Target Outcome: b) Sustainable technologies for a Smarter Society

(FP7 614100)

D7.3.2 Final Design and Implementation of the Configuration and Composition Manager

Date 07 January 2016 – Version 1.0

Published by the Impress Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme and the Conselho Nacional de Desenvolvimento Científico e Tecnológico
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation

Document control page

Document file: D7.3.2 Final Design and Implementation of the Configuration and Composition Manager_v02.docx
Document version: 1.0
Document owner: Enrico Ferrera (ISMB)

Work package: WP7 – IDE Framework for Model-driven development
Task: T7.3 Unified configuration management API
Deliverable type: P

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Enrico Ferrera	2015/12/01	Initial structure of the document. Contributions in all sections.
0.2	Davide Conzon	2015/12/15	Contribution on Configuration and Composition Framework Architecture chapter
0.3	Enrico Ferrera	2015/12/24	Document finalization

Internal review history:

Reviewed by	Date	Summary of comments
Lucas Lira Gomes	2015/12/29	Accepted with minor revisions
Marc Jentsch	2016/01/07	Accepted with minor revisions
Peeter Kool	2016/01/07	Accepted with minor revisions

Index:

1 Executive summary

This document describes the final architecture of the Configuration and Composition Framework (CCF). Furthermore, APIs for interaction with CCF are described.

The deliverable is organized as follow: section 2 introduce briefly the contents of the deliverable, section 3 describes two protocols related with the CCF: the Simple Network Management Protocol (SNMP) [1], which has inspired the architecture of the CCF and the eXtensible Messaging and Presence Protocol (XMPP[1]), which has been used to implement the framework. Section 4 presents the final architecture, specifying how the components of the framework have been designed and implemented. Finally, in Section 5, the operations exposed by the framework using XMPP are described.

This deliverable presents the final architecture and the implementation details of the CCF, for the description of the role of the CCF in the IMPReSS architecture and its basic concepts, please refer to the Deliverable *D7.3.1 Initial Design and Implementation of the Configuration and Composition Manager*.

[1] http://xmpp.org

2 Introduction

The CCF is the part of the IMPReSS SDP in charge of the configuration and composition of the different components of the platform in order to initialize them and establish how they have to work together so as to set the correct workflow of the platform.

The framework is composed basically by three different components:

- The Configuration and Composition Manager (CC_M), which is the central component, in charge to compose and configure the different components of the platform, exposing the IMPReSS APIs for this purpose.
- The Configuration and Composition Agents (CC_A), which are the distributed entities of the architecture, located on the components that monitor the associated component and interact with the Manager.
- The Composition GUI, which is the interface provided to the System Managers to interact with the Configuration and Composition Manager, using the API that it exports.

The architecture of the framework has been inspired by the one of the Simple Network Management Protocol (SNMP), the standard-de-facto for network management in IP networks. The solution presented in this deliverable addresses some of the issues of the SNMP protocol when used in the IoT scenario (security issues, request to implement a specific protocol only for network management), leveraging the XMPP, which is an open standard already used in IoT applications that can be used also for these purposes.

The following of the document provides an overview of the final design and implementation of the Configuration and Composition Framework.

3 Background

This section introduces SNMP and XMPP, two standards that has been used in different ways in the design of the CCF architecture. Before, the section introduces SNMP, presenting its architecture and explaining how it has influenced the one of the CCF; then it explains why, for the actual implementation of the CFF, the XMPP protocol has been used.

3.1 SNMP

SNMP is an Internet-standard protocol, standardized by the IETF, for managing devices on IP networks. This protocol is used to monitor the status of devices connected to a network. An SNMP managed network consists of three basic components: a set of managed devices, a set of Agents (one for each managed device), and a Network Management Station (NMS). The protocol is based on a distributed architecture: in the NMS, there are the Managers, which are the components responsible to monitor the devices. Each monitored system is controlled by an agent, which exposes the management data as variables that can be queried and written, in order to configure the device. Besides allowing the management of the variables, the SNMP protocol implements also, a notification system called trap, which allows the agents to notify the SNMP managers, when an important event happens on the device, like a malfunction.

