XMPP-based Network Management infrastructure for agile IoT application deployment and configuration

Enrico Ferrera[1], Davide Conzon[2], Paolo Brizzi[3], Lucas L. Gomes[4], Marc Jentsch[5], Peeter Kool[6]

[1,2,3]Istituto Superiore Mario Boella (ISMB), Torino, Italy
[4]Federal University of Pernambuco (UFPE), Recife, Brazil
[5]Fraunhofer FIT, Schloss Birlinghoven, St. Augustin, Germany
[6]CNet Svenska AB, Stockholm, Sweden
e-mail: [1,2,3]{ferrera, conzon, brizzi}@ismb.it, [4]lucas.gomes@gprt.ufpe.br, [5]Marc.Jentsch@fit.fraunhofer.de, [6]peeter.kool@cnet.se

***Abstract*—The computing technology ecosystem is today facing with the Internet of Things (IoT) innovation, driven by radical technological and methodological changes, which include communications paradigms among devices as well as instruments to enable the creation of added-value services on top of them. Although a number of platforms are today available to address the IoT needs at different level, few of them tackle the issue of rapid deployment and simple configuration. This paper focuses on those needs, providing the description of the Software Development Platform (SDP) developed within the IMPReSS EU project, particularly focusing on the Network Management infrastructure and the commissioning infrastructure. The latter supports the configuration and mashup of the different components of the IoT platform, while the Network Management infrastructure provides instruments for network monitoring and setup, based on the use of the eXtensible Messaging and Presence Protocol (XMPP). Furthermore, the paper describes other components of the IMPReSS SDP, which interact with the Internet-of-Things Platform's Infrastructure for Configurations (IoT-PIC): the Resource Adaptation Interface (RAI), which virtualizes the devices connected to the network, and the framework for Model-Driven application design, which has been developed to support actual developers in the discovering and composition phases of IoT services design.**

Keywords— Internet-of-Things; IMPReSS; Network Management infrastructure; Commissioning Interface; Model Driven Development; XMPP

I. INTRODUCTION

In the recent past, many researches have been carried out around the concept of Internet-of-Things (IoT). The holistic interaction among entities such as objects, systems, services and people, as prescribed by the IoT paradigm, represents the basic infrastructure on top of which is possible to develop complex platforms enabling a smarter environments and society. A plethora of research projects focuses on new and advanced platforms [1], providing more and more smart features for many different purposes, from Smart Energy [2] to Smart City [3] applications. In spite of the generic finality, existing IoT platforms are often designed to reach specific objectives and they lack of an easy and customizable way for instructing them properly to perform a given task.

The level of usability and the simplicity of current IoT platforms is quite low. The amount of complexity can be usually afforded only by people having enough programming skills. Filling this gap, providing means that ease the platform settings even by non-technical users, is considered a major challenge in the IoT panorama [4]. Within this scenario, the cooperative European-Brazilian project named IMPReSS [5] try to tackle this topic, since its objective is to make easy the realization of any kind of IoT applications that embrace the concept of "smart society". More specifically, IMPReSS aims at realize a so called System Development Platform (SDP), which enables rapid and cost effective development of systems involving the Internet-of-Things and Services and, at the same time, facilitates the interplay with users and external systems. Such kind of platform provides components and tools, which are designed to be more general-purpose possible, as well as easy to be integrated and used. Among the solutions developed in IMPReSS SDP, this paper will focus on the Internet-of-Things Platform's Infrastructure for Configurations (IoT-PIC). The aim of IoT-PIC is to extend the concepts of traditional network planning and management to IoT networks. It provides basic means for arranging, configuring and monitoring sub-components of an IoT platform, so that it can perform specific tasks, implementing specific applications and services. In other word, the proposed framework provides an easy-to-use model-based approach for the commissioning of general-purpose IoT platforms. With the term commissioning, the authors mean the logical deployment of available IoT platform sub-components necessary to instruct the platform about the workflow for implementing a specific application. Besides the commissioning, IoT-PIC allows to read

This work is part of the collaborative project IMPReSS, Intelligent System Development Platform for Intelligent and Sustainable Society, co-funded by the European Commission within the 7th Framework Program, FP7-ICT-2013-EU-Brazil, Grant Agreement no. 614100.

and set the parameters of each component of an IoT platform. Specifically, IoT-PIC provides these features using the XMPP protocol, an open standard originally designed for instant messaging, which has several extensions that makes it suitable for its use in the IoT scenario. Leveraging XMPP protocol is possible to monitor the status and availability of software and hardware modules belonging to the IoT platform. The architectural design of the proposed infrastructure is such that it can be used with different IoT platforms solutions.

