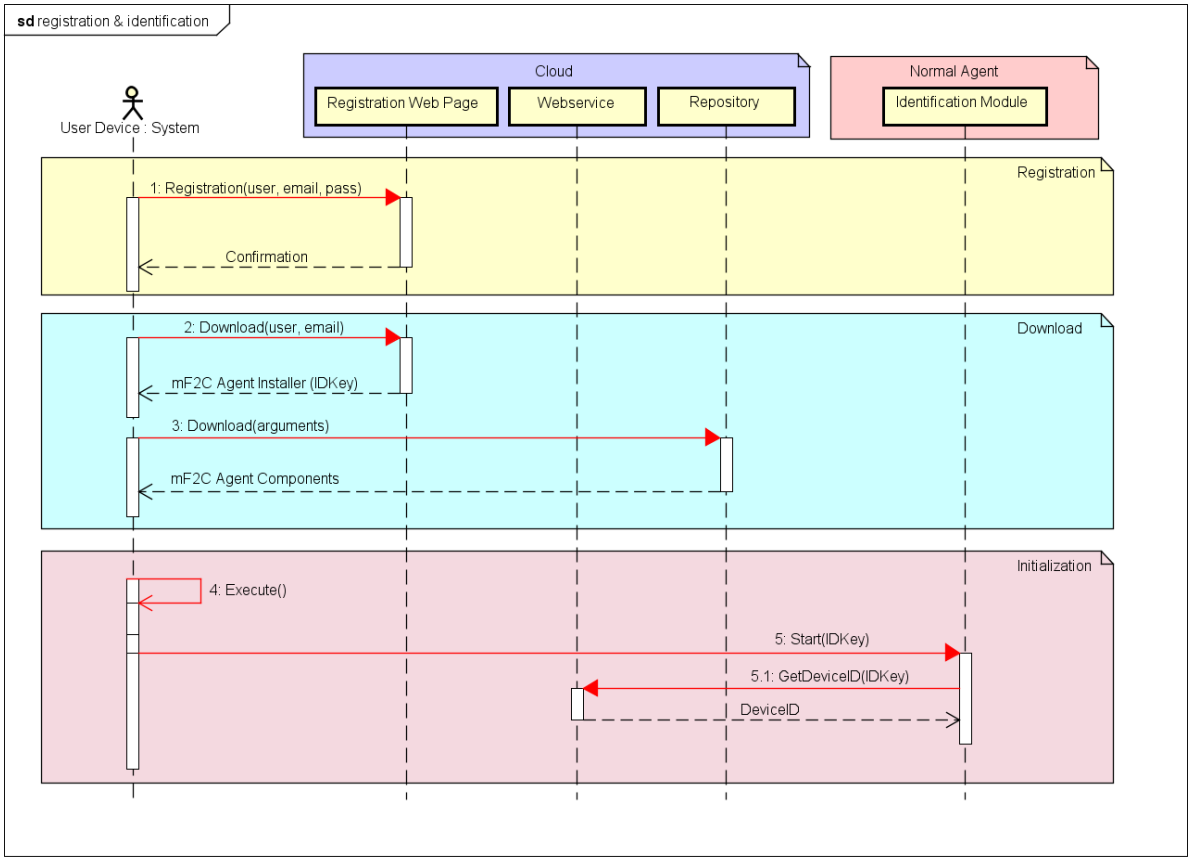

Figure 8. Registration and download using the webpage

3.6.2. Start Agent (Policies, Discovery, Categorization and Authentication)

The workflows in Figure 9 and Figure 10 show the next steps in the agent startup, once the identification module has executed and obtained the deviceID. Specifically, Figure 9 shows the leader Startup, and Figure 10 the normal agent Startup. The steps in the leader Startup are:

1. The policies module is initiated.
2. The policies module requests the deviceID from the identification module.
3. The policies module sends to the discovery different parameters (deviceID, frequency of beacons, etc.) to initiate the beaconing. The discovery module returns the leader IP.
4. The policies module triggers the CAU client module sending the leaderIP, the leaderID (in this case = to deviceID), the deviceID and the IDkey.

4.1 The CAUclient in the leader requests an mF2C Agent certificate by sending a Certificate Signing Request (CSR) to the CAU alongside the leaderIP, leaderID, IDkey and deviceID as additional identity information.

4.1.1 The CAU checks in the database in the cloud if the IDkey exists.

4.1.2 If the IDkey exists (the user is a valid user) the CAU forwards CSR to the CA.

4.1.2.1 The CA signs the certificate, which is returned via the CAU to CAUclient in the leader, and also the policies module is informed that the leader is already authenticated.

5. The policies module requests the execution of the categorization module indicating that the device is a leader.

6. If the categorization is successful, the policies module initiates the execution of the AreaResilience procedure in order to find a backup.

7. The policies module creates in the database the AgentResource with all the information of the device.

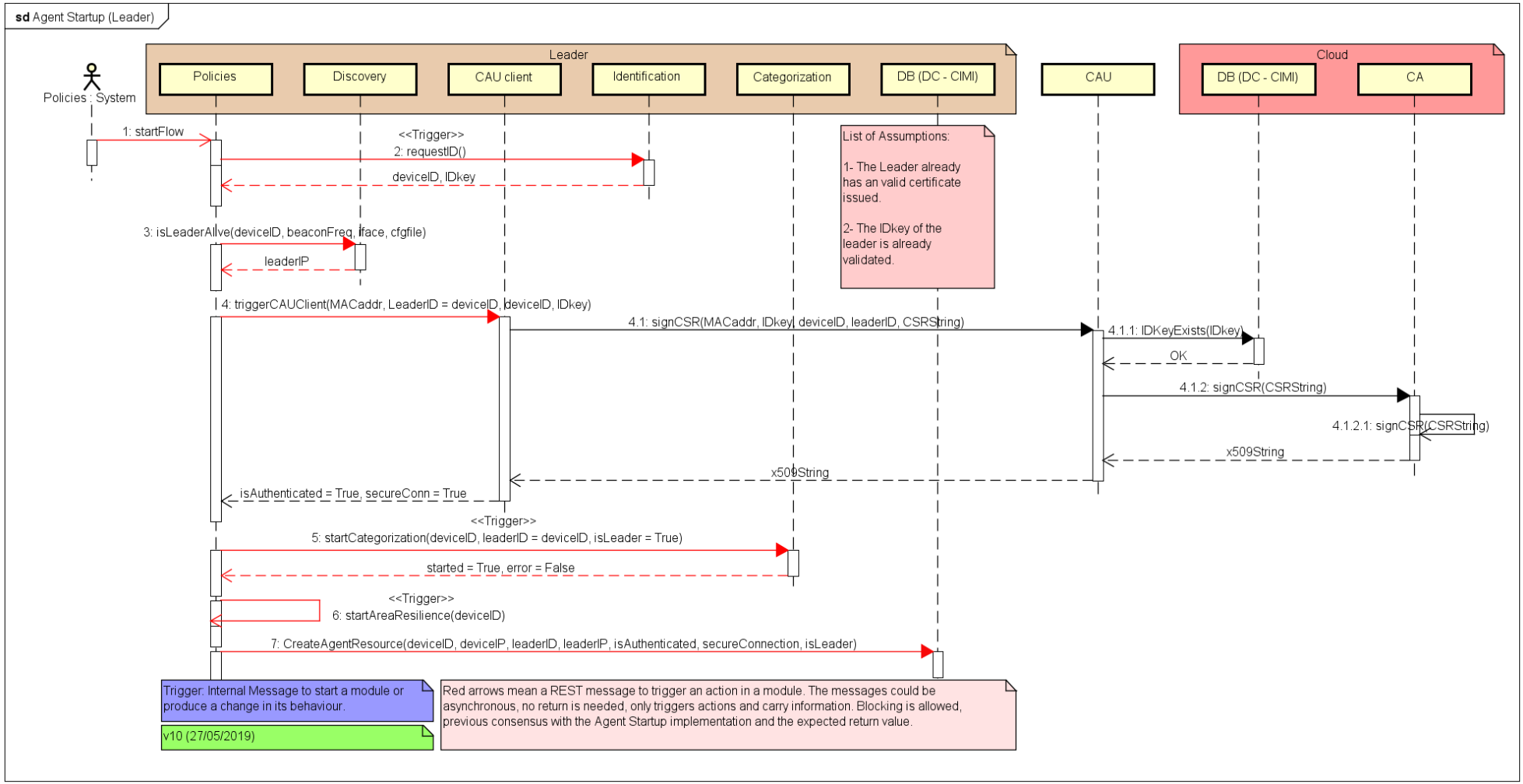

Figure 9. Start agent (leader)

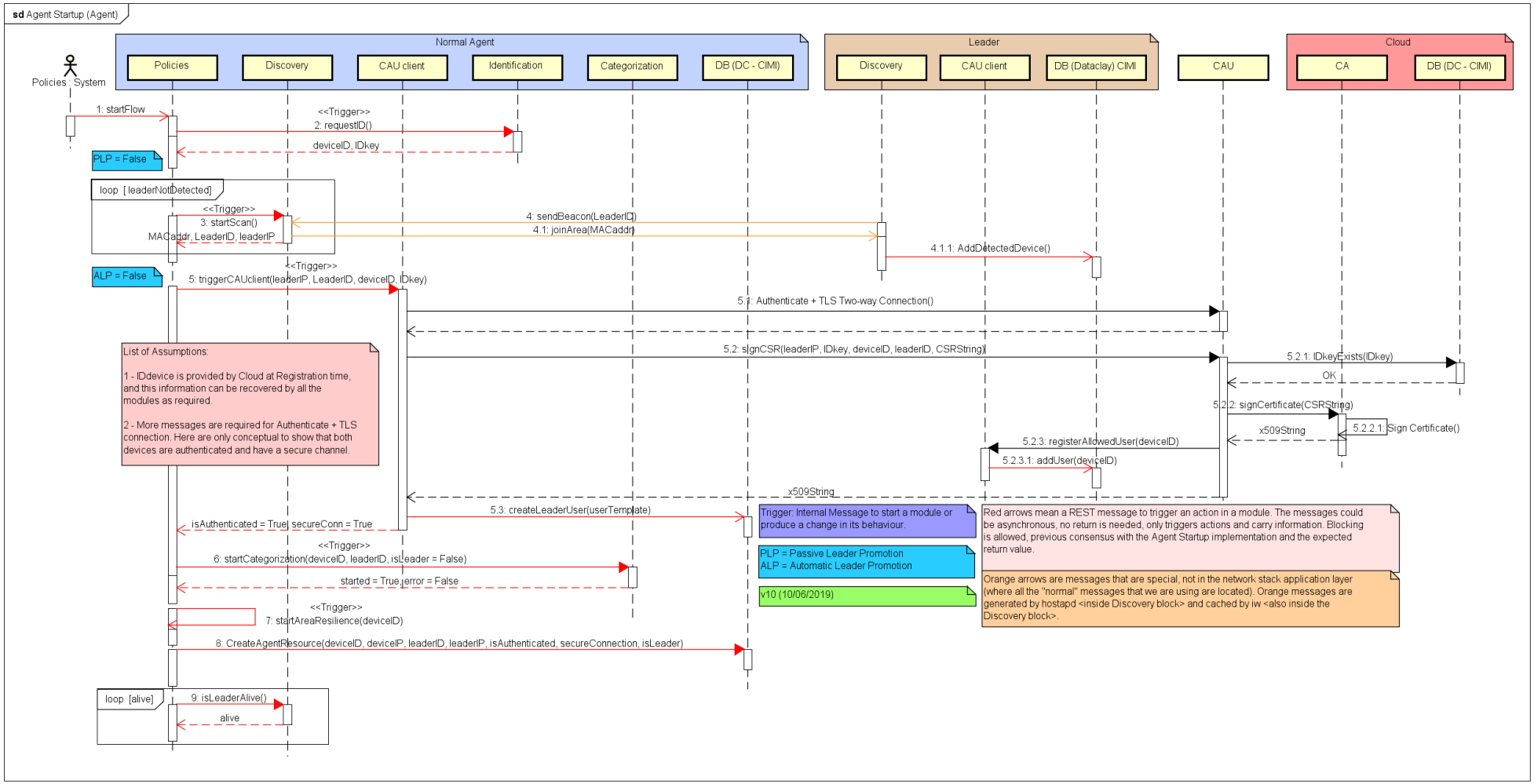

Figure 10. Start agent (Normal agent)

On the other hand, Figure 10 shows the Startup steps in a normal agent (not being a leader), and in this case, it includes the authentication.