Designing the architecture of CCF, the one of SNMP has been used as reference. Particularly, CCF uses the same structure with a central component for the management, which communicates with a set of agents, distributed on the managed entities. Furthermore, also the possibility for the agents to generate asynchronous notification to the manager has been replicated also in the CCF architecture.

The next paragraph explains the motivations, which has led to implement this architecture leveraging the XMPP protocol.

3.2 XMPP

SNMP is a standard de-facto protocol for management of devices in the IP based networks. In IoT the problems to address are similar, but the challenge is to manage typical components of an IoT network, i.e. IoT gateways, sensors and actuators. MQTT [2], RESTFul protocols like CoAP [3], HTTP [4] and XMPP are protocols already used in IoT application deployment and therefore were considered conceivable to use them also for the composition and configuration purposes. The object identification approach used in SNMP is quite hard to deal with, furthermore, much of the system is insecure, and the SNMP traps are not simple to manage. For these reasons, the CCF has been designed to use the same architecture of SNMP, but leveraging the XMPP protocol and its features. XMPP – an IETF standard also known as Jabber – is a protocol based on XML for the real-time messaging, for the exchange of presence information and for request-response services. XMPP supports a wide range of applications: instant messaging, presence notification, multi-party chat, audio-video call and, most generally, XML routing. XMPP is an open protocol and, thus, is free and open-source: over the long period, an open standard provides stronger security, greater extensibility, and is more open to improvement than proprietary technologies. XMPP is, after more than 10 years of development, proven and mature; it has been tested in scenario with thousands of Jabber servers on internet and millions users (e.g. it has been at the basis of Google Talk) and a large number of applications have been developed using the instant-messaging functionalities, in very different application fields. XMPP is fully decentralized: everyone can use its XMPP server and manages independently its network. XMPP is extensible: using the potential of the XML, everyone can add features to the core functions. It offers interoperability features, such as HTTP binding, service discovery, file transfer, server federation. Finally, XMPP natively provides security features, such as Simple Authentication and Security Layer (SASL) [5] and Transport Layer Security (TLS) [6], both for client-to-server and server-to-server communications.

Keeping the same architecture of SNMP, but leveraging on an XMPP server, can be sent out jabber notifications, kicked off restart jobs and, managed seamlessly device availability and changing status, updated web pages, etc. The XML data are small in this case, and one XMPP server can be used both to talk to humans in message stanzas, or to computers, using the same protocol. This is

particularly important useful to save resources in resource-constrained devices, avoiding requiring the support for additional protocols like SNMP or CMIP. IoT-PIC leverages many features provided by the protocol: every device is univocally identified through a Jabber Identifier (JID); and the presence mechanism is used to know in real-time the status of the devices. Furthermore, also XMPP Extensions (XEPs)[2] are used. The XEP-0030 (Service Discovery)[3], used to discover what entities are on the network and, exactly, which XMPP features those entities implement. The XEP-0050 (Ad-Hoc Command)[4], which provides workflow capabilities for any structured interaction between two XMPP entities. The XEP-0060 (Publish-Subscribe)[5] that allows to subscribe to an information node and then to receive a notification, only when an entity publishes an item to that node, providing a scalable and real-time alternative to constant and expensive polling for updates. Finally, also the XEP-0248 (PubSub Collection Nodes)[6] is used to organize the publish-subscribe nodes leveraged in the discovery mechanism in a hierarchical structure. Indeed, this XEP explains how to create nodes, which can contains one or more other nodes (both leaf nodes and other collection nodes); the subscription to one of these nodes allows receiving the notifications of all the events sent to the publish-subscribe nodes that it contains, therefore implementing an actual network management.