The paper is structured as follows. Section II reviews related works about commissioning tools and traditional network management protocols, from which IoT-PIC took its basic requirements, adapting and extending them to the IoT world. In Section III, the IMPReSS SDP is briefly introduced. Section IV describes the IoT-PIC in its architecture and functionalities. Section V provides details about the IMPReSS components directly connected with the IoT-PIC, including the RAI and the IoT-PIC GUI. In Section VI an use case for the SDP is described. Section VII presents the result of an evaluation test done on the platform. Finally, in Section VIII, conclusions and future works are discussed.

II. Related Work

This section presents a State-of-The-Art (SoTA) of solutions related to the one presented in this paper. Firstly, a list of commissioning tools are presented, then the main protocols related to network management are introduced.

A. Commissioning tools

The lack in the SoTA about commissioning procedures and tools for IoT networks is that, usually, those platforms are programmed to deal with for specific tasks and reach specific objectives. Furthermore, the availability of fully automated tools is limited. While deploying an IoT application, the user usually needs programming skills to manage a huge number of configurations, customizations and integrations, which are highly prone to systemic or human errors. Therefore, the disposition of complex IoT environment becomes a hard work even for professional developers. Nowadays, such tools are explored for commissioning of complex systems in restricted areas of Smart Society, e.g. in Smart Building area. [6] provides an overview of the state-of-the-art of automated and semi-automated commissioning tools; it provides a list of commercially available automated commissioning tools and automated commissioning prototypes. For instance, LonMaker [7] is a single-product, a proprietary solution, which implements a standard design, commissioning and network maintenance software for LonWorks energy control networks. [8] deals with the techniques for measuring persistence of commissioning benefits and describes two tools developed for tracking the building performances. [11] provides an overview of the main commissioning procedures, describing the projects and the activities that have led to the development of BEMS-assisted commissioning tools. The European web portal BUILD UP [9] shares on its website a series of European tools developed with the aim of implementing the energy efficiency in buildings. The list of tools includes software applications, checklist for practitioners, etc. The "ICT Roadmap for Energy Efficient Neighbourhoods" (IREEN) [10] project has provided a roadmap that shows that most existing energy efficiency tools are implemented focusing on proprietary solutions, e.g. deployed for single buildings or based on proprietary protocols.

B. Network Management Protocols

Simple Network Management Protocol (SNMP) [12] 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. Anyway, SNMP presents unsolved vulnerabilities, which affect the security of the whole network [34], [35].

Common Management Information Protocol (CMIP) [13] is a network management protocol standardized by OSI that allows communication between management applications and management agents; specifically, CMIP implements the services defined by the other OSI standard Common Management Information Service (CMIS)[14]. The management information are organized in objects that can be read and modified. Furthermore, the agents can use the notification system to send alarms about the status of the network. Compared to SNMP, CMIP provides additional features: as IoT-PIC it allows the definition of any type of action to alter the state of a managed device, while SNMP allows only the "set" function. CMIP is a good alternative to SNMP and provides diverse features to model complex communication networks effectively. However, CMIP has a large amount of overhead, operating on top of the 7 layer OSI protocol, and furthermore is complex to be implemented and maintained [33].

Both solutions are standard protocols 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 [15], RESTFul protocols like CoAP [16], HTTP [17] and XMPP are protocols already used in IoT application deployment and therefore were considered conceivable to use them also for the network management 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. The aim of this work is to propose XMPP protocol as a valid alternative for network management in IoT contexts. 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. The performance of XMPP in term of latency, scalability and robustness have been widely demonstrated during the years [36]. 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 manage 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) [18] and Transport Layer Security (TLS) [19], both for client-to-server and server-to-server communications.

The proven scalability, the openness of the protocol, the security features natively provided, the possibility to identify univocally the entities and the possibility to easily interconnect things and humans are the features that have made XMPP one of the most used protocols in the IoT field. Keeping the same architecture of SNMP, but leveraging on an XMPP architecture, can be sent out notifications, restart tasks and seamlessly manage device availability (changing status, forcing web pages update, 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) [20] are used. The XEP-0030 (Service Discovery) [21], used to discover what entities are on the network and, exactly, which XMPP features those entities implement. The XEP-0050 (Ad-Hoc Command) [22], which provides workflow capabilities for any structured interaction between two XMPP entities. The XEP-0060 (Publish-Subscribe) [23] 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) [24] 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 contain 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.

III. THE IMPRESS SYSTEM DEVELOPMENT PLATFORM

This section introduces the architecture of the IMPReSS platform and describes the concept at the basis of the components described in the paper.