1. The policies module is initiated.
2. The policies module requests the deviceID from the identification module.
3. The policies module sends to the discovery module the request to start the scanning. This is done in a loop, reflecting the number of re-attempts.
4. The discovery in the leader sends beacons periodically.

   Once a leader is detected, the discovery module sends to the policies module the MAC address, the leader ID, the deviceID (=leaderID) and the IDkey of the detected leader (red dotted line below step 3)

4.1 The agent joins the leader's area sending its MAC address to the discovery module in the leader.

5. The policies module triggers the CAU client module sending the leader's IP, the leaderID, the deviceID and the IDkey.

5.1 A secure connection is established between the CAU-client and the CAU.

5.2 The CAUclient in the device requests an Agent certificate by sending a Certificate Signing Request (CSR) to the CAU accompanied by its MAC address, deviceID, IDkey and the detected leader deviceID to facilitate identification

5.2.1 The CAU checks in the database in the cloud if the Agent's IDkey exists.

5.2.2 If the IDkey exists (the user is a valid user) the CAU forwards the CSR to the CA.

5.2.2.1. The CA signs the certificate, returning it via the CAU to the CAU client in the device.

5.3 On receiving the Agent certificate, the CAU client in the device creates the leader user in the device database, and sends to the policies block the confirmation that the device is authenticated.

6. The policies module initiates the execution of the categorization module, sending the deviceID, the leaderID and indicating that the device is not a leader.
7. If the categorization is successful the policies module executes the area resilience procedure, to participate in the process of selecting a backup.
8. The policies module creates in the database the AgentResource with all the information of the device.
9. The policies module executes a loop checking with the discovery module if the leader is alive.

3.6.3. Leader election

In this section, we consider all the policies related to elect a leader, both, the policy of selecting a backup node, and also in the case of leader failure, when this backup is promoted to be leader, the election of a new backup.

3.6.3.1. Backup selection

Figure 11 shows the workflow for selecting a backup node, which is slightly modified from that proposed in deliverable D3.3, [3]. In the workflow, we only show a leader and two devices (normal agents) that are candidates to be backups, and we consider the simplified case where we select the backup among these two candidates, selecting the first fulfilling the requirements to be a backup (to be a leader); and not showing how the selection policy ranks the candidates taking into account their capacities. The specific steps in the workflow are:

1. The policies module in the leader initiates the AreaResilience procedure.
2. The policies requests from the leader's database its own device info, obtaining the static and dynamic info as well as the confirmation that it is a leader. The leader starts a loop with all the devices (agents) in the cluster:
3. Obtaining in the leader's database the information of the device, static and dynamic.
4. Checking if the device fulfils the requirements to be a backup (that is, to be a leader).
5. The policies module of the leader sends to the policies module in the selected device the information that is promoted to be the backup; and the device agrees.
6. The leader stores the deviceID of the selected backup.
7. The leader stops
8. Continuously the backup agent sends keepalive messages to the leader in order to check its availability; and the leaders acknowledges.
9. The backup stops.
10. If after a predefined period of time, and after some predefined reattempts the backup does not receive the acknowledgement from the leader, it switches to leader, and the policies module sends this information to the discovery module.
11. The policies module sends this switching from being a backup to be a leader to the categorization module.
12. The policies module requests to the database all the information, static and dynamic about the device.

Steps 13, 14, 15, 16 and 17 are similar to 3, 4, 5 and 6 and show how the new leader selects the new backup.

Steps 18 and 19 are similar to 8 and 9 and show how the new backup is continuously checking if the new leader is alive.

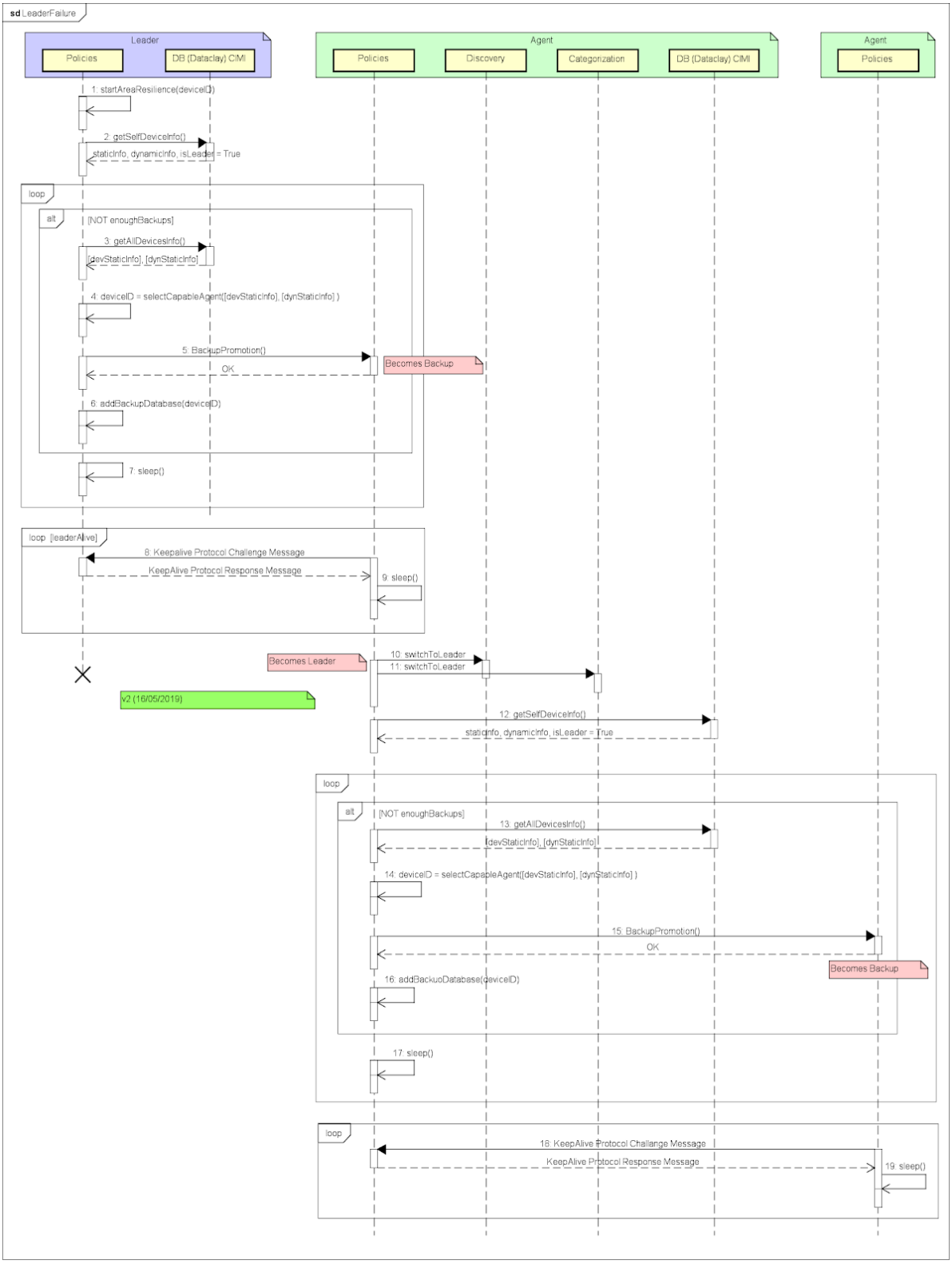

Figure 11. Backup selection workflow

3.6.3.2. Leader re-election

Finally, we show in Figure 12 the workflow for re-electing a new leader, which in fact is the re-election of a new backup that will become leader when the leader fails. This procedure and the policies associated for selecting a new leader are new for IT-2, since in IT-1 the leader selection was done manually. The specific steps in this re-election are:

1. The policies module in the leader initiates the re-election procedure, indicating that agent 4 will be the device in the cluster with higher preference to be a backup, and then changing its range from 1 to 0.
2. The policies modules in the leader requests all the information about agent 4 from the leader's database.
3. The policies module in the leader sends to the policies module in agent 4 its new preference to be a backup: 0; and agent 4 agrees, becoming the new backup.
4. The policies module in the leader demotes all the agents in the cluster except agent 4.

4.1 The policies module sends this demotion to all the nodes in the cluster, except to agent 4; and the devices (agents) agree. For example, the agent with preference 0 now becomes an agent with preference 1.

5. The policies module in agent 4 (new backup) sends keepalive messages continuously to the leader; and the leader acknowledges.
6. The backup (agent 4) waits.
7. The leader demotes itself, and stops being the leader. The backup (agent 4) will detect this demotion because the leader stops acknowledging the keepalive messages. For agent 4 (backup) the leader will be down, and it will become the new leader.