[2] http://xmpp.org/xmpp-protocols/xmpp-extensions/
[3] http://xmpp.org/extensions/xep-0030.html
[4] http://xmpp.org/extensions/xep-0050.html
[5] http://xmpp.org/extensions/xep-0060.html
[6] http://xmpp.org/extensions/xep-0248.html

4 Configuration and Composition Framework Architecture

The deliverable D7.3.1 Initial Design and Implementation of the Configuration and Composition Manager has introduced the context of the CCF and the initial design of its architecture. This section describes the final architecture and the implementation details of the framework's components.

4.1 Architecture description

Figure 1 shows the IMPReSS platform architecture from a functional point of view. Besides the ones already listed (CC_M, CC_A and Composition GUI) the components of the platforms, related with the CCF are the Resource Adaption Interface (RAI) and the Resource Catalogue (please, refer to D3.1 Resource Adaption Interface Framework for details about RAI and to D3.2 Resource and Service Discovery Solutions for details about Resource Catalogue). These components collaborate to achieve two main goals: the first one is to allow interconnecting the modules of an instance of the IMPReSS platform, in order to connect them and to realize desired applications and services. The second one is to allow the configuration of the platform and its components, furthermore, enabling to monitor their status in real-time.

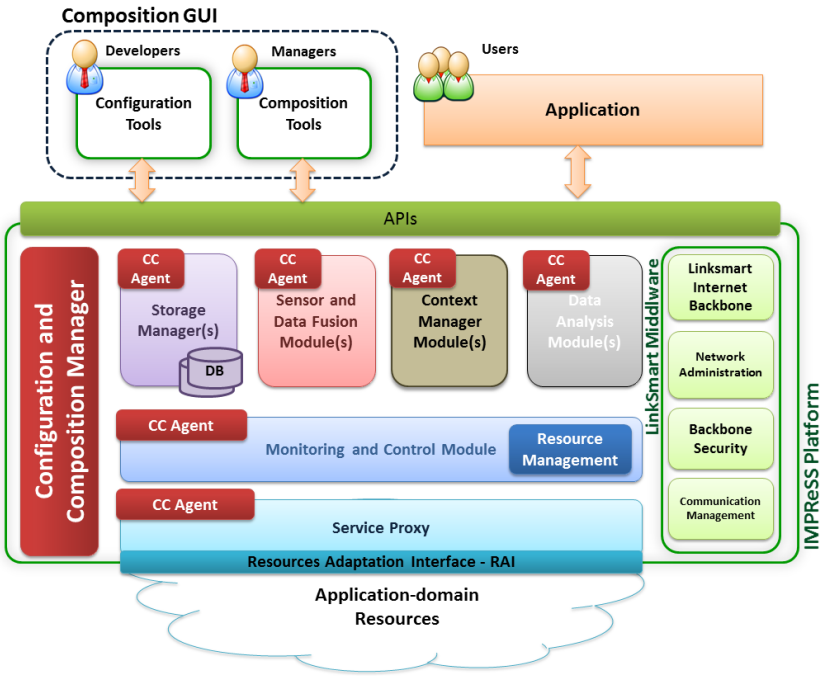

Figure 1 - IMPReSS SDP

The stakeholders execute the commissioning tasks using the Composition GUI, which relies on the CCM. The RAI, instead, is responsible of the virtualization of the devices connected to the network. The stakeholders mostly involved with these tasks are the Developers and the Managers. The developers use these tools to combine different modules and compose the specific logic flow suitable for their applications. In this way, actually, they implement the final "IMPReSS-enabled" application, leveraging the SDP. Instead, the system Managers set the parameters of the platform modules to make the system effective. They install, configure, deploy the applications, and connect them to other external services and hardware components. Managers must have a specific interface (GUIs actually, in different flavors, such as Web-based and smartphone/tablet apps), so that they are easily able to operate on the system under different circumstances into different environments. To fit both needs (Developers' and Managers' ones), the CC_M allows dealing with the main aspects of commissioning and platform configuration:

IoT Platform sub-components composition: i.e. interconnect different available components of the platform (e.g. service proxies, data filtering and aggregation modules, decision support systems, etc.). In other word, the composition aims to realize the application, defining, for each relevant platform component available, from which other components it has to take the inputs and to give its outputs. This stage defines the workflow of the application. This feature is used by the platform Developers for defining connections among different sub-components in order to implement specific application logic. In fact, through the composition is possible to realize the actual application to be executed. For instance, in order to make a building management application, the developer can use the GUI in order to graphically connect the outputs of the logic blocks representing physical temperature sensors with the inputs of a module calculating mean values of incoming data series. The output of this last module can be connected with the input of another module that checks if incoming values are above or below a specific threshold. The output of this last module can be used as input for the software module that drives a bell for announcing a critical situation. CC_M has the role of concretely implement the logical connection sketched on the GUI.

IoT Platform sub-components Configuration: this stage provides to each platform component involved in the realization of the application (i.e. the ones interconnected through the composition stage) the values for the correct behavior of the applications. For instance, suppose we have interconnected, through the composition stage, the output of a temperature sensor to a module that raises an alert whenever the temperature exceeds a threshold. In this case, the configuration stage is responsible for set parameters, such as the sensing rate of the sensor temperature and the threshold temperature at which the second module has to raise the alert. The CC_M provides to the platform Manager all the available services and entities, allowing to configure the parameters of the components of the overall IoT platform.

IoT Platform sub-components Discovery: it allows detecting automatically devices joining the IoT platform and the services they provide. A common language has to be used to describe the services, in order to allow their usage without the need of users' intervention.

The Composition GUI is a model-driven development toolkit that allows inexperienced developers to discover and compose distributed devices and services into mashups. The proposed modelling tool allows operators to model the integration of IoT components visually and programmatically, transforming the model into actual source code, executable as a standalone application, with software interfaces selectable during prototype modelling. This interface has been designed to allow users to configure, compose and manage entities to provide different services for the Internet-of-Things by a single access point. Through its interface, users are able to compose the IoT platform, installing and configuring, among those available only the services required for their own purposes.

4.2 Configuration and Composition Framework Components

The role of the CCF is to provide a unique and general way of performing the commissioning of the platform. The architecture of the CCF is shown in Figure 2. This architecture, as already said, is inspired by the SNMP one (described in the previous section) and aims at performing the configuration and composition of hardware and software resources. The architecture of CCF consists of two levels, the global and local one, and is mainly composed by two components:

- A Configuration and Composition Manager (CC_M) at a global level.
- A Configuration and Composition Agent (CC_A) at local level.

The communication among the components leverages the XMPP protocol.

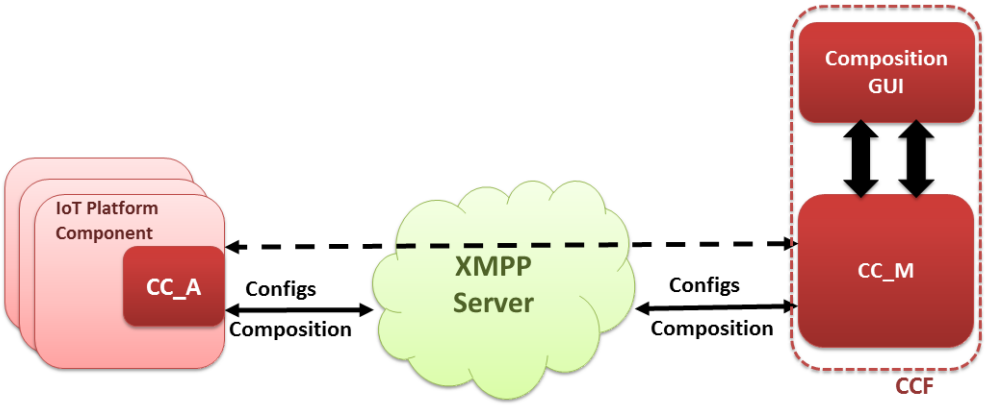

Figure 2 - Configuration and Composition Framework architecture

The CC_M is the module in charge of managing the configuration and composition processes of the other modules into the platform; it works as an interface between the applications and the various components of the platform. The functionalities of CC_M consist in the following:

- Notifies the applications on the status of the components available in the middleware.
- Retrieves the configuration from the CC_A, when required through a XMPP ad-hoc command.
- Updates the configuration of the components through the CC_A via XMPP.