Figure 1 shows the IMPReSS platform architecture from a functional point of view.

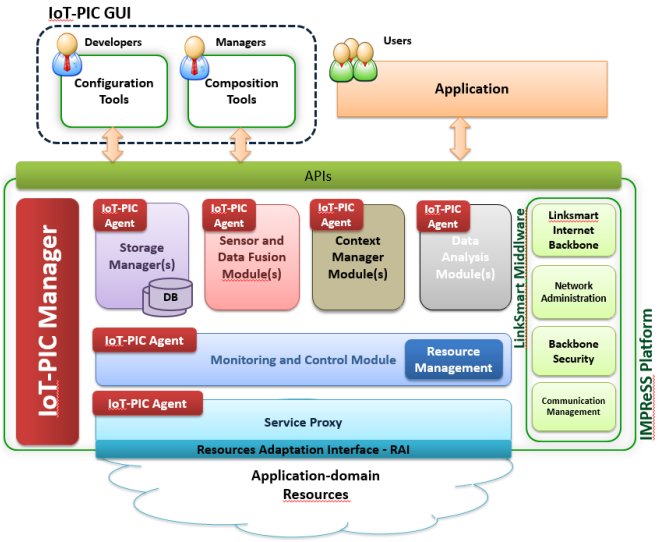

Figure 1. IMPReSS Functional architecture

Among the components of the platforms, this paper particularly focuses on the ones related with the commissioning and network management of the platform: the RAI, the IoT-PIC GUI and particularly the IoT-PIC. They have two main goals: the first one is to allow the composition of the different available modules in order to connect them and to realize desired applications and services; the second one is to allow the management of the status of both the entire platform and the single modules. The IoT-PIC GUI is a user interface, which relies on the IoT-PIC, responsible to perform the commissioning tasks requested by the stakeholder. The Resource Adaptation Interface (RAI), instead, is responsible of the virtualization of the devices connected to the network.

The stakeholders mostly involved with the provisioning issues are the Developers and the Managers. Developers combine different modules and compose the specific logic flow that the desired application has to compute. Actually, he/she realizes the final "IMPReSS-enabled" application through the usage of the SDP. 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 IoT-PIC allows dealing with the main aspects of commissioning and network management:

- *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 check if incoming values are above or below specific thresholds. The output of this last module can be used as input for the software module that drives a bell for announcing a critical situation. IoT-PIC framework 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 rise the alert. The IoT-PIC shows to the platform Manager all the available services and entities, allowing to configure the parameters of the entities 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 IoT-PIC GUI is a model-driven development toolkit that allows inexperienced developers to discover and compose distributed devices and services into mashups [30]. The proposed modeling 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 modeling. 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 their IoT platform, leveraging only on the actual services, among the ones available, required for their specific purposes.

IV. Internet-of-Things Platform's Infrastructure for Configurations (IoT-PIC).

The role of the IoT-PIC is to provide a unique and general way of performing the commissioning of the platform. The architecture of the IoT-PIC is shown in Figure 2. This architecture is inspired by the SNMP one (described in the section II) and aims at performing the configuration and composition of hardware and software resources.

Commercial devices for IoT (i.e. sensors and actuators, appliances) that provide Network Management functionalities usually leverage on proprietary or SNMP-based solutions, accessible from SDKs. The proposed solution aims to provide similar approaches at a GW level, which can act as aggregation point for NM information coming from WSANs abstracted by the RAI through technology-specific drivers (for details about RAI, see section V). This is particularly useful when the IoT platform backbone leverages on resource-constrained GWs such as Raspberry PI, where using a unique protocol that manages all IoT issues helps to save computational resources and power consumption. The architecture of IoT-PIC consists of two levels, the global and local one, and is mainly composed by two components:

- An IoT-PIC Manager (PIC_M) at a global level.
- An IoT-PIC Agent (PIC_A) at local level.

The communication among the components leverages the XMPP protocol.

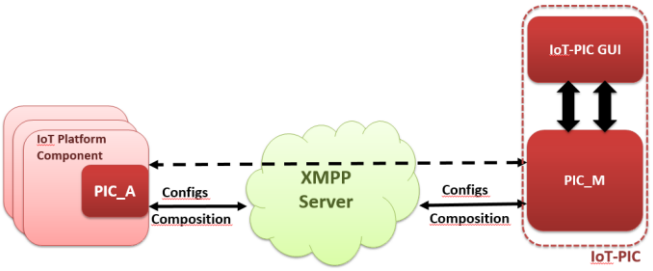

Figure 2. Configuration and Composition Framework architecture

The PIC_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 PIC_M consist in the following:

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

PIC_M is responsible for the management of the composition stage. In order to do this, the IoT-PIC 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 PIC_A is associated with each component of the platform, for example, for the devices, the PIC_A is embedded in the RAI driver, which manages the device. It exposes "get" and "set" ad-hoc commands, to manage configuration parameters of a specific component to the PIC_M. The PIC_A operates actually the configuration commands coordinated by PIC_M. The

association of an agent to each module makes the system more expandable and scalable from the point of view of configuration issues. PIC_A is responsible for:

- Register the component in the PIC_M.
- Handling the configuration parameters of the component.
- Handling the interconnection of the components with each other, adding and removing input sources.