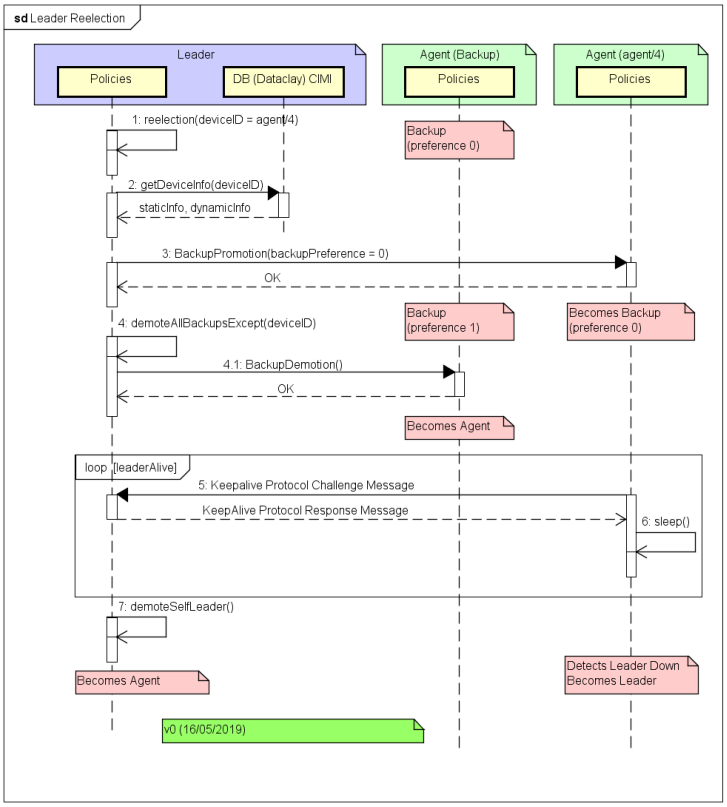

Figure 12. Leader re-election workflow

4. User Management Design

The User Management module is responsible for managing the user-profile and the sharing-model properties of the mF2C users. These properties define how the users' devices participate in mF2C, and what they want to share with other users. The user-profile definition specifies if the user's device acts as a contributor by sharing its resources, or as a service consumer for launching and executing mF2C services. The sharing-model specifies the resources and number of applications that a user allows to share in a specific device. The User Management component is also responsible for checking that devices running in mF2C comply with these profiles and sharing model properties.

The focus of work during this second iteration includes the improvement of the different components of the User Management module, and the final implementation of the assessment module responsible for checking the user-profile and sharing-model properties, which was delayed for this Iteration. The "QoS enforcement" subcomponent described in previous deliverables was renamed to "Assessment". Thus, the User Management is composed by the following subcomponents:

- Profiling
- Sharing Model
- Assessment

Internal Architecture

The internal architecture of this module presents some major changes with respect to the one presented in Iteration 1. On one hand, all the subcomponents interact directly with **CIMI** to get all the information (resources) handled and needed by them. And on the other hand, the Assessment module calls or sends events to the Lifecycle (Platform Manager) when it detects a violation during the assessment process. This architecture is shown in next figure:

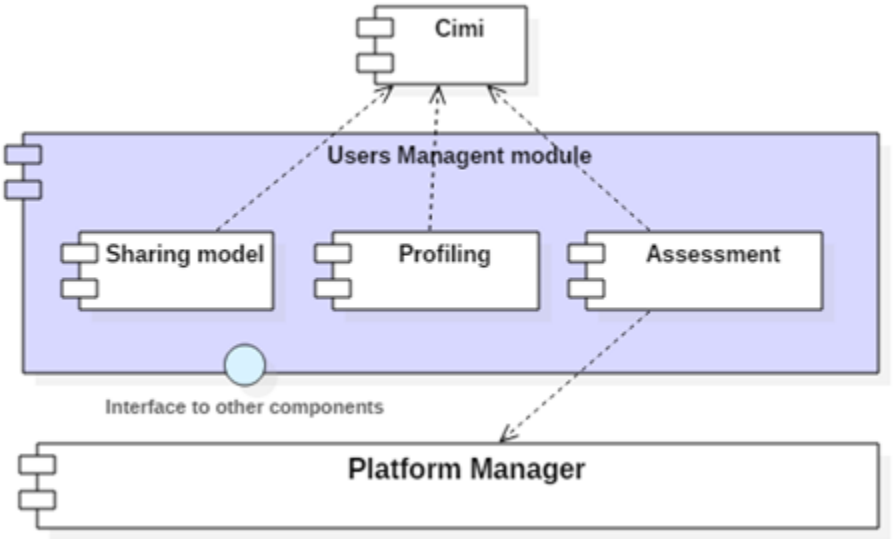

Figure 13. User Management module

4.1. Profiling

The profiling component is responsible for defining how the user's device behaves in an mF2C cluster. There are three available setup options: a service consumer is a device that can launch mF2C services; a resource contributor is a device that shares its resources to other users; and finally, both options at the same time. In other words, this component handles the following properties: *service_consumer* and *resource_contributor*.

This section presents also the updated workflows of the Profiling subcomponent. They present some changes with respect to the previous version described in deliverable D3.3, [3]. In Iteration 2 the user-profile properties are generated automatically during the initialization of the User Management component. The initial properties (*service_consumer* and *resource_contributor*) are taken from the environment variables and the agent resource (information about the device), and then stored in CIMI. The following picture depicts this workflow:

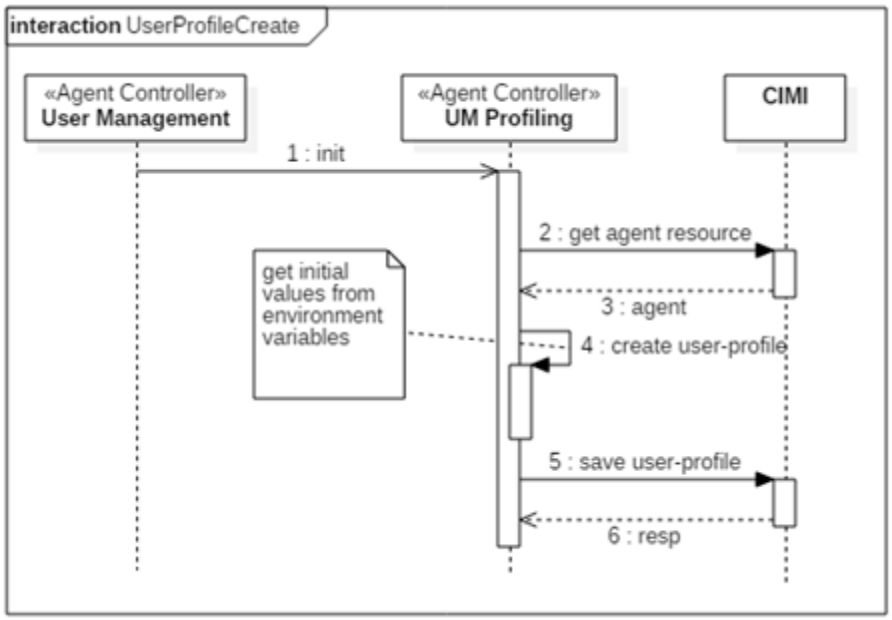

Figure 14. Profile properties initialization.

A user can modify these properties by accessing the device's mF2C GUI and editing there the user-profile values. First, the Profiling module gets the request from the GUI component, and then it process the information and stores it in CIMI:

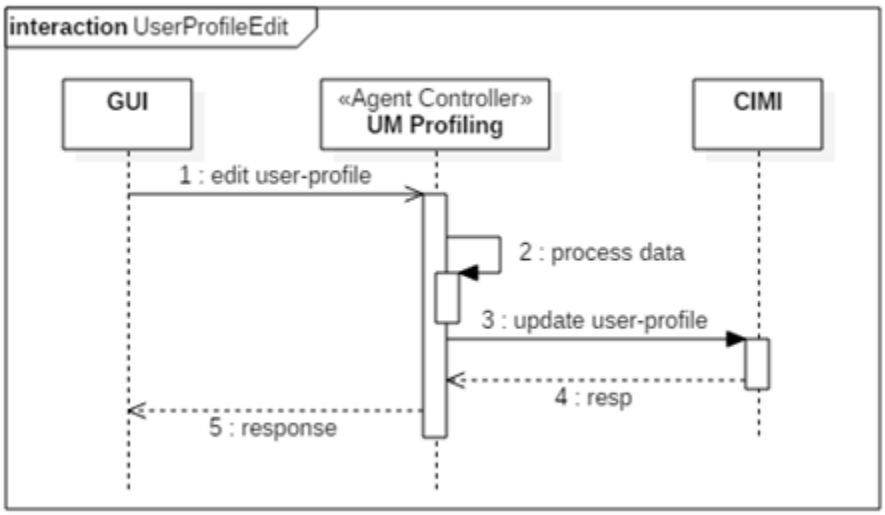

Figure 15. Profile properties update.

Finally, there is a new workflow defined for Iteration 2. This new workflow is responsible for deleting all the information of a user in mF2C according to the new GDPR policies. During this process, the Profiling module first deletes all user-profile and sharing-model resources based on a *username* or user identifier, and then it deletes the user resource of this *username*.

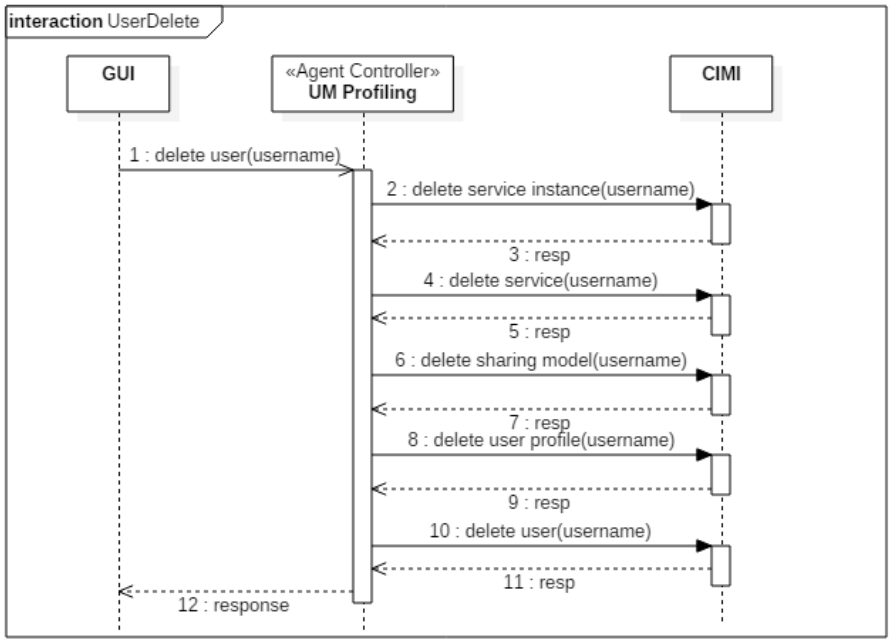

Figure 16. Delete user from mF2C.

4.2. Sharing model

If a user's device is defined as resource contributor, then the sharing model properties are used to define which resources can be shared, and how many mF2C applications can run in the device. This module is responsible for handling the following properties:

- maximum number of mF2C applications allowed to run in the device
- maximum CPU, RAM and Storage usage
- battery level limit (when to stop sharing resources)

The initialization workflow (creation of the sharing model resource) also presents some major changes with respect to the one defined in Iteration 1. The sharing model resource is also generated automatically during the initialization of the User Management component (see next picture).