CC_M is responsible for the management of the composition stage. In order to do this, the CCF leverages on the publish-subscribe paradigm, which allows the complete decoupling of the various components. Specifically, when the output of a component is connected to the output of another component, it means that the last one will subscribe itself to the publish-subscribe node, where the first one publishes its data.

A CC_A is associated with each component of the platform. It exposes "get" and "set" ad-hoc commands, to manage configuration parameters of a specific component to the CC_M. The CC_A operates actually the configuration commands coordinated by CC_M. The association of an agent to each module makes the system more expandable and scalable from the point of view of configuration issues. CC_A is responsible for:

- Register the component in the CC_M.
- Handling the configuration parameters of the component.
- Handling the interconnection of the components with each other, adding and removing input Configuration and Composition Framework API

In this chapter are reported the operations that can be managed by the REST API exposed by the CCM and CCA.

4.2.1 Composition operations

To allow the composition of the applications, the CCF implements a set of features for the service discovery, implemented through the XMPP protocol. These features allow, on one hand, to register automatically the new devices connected to the network, describing them and their functionalities, with a common format; and, on the other hand, to allow their discovering. Particularly, in the proposed solution, every resource discovered, is associated with an account on the local XMPP server. When a new resource is connected to the network, the manager of that network calls an ad-hoc command on CC_M, This command creates one or more publish-subscribe nodes, representing the resource and its features. Following a concept similar to the one defined by the OSGi Device Abstraction Layer standard specification[7], the devices are described through the functions they

[7] https://github.com/osgi/design/raw/master/rfcs/rfc0196/rfc-0196-DeviceAbstractionLayer.pdf

provide and the operations possible on them. Specifically, the CC_M creates a collection node with the name of the id of the resource and, then, inserts in this collection node one leaf node for each function provided by the resource. For example, if the resource is a sensor that measures humidity and temperature, the CC_M creates a collection node with the id of the device, containing two nodes, one for the function temperature and one for the function humidity. Using the features defined in the XEP-0030[8], the CC_M associates to each node the information useful for the service discovery. The resource nodes have associated the list of resource types (for the sensor taken as example, the list will contain Humidity Sensor and Temperature Sensor). Instead, the function nodes have associated the list of operations possible for that function (i.e. getTemprature for the temperature function and getHumdidity for the humidity one). This is the lowest part of the hierarchy; the Context Manager can create nodes related to the location where the nodes of the devices can be inserted to set their location. Finally, all the nodes have to be inserted in one source node, in order to allow browsing the tree starting from the root. Besides the nodes, the CC_M publishes also a set of ad-hoc commands callable on the resource: this set of commands maps the list of operations registered in the Service Discovery nodes. Accordingly, when a user discovers the operations on a resource, he/she knows that he/she can use that name to call a command on the resource. In this way, using the service discovery provided by XMPP, it will be possible to search for a resource node on the server and, through its name, it will be possible to retrieve the commands that it exports. Particularly, the CCF provides and ad-hoc command, which allows discovering the resources; if the user does not indicate parameters, the entire hierarchy of nodes is returned. Otherwise, if the user needs to limit the discovery, it can use this format:

{"nodes": [], "types": [], "devices": [], "functions": [], "operations": [] }

Where, the different fields are used in this way:

- "nodes": if a list of nodes is indicated in this part, the result indicates only the nodes contained in these ones.
- "types": if a list of types is indicated in this part, the result indicates only the devices of these types.
- "devices": if a list of devices is indicated in this part, the result indicates only functions and operations of these devices.
- "functions" : if a list of functions is indicated in this part, the result indicates only devices that support these functions.
- "operations": if a list of operations is indicated in this part, the result indicates only devices that provide these operations.