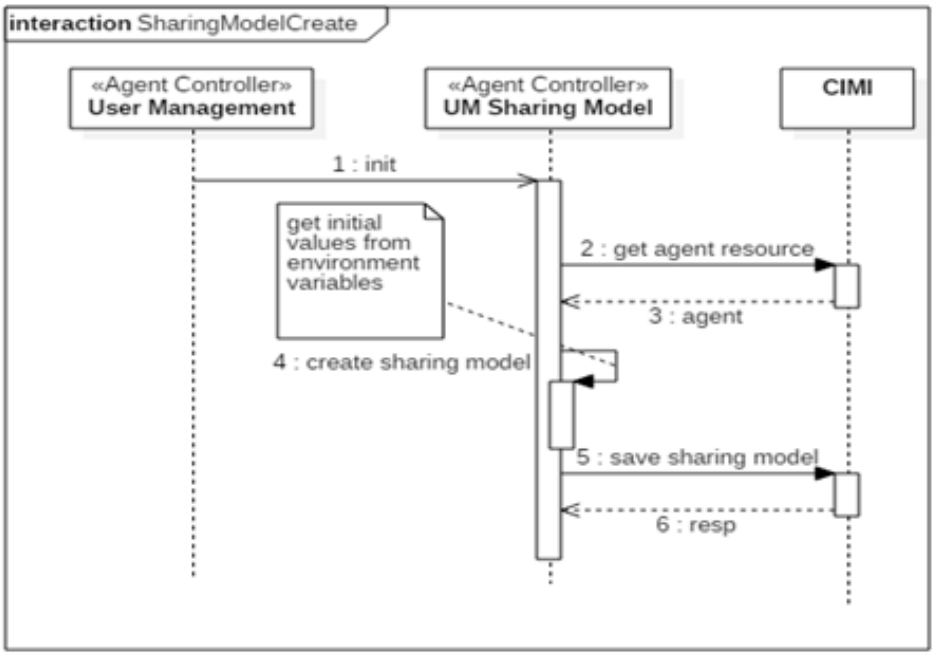

Figure 17. Sharing model properties initialization.

The user can modify these properties by accessing the device's mF2C GUI and editing there the user-profile values, the same way it was described in the previous section (Profiling):

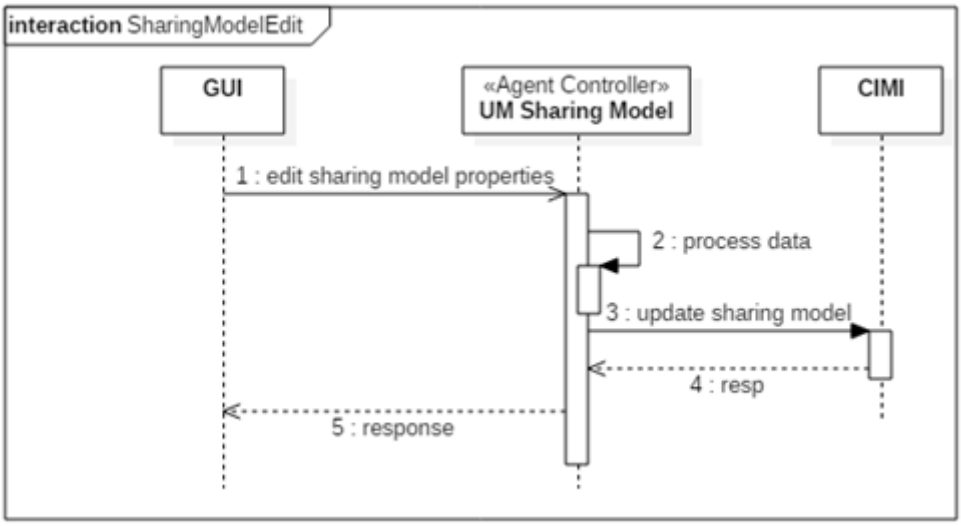

Figure 18. Sharing model properties update.

4.3. Assessment

Finally, the Assessment module is responsible for checking that a mF2C device (or agent) doesn't share more resources than defined by the user. This assessment process is continuously running in the background, checking the properties defined by the user, and comparing them with the status of the device.

The workflow of this assessment process (Figure 19) in each mF2C device is the following:

- Get the user-profile information
- Get the sharing-model information

- Get resources used by the device
- Get the number of mF2C applications running in the device
- Compare device's status with the constraints defined by the user
- If there is a violation
    - Generate an event
    - Send a message to the Lifecycle

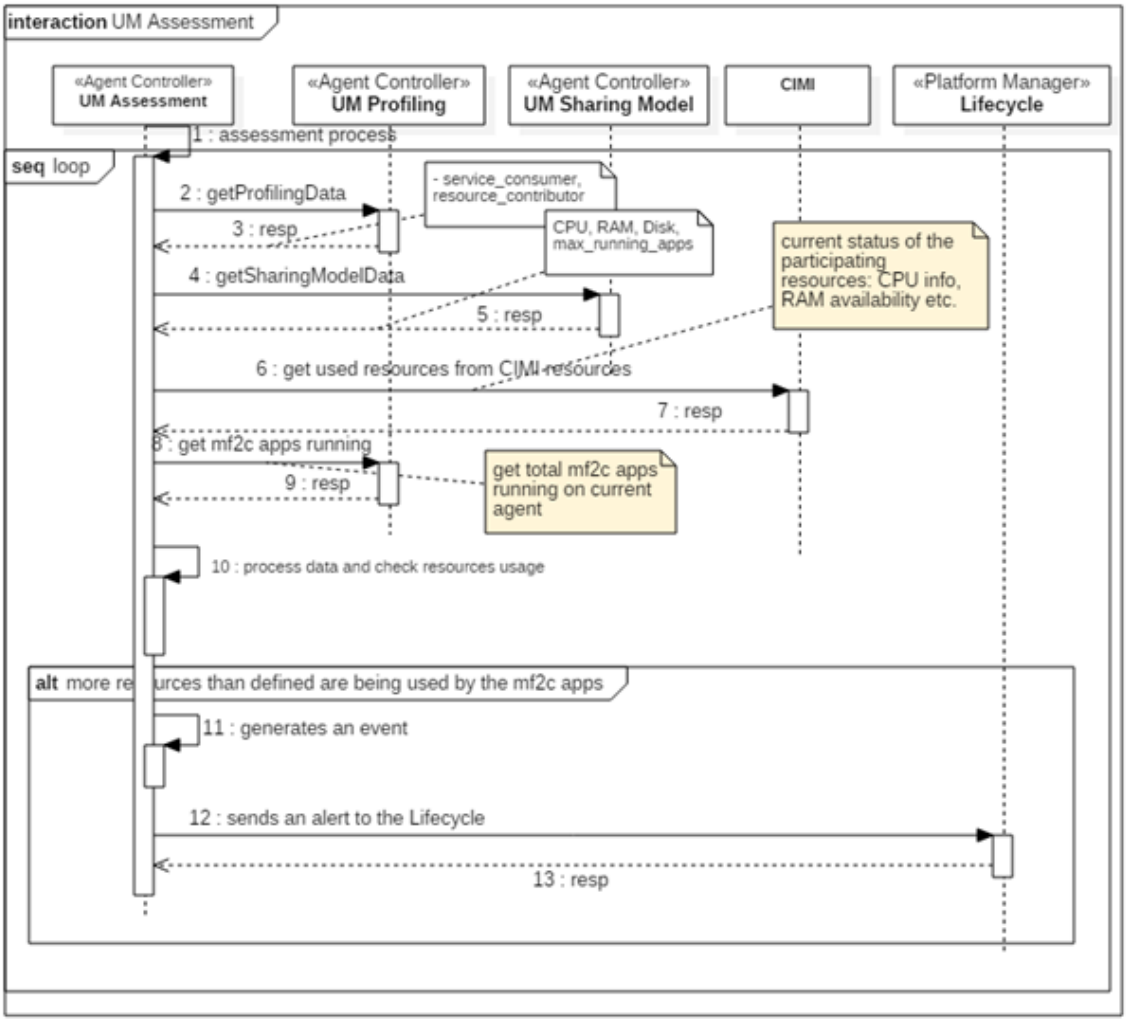

Figure 19. Assessment process.

5. Security

D3.2 Security and Privacy Aspects for Agent Controller (IT-2), [4] has already provided a full description of the requirements and design of the IT2 Security Module. To avoid duplication, we summarise below supplementary information arising and relevant enhancements to the design during the implementation phase.

5.1. Reverse Proxy

In IT-2, we replaced the self-signed certificate used by Traefik which implements the reverse proxy used to protect the Agent blocks exposed endpoints (see D3.2 [4] Section 4.7.1). The motivation is to help bake in the Agent's unique mF2C identity and to facilitate trust between mF2C components.

5.2. AC Library

The AC Library[1] is a new component implemented for IT-2 to facilitate end-to-end message level data security in line with the mF2C data security policy (see D3.1 [8]) which classifies data into three categories: Public, Protected and Private. Figure 20 gives an overview of its usage. The main objectives of the rewrite are to:

- streamline the implementation by removing:
    - dependencies on Bouncycastle[2] artefacts and reverting to the native JRE security and light-weight JOSE4j[3] libraries
    - the communication protocol wrappers and the built-in MQTT message broker. We originally envisaged that an mF2C component could use the library to abstract away the communication protocols when exchanging messages. However, it became apparent that networking is defined at the Docker container and VPN levels while the communication protocol is to a large extent constrained to HTTPs through adopting CIMI as the single interface to an Agent. (CIMI has a Resource-oriented architecture and exposes a HTTPs REST interface to its managed resources.) Consequently, the protocol wrappers complicate rather than simplify both inter- and intra-Agent communication
- support interoperability and leverage trialled-and-tested security protocols. The new library adopts the suite of JSON-based security standards (JWS, JWE and JWT: RFCs 7515, 7516 and 7159 respectively) to encapsulate messages as secure tokens and to enforce data integrity and confidentiality in line with the JWS and JWE protocols (see Figure 21 for an example JWS)
- provide library functions for mF2C components to work with the mF2C PKI trust model, e.g. create identity JWT (see Figure 21) to assert Agent identity (see D3.2, [4], Section 3.2) and to create and verify secure message tokens.

[1] https://github.com/mF2C/aclib.git

[2] https://www.bouncycastle.org/

[3] https://bitbucket.org/b_c/jose4j/wiki/Home

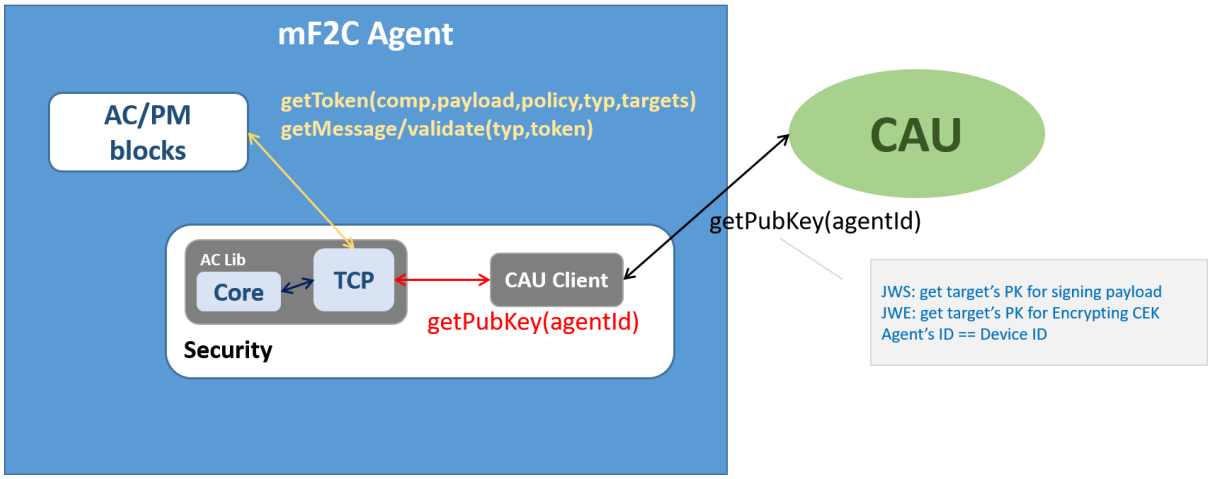

Figure 20. An overview of the refactored AC Library usage

Token specifications:

An Agent's Identity token is structured as a signed JWS with the payload containing the JWT claims

List of claims (in Payload)

CODE	DESCRIPTION
iss	Issuer ID. This would be the requesting Agent's deviceID. In mF2C, the issuer is the same as the presenter.
aud	Audience (intended recipient/s). This is set to the target agent's device ID.
exp	Expiration time in NumericDate format, set to 10 mins
nbf	Validity start time, set to 2 mins.
iat	Issued at time.
jti	Unique ID of the JWT.

JWT Structure = signed JWS

BASE64URL(UTF8(JWS Protected Header)) || '.' ||
BASE64URL(JWS Payload) || '.' ||
BASE64URL(JWS Signature) Structure

eyJ0eXAiOiJKV1QiLCJraWQiOiJrZXkxIiwiYWxnIjoiUlMyNTYifQ
.
eyJpc3MiOiIwZjg0OGQ4Zml3OGNiZTU2MTU1MDdlZjVhMTk4ZjY2MGFjODlhM2FlMDNiOTVlNzlkNGViZmIzNDY2YzIwZDU0ZTlhN
WQ5YjljNDFmODhjNzgyZDFmNjdiMzIyMzFkMzFiNGZhZGE4ZDJmOWRkMzFhNGQ4ODQ2ODFiNzg0ZWM1YSIsImF1ZCI6IjBmODQ
4ZDhmYjc4Y2JlNTYxNTUwN2VmNWExOThmNjYwYWM4OWEzYWUwM2I5NWU3OWQ0ZWJmYjM0NjZjMjBkNTRlOWE1ZDliOWM
0MWY4OGM3ODJkMWY2N2IzMjIzMWQzMWI0ZmFkYThkMmY5ZGQzMWE0ZDg4NDY4MWI3ODRlYzVhIiwiZXhwIjoxNTU4NTIzMj
k0LCJqdGkiOiI1ejRtWGFGd1dXR21fYU1la3NpZlZ3IiwiaWF0IjoxNTU4NTIyNjk0LCJuYmYiOjE1NTg1MjI1NzR9
.
J8sP7LZ61MirRWyArdKadZ3ksq0mxphVWtHA7cX59M8uHKIR-k87wwKJNskAeJryJrR9EEJhzrwr-
TvtUbb7ltQsF0IRe65kk6gs_iTnOPKAZuJQUOnbho3hhlgwRZhfzX7KgB-
iJOFmBOxkeyBtGeanbO0oPnW0A5LXFn1y6PhVwV_UUoeZ-TI8ed8ANyli8NUh70yZ8Sn7wOLU3CQnRFHfdy2-
mfEDLQjmX1xhcRfTQ-I79CLobWa8AIVYFiAj5-
085oJRbvlGRiA8rvfce0j1cLmFf5p69tPAtNjIQuDUi7fcunmcrvrLCSzoW7x1HC5EfQxq83Mp7R-LgrFliw
Example

Figure 21. Example JWT Identity Token structured as a JSON web signature object

5.3. Agent Authentication

In IT-2, we extend our security implementation beyond simply issuing an X.509 certificate to identify an Agent to actually using the credential to secure message exchanges (see Section 5.2) and to assert identity to support Agent authentication. The latter covers vertical communication via a public network between the Leader Agent and its children Agents. Figure 22 illustrates how the AC Library and CAU Client work together with the CAU middleware to authenticate an Agent's identity.

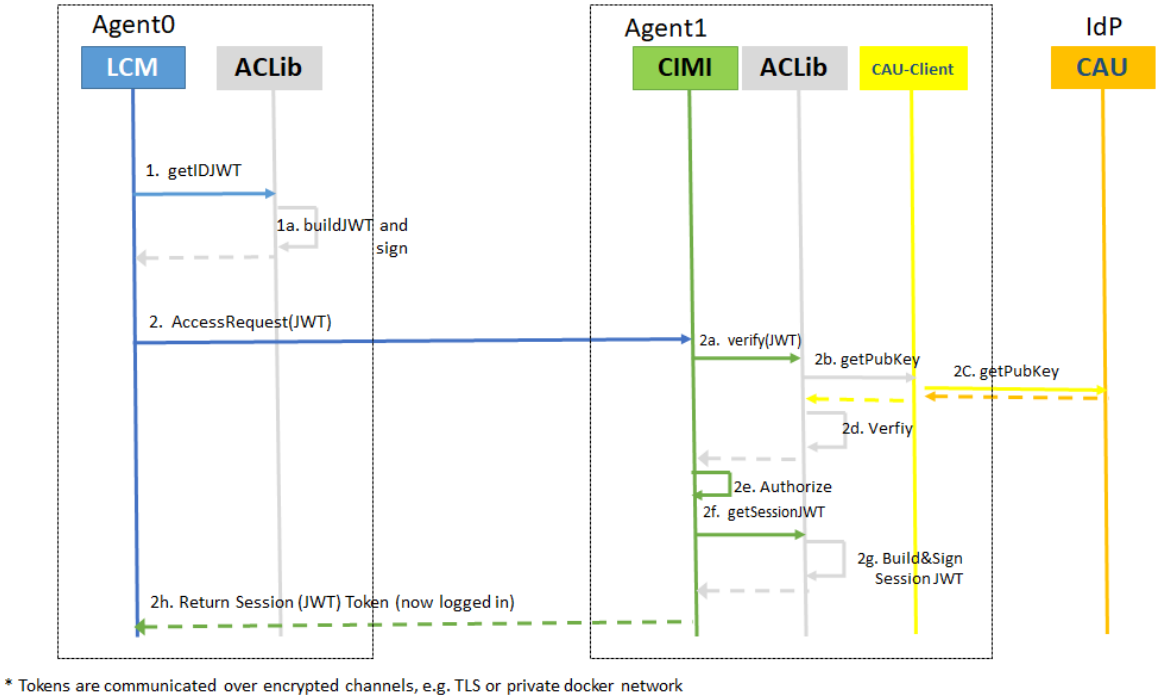

Figure 22. Securing inter-agent communication (Life Cycle Manager, LCM, use case).

In mF2C, an Agent is uniquely identified via its X.509 certificate issued during the Discovery, Authentication and Categorisation process (D5.1, [9] Section 3.2). In a PKI trust model, the private and public keys tied to the certificate constitute asymmetric secrets used in cryptographic operations for authentication, data protection and message non-repudiation. In mF2C, the loosely coupled CAU middleware (see D3.2, [4], Section 4.4) serves as the identity provider (IdP) which has a global knowledge of mF2C Agent public keys. The CAU intelligence is accessed via the CAU Client built-in to each Agent. The AC Library in turn uses the public key information, retrieved via the CAU Client, in its handy cryptographic functions to create and validate message (JWS, JWE) and identity tokens (signed JWT).

From a security point of view, we need to both authenticate and authorise inter-agent communication. Authorisation however rests with CIMI, the single entry-point to an Agent. CIMI has an internal access control list which it uses to authorise an authenticated Agent's request to access the different Agent blocks exposed as REST resources by the CIMI API.

5.4. VPN

As a part of the network security, mF2C provides a Virtual Private Network (VPN) in IT-2. As the name suggests, a VPN layers a private network on top of the existing network (which in turn may be using Wi-Fi). In order to prevent an unauthorised agent from hijacking a session, the VPN server authenticates itself with a certificate from the trusted infrastructure. Conversely, the client simply authenticates itself to the server using its Agent credentials. We could provide additional security by using shared secrets, but this should not be necessary. This design choice has the a dvantage that there are no secrets in the client's (Agent or Micro Agent) configuration; nor are there any secrets in the servers. This advantage, in turn, makes it easy to set up and deploy both the client and the server; all that is required is that the client is deployed with the trust anchor (CA certificate) for the server's certificate, and the server is deployed with the trust anchor for the client. By the time the client's VPN configuration is started, the client credential should already be present.

Note that, while the server can run as a standalone service, it may not make sense to run the client as a standalone service, because the client, once started, has access to the VPN, but only from within the

client container. It may make more sense to build an image with an OpenVPN executable, and run the configuration script within this image to configure and start up OpenVPN. The script is designed to be the only thing that is needed, other than the credential, and can be configured through environment variables to launch the VPN client as a daemon or in the foreground (the latter meaning that the container exits when, and only when, the VPN client exits.)

Within each VPN, clients can ping each other and connect to each other, using a private address space. The client is also, by default, configured with routing information which enables them to access services outside of the VPN; this accesses tunnels through the VPN server. Thus, in a sensitive environment, a network could be deployed with no external access at all provided to clients, and clients will need to authenticate and connect to the VPN in order to access other clients, or the outside world. This level of security is quite common in corporate environments where a wireless network is available to staff but not to unauthorised clients. Moreover, these scenarios could be extended within mF2C to provide separate networks for different use cases (which already have distinct PKIs) - the current deployment focuses on Agents only, and hence uses only the Fog and Infrastructure CAs, as described in D3.2, [4] (section 4.4 in particular.)

6. Event Manager

The Event Manager is a standalone microservice which has been developed out of the need from other internal components to be advised of certain events occurring throughout the system.

This new component has been built to lookup into an existing CIMI server and perform regular checks based on user criteria.

It includes a Server Side Event (SSE) based server which loads a user configuration at startup, where a set of jobs are defined. These jobs are scheduled to run on a regular basis, performing queries to the CIMI server. These queries are crafted by the component developers who need to be advised of the occurrence of an event. CIMI offers a system resource called ***event*** which can also be used to assist the Event Manager. This resource is populated by CIMI itself upon certain actions. For example, whenever a ***device*** is deleted, CIMI will register that action as an event and save it in ***event***. The jobs from the Event Manager can simply query CIMI to lookup for new entries of this ***event*** resource, and publish a Server Side Event to all subscribed clients, on a specific channel. Please refer to the simplified architecture of the Event Manager shown in the figure below.