The CC_M will maintain the tree synchronized with the presence of the resources, when a resource disappears, the manager associated to its network calls an ad-hoc command on the CC_M, which deletes the collection node of the device and its children.

4.2.2 Configuration operations

For the configuration part, specifically, every CC_A exposes two ad-hoc commands: the first command provides a list of all the management data, into a XML structure, which associates to every variable: the type, the current value and a list of possible values to assign (if range is limited). The second command allows updating the values associated to one or more of these variables (if they are writable); to write the values, the ad-hoc command has to be called, passing to it the XML used in the reading command with new values. The application, which has to configure the various components, interacts with the CC_M that provides two ad-hoc commands to read and write the configurations on the various CC_A.

[8] http://xmpp.org/extensions/xep-0030.html

4.3 Composition and Configuration Manager Ad-Hoc commands

Name	Parameters	Returns	Description
getAvailableDrivers	-	The list of the drivers available on the repository	Returns the list of the drivers available on the repository and not already installed in the RAI.
getConfiguration	Component identification name	The parameters that can be configured in the component.	This operation is used to get the list of the parameters that can be configured in the component (with the current values).
setConfiguration	Component identification name ConfigurationForm		Updates the configuration of the component, setting the values passed as parameter.

4.4 Configuration and Composition GUI

Configuration and Composition GUI is composed by two parts, the first one allows the system integrators to manage and monitor an instance of the IMPReSS platform, the other one allows composing the module of an instance.

The configuration interface (Figure 3) allows a number of features that simplify commissioning and management of the implemented IoT platform. The features include:

- Connection management, for the XMPP features.
- Management of system bundles (RAI, Managers of the IMPReSS SDP).
- Download of desired system bundles.
- Installation and removal of system bundles.
- Start and stop of system bundles.
- Management of system bundles updates.