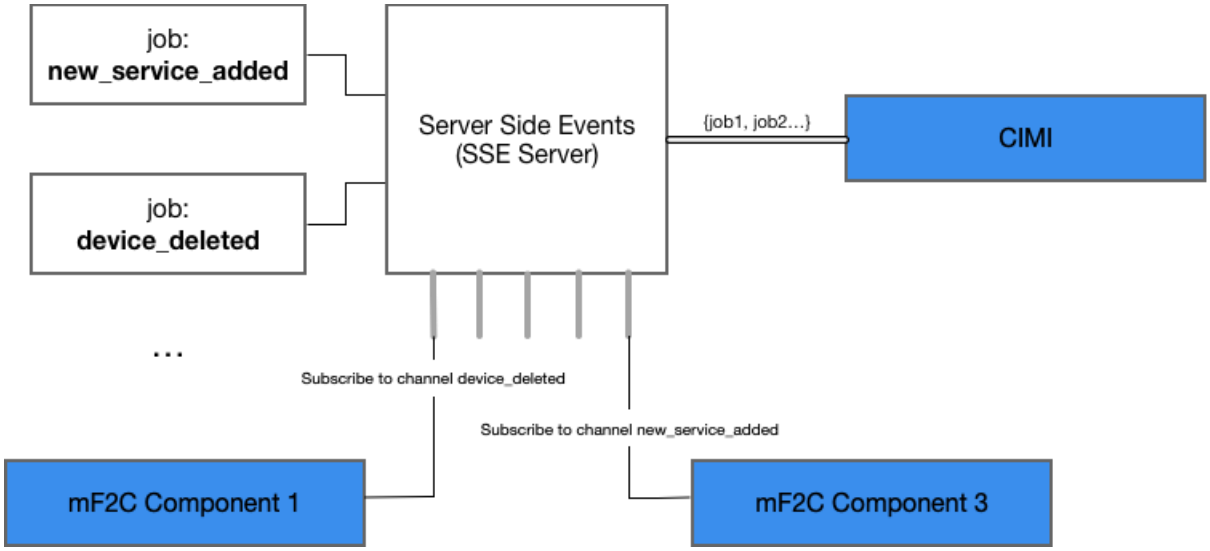

Figure 23. Event Manager workflow.

7. Dashboard/GUI

This is new functionality added to the mF2C agent design for IT-2. Initially, in IT-1, we proposed as a GUI, a frontend in the cloud with two main functionalities, registering people (see section 3.1) and a repository of mF2C services (service catalogue) to be used for both adding new services to mF2C, and launching the services as described in previous deliverables.

For IT-2, we extend these GUI capacities by adding a dashboard intended to facilitate the supervision of the mF2C system, not from a user perspective, but rather from a mF2C provider one. Indeed, the dashboard is proposed to be deployed on all agents, and would be addressed, not to the mF2C user interested in registering devices and launching services, but to the one controlling the system (the mF2C provider or operator), providing control information such as the state of resources, events, SLA violations, etc. We envision this component is also a critical piece of the mF2C system since it will be the one facilitating the interaction with potential mF2C deployers (cities, communities, etc.). Certainly, the main aim in IT-2 regarding the dashboard will focus on setting the key functionalities an mF2C control system is supposed to have in terms of the information that should be highlighted to have a comprehensive picture about how the system works, and hence, will not put a large focus on design cosmetics, since we all understand that this phase could be more elaborate in a real commercial product design.

The following subsections describe the designs done so far mainly highlighting the control or status information we envision the system could provide.

7.1. Landing page

This subsection describes the design of the landing page a control operator could see when accessing mF2C. As shown in Figure 24, the home page could contain a menu, located up at the left corner, giving access to the whole set of mF2C dashboard functionalities, to be described in the next subsections, as follows:

- Dashboard: The current landing page
- CIMI Resource browser
- Events Status
- SLA Violations
- User management
- Service Launching Control (LC) and instances

We think that by browsing this information the control operator may have all information required to know how the system is performing and if needed, take the proper decisions when possible.

On the other hand, in the right side of the landing page we propose to include a sort of summary of information to show key glimpses about system performance, including:

- A pop-up warning banner about some events
- The topology of the device (where the dashboard is running) and its children if and only if the device is a leader, otherwise the topology will be only the device
- When clicking in one of the devices of the topology, graphical information about this specific device (including CPU, memory and disk) is visualized
- Two tables with additional information about the specific device, such as IDkey, deviceID, operating system, architecture, number of cores, etc.

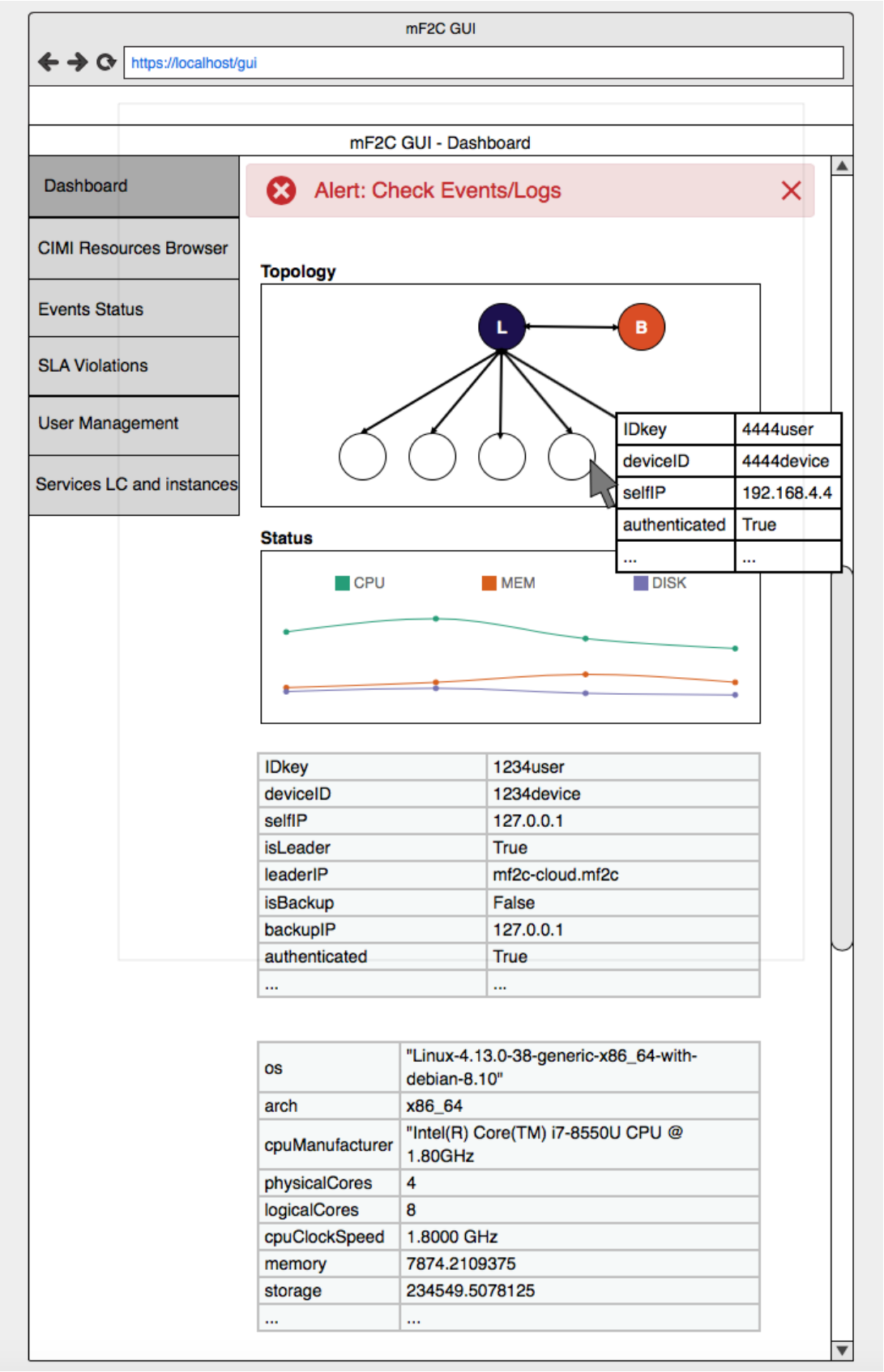

Figure 24. Dashboard landing page

7.2. CIMI Resource Browser

The graphical user interface for the CIMI server will simply be an API explorer which lets users visualize and manage CIMI resources without the need to manipulate raw data nor complex RESTful API queries. There is already an open source GUI from SIXSQ which can be fine-tuned and used for this purpose. The source code can be found at https://github.com/nuvla/ui. A screenshot of this graphical interface is shown below.

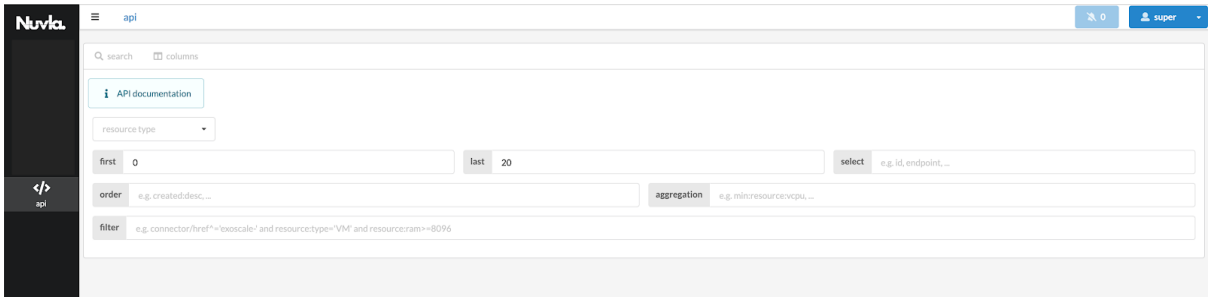

Figure 25. Graphical interface for the CIMI server.

7.3. Events Status

Changes to the overall system Landscape are monitored by the Landscaper component. Any time a change occurs, e.g. a new docker service is created, the new service is added to the landscape, together with the relationships of the new service with the previously existing components of the landscape. The changes are also written to the application log of the Landscaper. By monitoring the application log, an external agent can get a picture of the events as they happen. A Web API has been developed on the Landscaper through which the log entries can be extracted by any external system that intends to use this data. A Dashboard/GUI can access this data and display the events as they occur.

Endpoint::

```
GET /landscaper_log
```

Example CURL calls:

The following call returns the latest 50 entries (by default) from the log:

```
curl http://landscaper_web:9001/landscaper_log
```

The output can be further filtered by a date/time by providing "after_timestamp" as follows:

```
curl http://landscaper_web:9001/landscaper_log?after_timestamp=2019-05-16T15:59:34
```

Output :

The output returned is of content type JSON and a sample is shown below:

```
[
        {
          "log_text": "Adding physical machines to the landscape... ",
          "log_time": "2019-05-14T10:55:47",
          "log_type": "INFO"
    },
    {
          "log_text": "HWLocCollector - Adding machine: NODE39 ",
          "log_time": "2019-05-14T10:55:47",
          "log_type": "INFO"
    },
    {
```

```
            "log_text": "Finished adding physical machines to the landscape. ",
            "log_time": "2019-05-14T09:17:05",
            "log_type": "INFO"
        },
        {
            "log_text": "Subscribing to CIMI events ",
            "log_time": "2019-05-14T09:17:05",
            "log_type": "INFO"
        }
]
```


Figure 26. Event Status output (JSON)

7.4. Violations

The GUI contains a section to check the SLA violations that have occurred during a service execution (see D4.4). This information can only be obtained on the leader agent: SLA assessment is performed on the leader, and accessing the results is only guaranteed from it. Anyway, it is possible to show the results related to services executed on a non-leader agent requesting the information from the leader agent.

The SLA information is shown on two tables:

- A table that summarizes the active services and the number of violations on each of them;
- A detailed list of occurred violations in a given timeframe (e.g. last 24 hours).

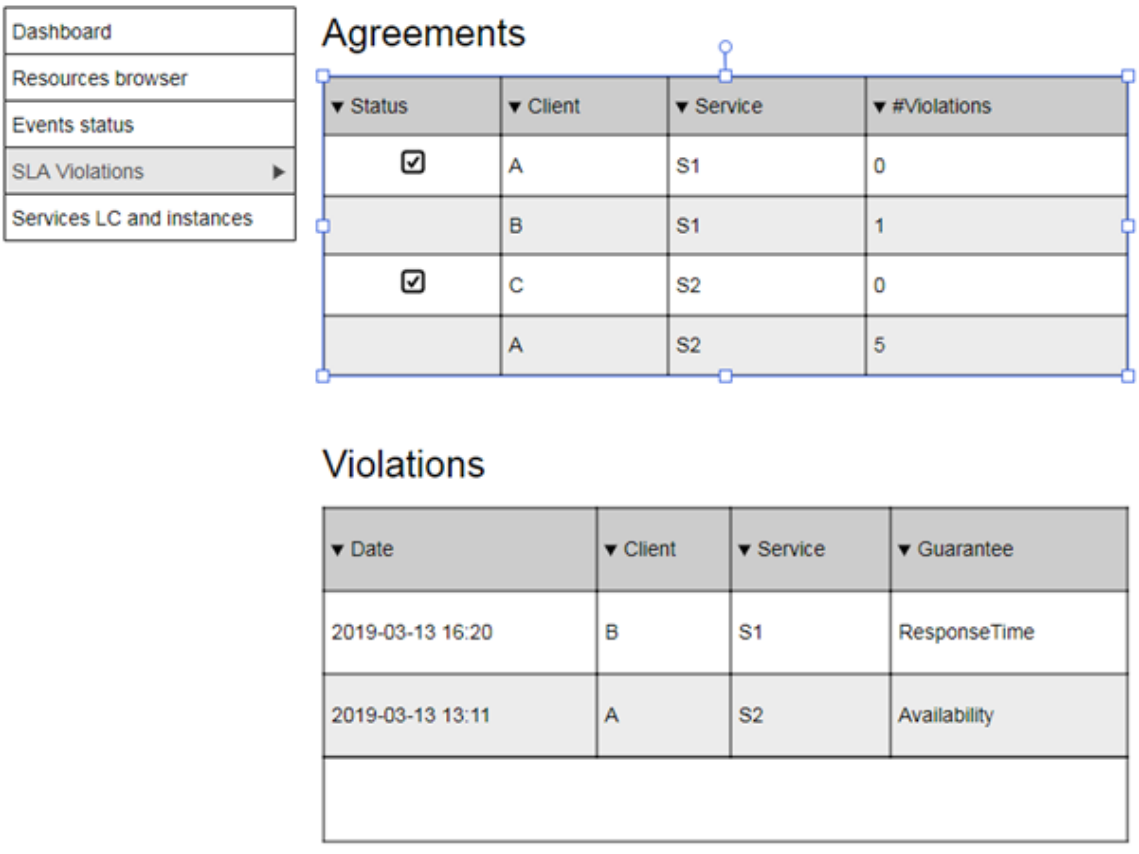

Figure 27. SLA Violations mock-up

7.5. User Management

The GUI includes a section where the user can define the user-profile and sharing-model properties of the device. These properties define the role of a specific device in mF2C.

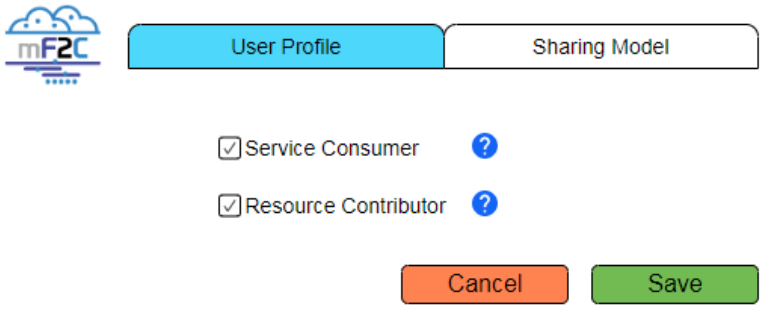

Figure 28. User profile mock-up

The device can act as a service consumer and / or as a resource contributor. In the first case, the device will be able to launch services in a mF2C cluster. In the second case, the device will share its resources to other services launched by other devices. In this case, the user defines the values or properties that will be shared:

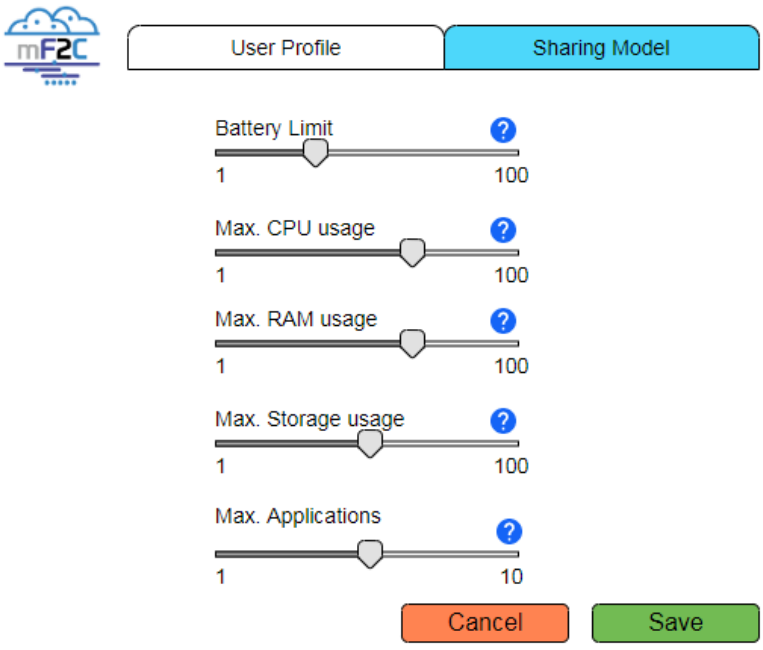

Figure 29. Sharing model mock-up

7.6. Launching services and monitoring service instances

The part of the GUI related to the service registration and service catalog, explained in D2.7 [1], now would include the part related to SLA templates. When the user registers a new service, there would be a new option to add different SLA templates to the service, while the rest of the fields remain the same, as can be seen in Figure 30.

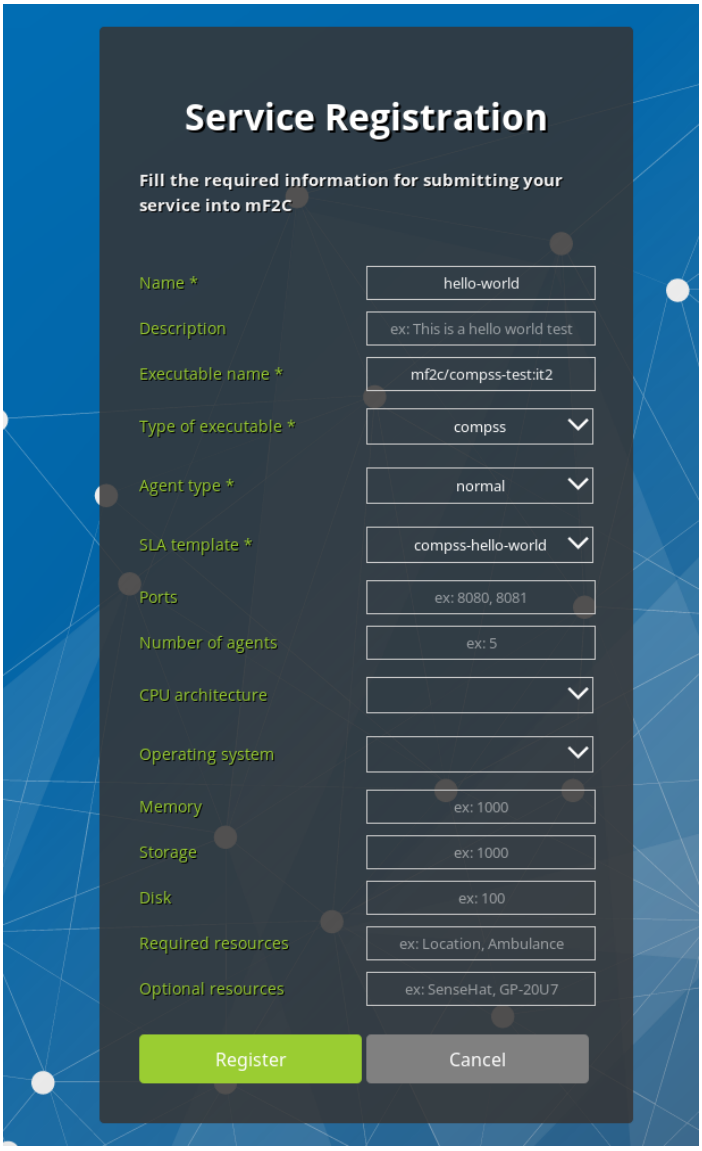

Figure 30. Service registration GUI

Once the service is registered into the system, the user can select from the catalog the service to launch. It is to be noted that differently from IT-1 where services could only be added from the cloud agent, in IT-2, they could be added from any agent in the system. Then, when the user selects the service to launch, now a new option appears in order to select the SLA template to use for that specific service, as can be seen below in Figure 31. Once the SLA is selected, the service can be launched creating a new service instance in the system.