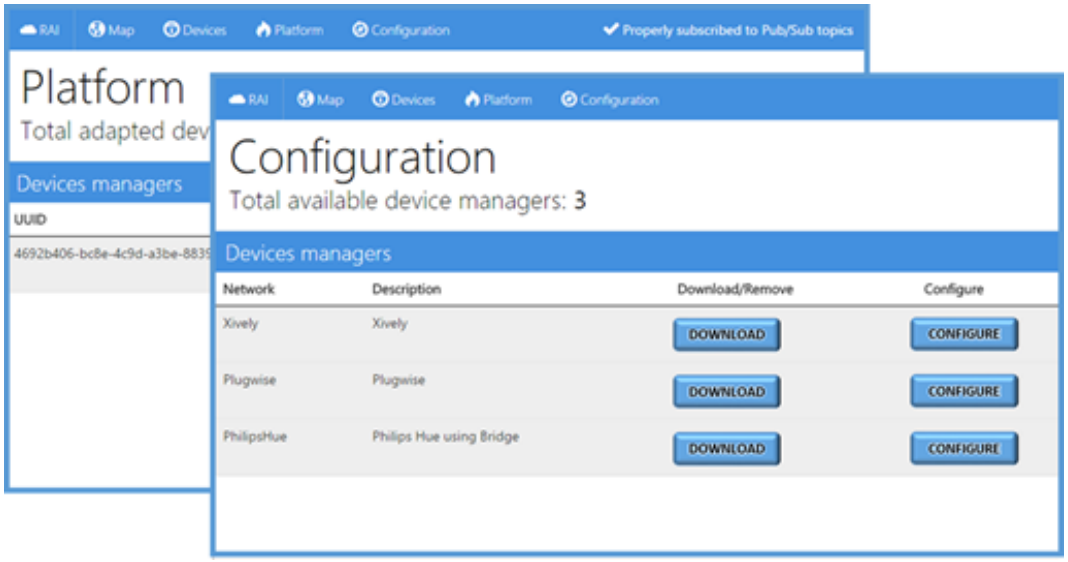

Figure 3 - Composition GUI

An installation wizard procedure has been designed to guide the user into the different configuration tasks. Once launched, the Composition GUI reads its Platform Configuration File loading the values, including the web link to download an xml file containing the updated list of the available system bundles. Using these data, users can fill a form and save their personal settings, related to XMPP server configurations, as Internet Protocol (IP) address, hostname, listening port and pub/sub node. After saving above information, the IOT-PIC GUI tries to connect to the local XMPP server. If the connection fails, the Composition GUI displays an error message and let user to try to establish the connection again. Once the connection is established, users can access all the functionalities of the interface. This interface is dynamically built using the list of available bundles downloaded from the web link. The GUI provides, for each bundle, the following operations:

- Download/remove: when the user clicks the download button, the CC_M download the bundle from the remote repository and install it in the middleware instance. Once the bundle is installed, if no more needed, it can be removed.
- Update: when a new version of an installed bundle is available on the repository, the GUI alerts the user. Once the Update button is clicked, the software automatically uninstalls the previous module and replaces it with the new one.
- Start/stop: this operation allows starting and stopping the execution of the bundles installed.

The interface provides also a visual indication of the status of the bundles, in order to inform in real-time the user if the bundle is correctly running, or if there is some error in its execution.

Besides the page for managing the system bundles, the Composition GUI provides also an administrative page used by system Managers to install and configure the RAI Device Managers. The two views represent different levels of management, for this reason, the first page is only accessible for the system administrator, while this latter one is accessible to all the users of the platform. For this page, the web interface uses the CC_M to retrieve the list of Device Managers available on the repository, and using this list, it dynamically creates the web page, shown in Figure 3. Through this web page, it is possible to interact with the CC_M, in order to indicate the Device Managers to install (or remove) in the RAI. Furthermore, the interface can be used to configure one Device Manager: when the user clicks the configure button, the GUI queries the CC_M, to retrieve the configuration parameters for the corresponding bundle (e.g. data related to sensors, addresses, communication protocol, thresholds, or sampling rate). The GUI uses this information to build a form, which has to be filled by the user to indicate the value to set for each parameter. Once saved the values set are set in the bundle, through the CC_M.

As said before, the IMPReSS toolkit is completed by a model driven development (MDD) tool for IoT applications. This tool uses the features provided by the CCF, to manage the components and to create the interactions among them, to implement applications, based on the IMPReSS platform.

The tool provides a list of the components available in the platform; this list is maintained updated in real-time, through the CCF. Furthermore, when the application is created using the MDD tool, the configuration generated contains information useful to connect components among each other, as indicated by the links created in the application model.

5 Summary & Conclusion

In this deliverable, the final Composition and Configuration Framework (CCF) architecture and features have been outlined. It manages the provisioning of the platform performing two different stages: the composition of the application and its configuration.

This document presented the final design and implementation of the CCF, explaining how the different component work and how the CCF interacts with the rest of the SDP and how it can be used by the stakeholders of the platform.

6 Bibliography

[1] Case JD, Fedor M, Schoffstall ML & Davin J. (1990) Simple Network Management Protocol (SNMP).

[2] IBM & Eurotech. (2010) MQTT V3.1 Protocol Specification. (online) http://public.dhe.ibm.com/software/dw/webservices/wsmqtt/MQTT_V3.1_Protocol_Specific.pdf.

[3] Shelby Z, Hartke K & Bormann C. (2013) Constrained Application Protocol (CoAP) draft-ietf-core-coap-18, RFC 7252. (online) http://datatracker.ietf.org/doc/draft-ietf-core-coap/.

[4] Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P & BernersLee T. (1999) Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, (online) https://www.ietf.org/rfc/rfc2616.txt.

[5]] Myers J. (1997) Simple Authentication and Security Layer (SASL) (online) https://www.ietf.org/rfc/rfc2222.txt.

[6] Dierks T., Allen C. (1999) The TLS Protocol Version 1.0 (online) https://www.ietf.org/rfc/rfc2246.txt.