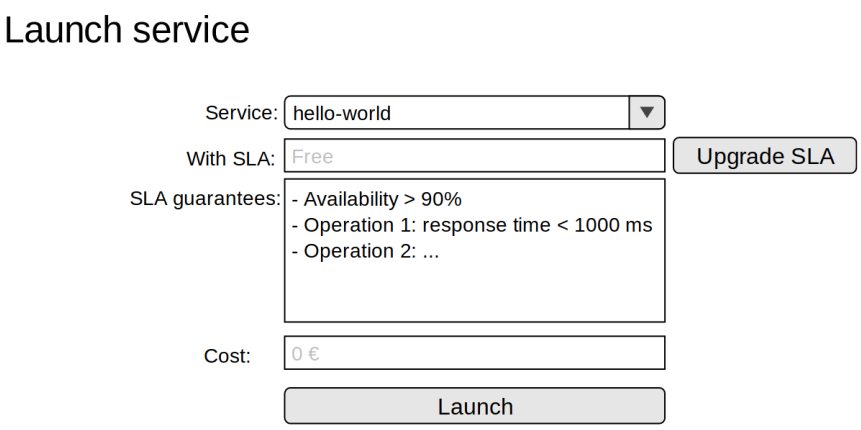

Figure 31. SLA template

7.6.1. Manage a service instance

Once a service has been launched in mF2C, the device's user can use the GUI to track the status of the resulting service instance. A form is defined that provides a way to stop, restart and terminate these services:

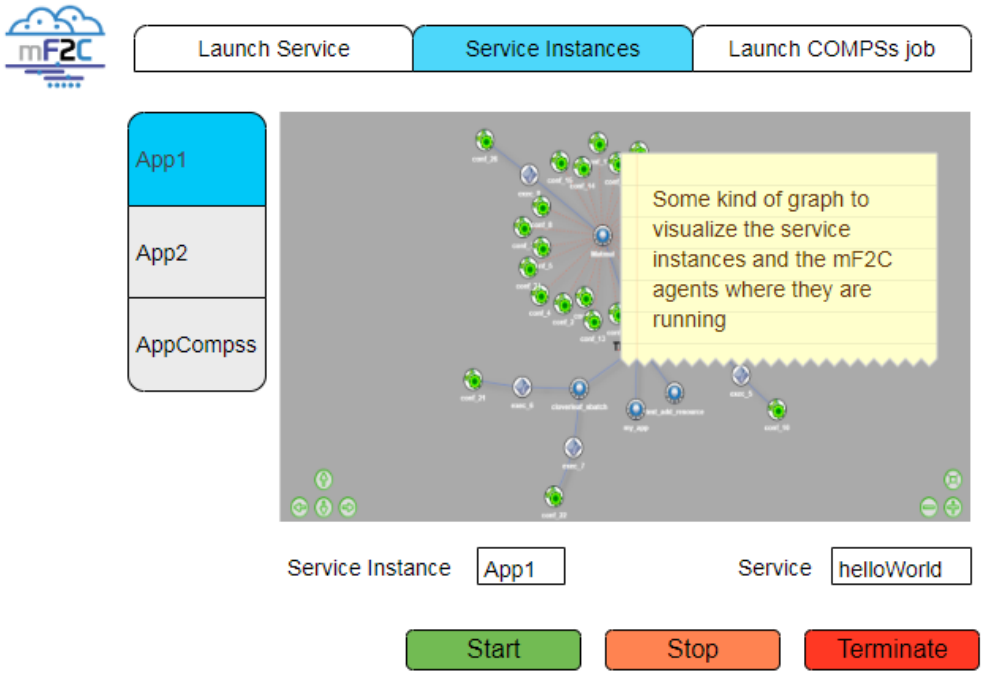

Figure 32. Services Instances GUI

First, this form should show a list of the service instances running in the device, and the service instances launched by this device. Then, if the user selects one of these service instances, the GUI should show the information about all the devices running the selected service instance. This information can be a graph or a tree representing these devices' cluster. Finally, the GUI gives the option of stopping, restarting and terminating these service instances.

7.6.2. Launch COMPs job

COMPSs jobs require an additional step to be executed. After launching a **COMPSs service**, the user can launch jobs in them through the following GUI:

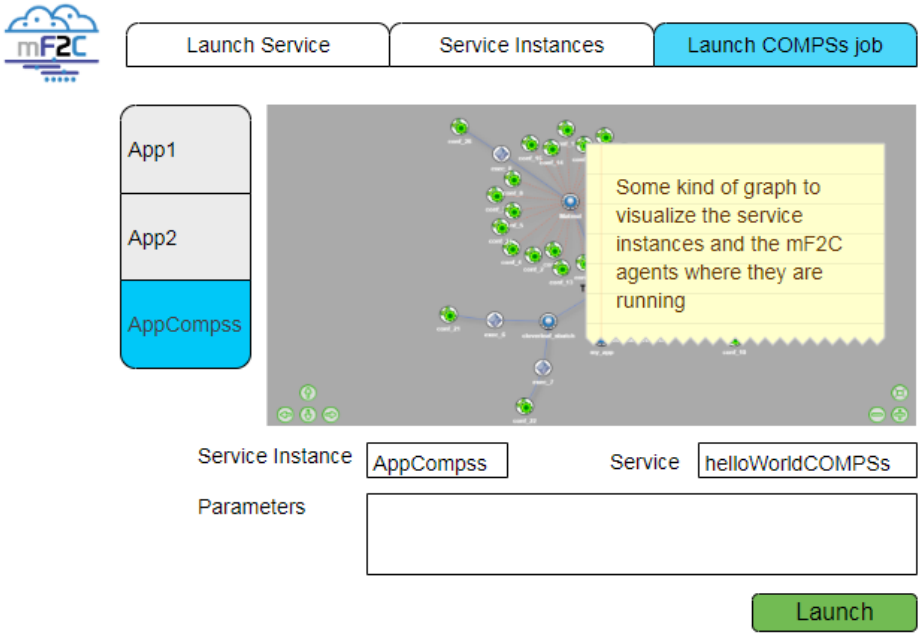

Figure 33. Launch COMPSs job GUI

There would be a form where the user could define the parameters of the job to be executed in the Distribution Execution Runtime (COMPSs).

8. Conclusions

First of all, in this deliverable we have reviewed the main characteristics of the final mF2C architecture proposed in deliverable D2.7 [1], and highlighted the differences with the preliminary proposed architecture in D2.6 [2]; specially justifying which blocks from the Agent Controller have been removed because they are obsolete, or have been moved to the Platform Manager, or have been merged, or finally have been considered transversal blocks.

The main rationale for moving the Service Management from the AC to the PM is because the PM provides high-level functionalities, including the intelligence to take decisions with a more global view. Whereas, the AC has a local scope, that is, taking decisions based on this local scope; and the Service Management tasks, as global tasks, are not only related to a particular device. Thus, now the main AC components are the Resource Management and the User Management. The list of the Resource Management functions is:

- Discovery
- Identification and Naming
- Categorization
- Policies

Two of the subcomponents in the IT-1 Resource Management have been removed; the Monitoring has been merged with the Categorization module, and the Data Management, apart from being merged with the Data Management in the Platform Management, is now considered a transversal block, used by both the Agent Controller and the Platform Manager.

The Agent Controller is also responsible for managing the users related to the device where the mF2C agent is installed. The three functionalities of the AC related to the user management are:

- Profiling
- Sharing model
- Assessment

Here the main differences with the design in IT-1 are two-fold:

- The QoS enforcement has been moved to the Service Management in the Platform Manager since we decided to put all QoS related stuff into a single block.
- And we have added a new block, not present in IT-1, which is the Assessment, responsible for checking the user-profile and sharing-model properties

In this deliverable, we have also described the design of three of the five transversal blocks in the mF2C agent:

- The Event Manager, a new component in IT-2, allows for any of the modules in the mF2C agent to subscribe to the events occurring throughout the system; and for the events to be advertised.
- The Dashboard/GUI, which is an update of the previous dashboard in IT-1, is enriched with new functionalities such as the monitoring of the mF2C system and is installed in every mF2C agent.
- The Security block that for IT-2 includes the enhanced CAU client which serves as an Agent's local gateway to the CAU middleware, the Reverse Proxy and the refactored AC library.

For each one of the components of the Resource Management and the User Management we have provided different workflows illustrating the main functionalities of these components, and trying to ease the later implementation. In almost all cases, these workflows are updates of the previous ones in IT-1, and they have been modified and improved, on one hand because we postponed some of the

functionalities to IT-2, and on the other hand due to the feedback received from the implementation in IT-1. Also for some of the security functionalities, Agent Authentication, we have provided the corresponding workflow; as well as for the Event Manager.

Finally, for the Dashboard/GUI we have provided the mock-ups of the landing page, and also for the different pages in the menu: CIMI Resource browser, Events Status, SLA Violations, User management and Service Launching Control (LC) and instances. These mock-ups will set the basis for future development and implementation.

As a final conclusion, we want to highlight that this deliverable has described the final and definitive design of the Agent Controller (AC) in the mF2C agent, to be implemented in IT-2. This design, and the functionalities we describe for each component, are definitive, unless final minor refinements will be included in the design after the feedback received from the implantation and integration phases.

References

[1] "(IT-2) D2.7 mF2C Architecture," [Online]. Available: http://www.mf2c-project.eu/2-7-mf2c-architecture-it-2/.

[2] "(IT-1) D2.6mF2C Architecture," [Online]. Available: https://www.mf2c-project.eu/wp-content/uploads/2017/06/mF2C-D2.6-mF2C-Architecture-IT-1.pdf.

[3] "(IT-1) D3.3 Design of the mF2C Controller Block," 2017. [Online]. Available: https://www.mf2c-project.eu/wp-content/uploads/2017/09/D3.3-final.pdf.

[4] "(IT-2) D3.2 Security and Privacy Aspects for Agent Controller," [Online]. Available: https://www.mf2c-project.eu/wp-content/uploads/2019/02/mF2C-D3.2-Security-and-Privacy-for-Agent-Controller.pdf.

[5] "(IT-1) D3.5 mF2C Agent Controller Block integration," [Online]. Available: https://www.mf2c-project.eu/wp-content/uploads/2017/12/D3.5-mF2C-Agent-Controller-Block-integration-IT-1.pdf.

[6] X. M.-B. a. E. M.-T. Z. Rejiba, "A Beacon-assisted direction-aware scanning scheme for 802.11-based discovery in Fog-to-Cloud systems," in *Proceedings of the 2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2018.

[7] X. M.-B. a. E. M.-T. Z. Rejiba, "Towards a context-aware Wi-Fi-based Fog Node discovery scheme using cellular footprints," in *Proceedings of the 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2018.

[8] "(IT-1) D3.1 Security and privacy aspects for the mF2C Controller Block," [Online]. Available: https://www.mf2c-project.eu/wp-content/uploads/2017/06/mF2C-D3.1-Security-and-privacy-aspects-for-the-mF2C-Controller-Block-IT-1.pdf.

[9] "(IT-1) D5.1 mF2C reference architecture (integration)," [Online]. Available: https://www.mf2c-project.eu/wp-content/uploads/2018/06/D5.1-mF2C-reference-architecture-integration-IT-1.pdf.

[10] mF2C. [Online].