IoT-PIC has been designed to be a stand-alone component that can extend different existing IoT platform with commissioning and configuration support, in order to build more complete IoT platforms. The user can interact directly using an XMPP client, modifying the configuration file or using a GUI, which communicates using the API provided by the PIC_M (as described in section V). This choice has been done to avoid to strictly coupling the IoT_PIC with other components of the IMPReSS platform. This design follows the request done by the EU Commission in the ICT-30-2015 call of the H2020 program [25], to "break the silos". The European Commission so identifies the need in the current IoT scenario: to build systems and tools, which can work in different platforms, instead to continue building new IoT solutions incompatible with existing ones. The IoT-PIC already address this issue, since even if has been developed as a LinkSmart [26] in IMPReSS project, it is compatible with different IoT platforms such as VIRTUS [27].

A. Composition operations

To allow the composition of the applications, the IoT-PIC 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 PIC_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 [28], the devices are described through the functions they provide and the operations possible on them. Specifically, the PIC_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 PIC_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, the PIC_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 PIC_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 IoT-PIC 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 PIC_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 PIC_M, which deletes the collection node of the device and its children.

B. Configuration operations

For the configuration part, the architecture built is similar to the one designed for other network management protocols described in section II. Specifically, every PIC_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 applications, which has to configure the various components, interact with the PIC_M that provides two ad-hoc commands to read and write the configurations on the various PIC_A.

V. IMPRESS COMPONENTS

This section describes the components, which interact with the IoT-PIC to enable the system Integrators and Managers to deploy and configure an instance of the IMPReSS platform.

A. *Resource Adaptation Iinterface*

The Resource Adaptation Interface (RAI) is an evolution of the Physical World Adaptation Layer (PWAL) described in [29]. The RAI virtualizes each resource as a Virtual Device that exposes features or functionalities, provided by physical devices or third-party services, through a set of methods and parameters defined by specific Java interfaces.

The RAI improves the PWAL in several aspects and its architecture has been fully refactored. Three layers (see Figure 3), completely decoupled with each other, compose the RAI architecture. In this way, it is possible to change one of them, without requiring many modifications to the others.

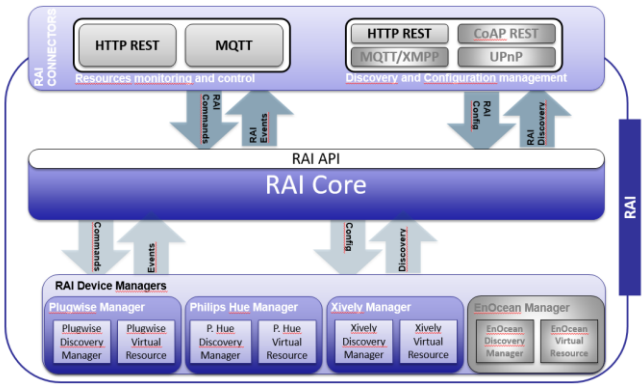

Figure 3: RAI architecture

The lower layer of the architecture consists in a set of technology-specific Device Managers classes that are responsible for the actual integration of different resources. These components are able to handle specific types of networks and furthermore, they contain the implementation of specific device discovery features.

A set of application-level resource models are used for the virtual device interface definition. The modelled interfaces are implemented with specific commands, depending on the specific resource to be integrated. The middle layer is the RAI core, which is in charge to map the southbound devices and to notify upper layers about each network changing. The upper layer is responsible for the exposition of the methods/services provided by the resources. This layer is made of the APIs offered by the RAI core, in order to retrieve and manage virtual devices and call their resource-specific methods.

In IMPReSS, the IoT-PIC interacts with the RAI in several ways, through the XMPP protocol. The user, through the IoT-PIC, can install/remove and start/stop the Device Managers of the RAI. Furthermore, the IoT-PIC is used to configure the Device Managers, using the methods described in the previous section. Finally, the IoT-PIC receives the notifications from the RAI, when a new device is connected to the network.

B. *IoT-PIC GUI*

IoT-PIC GUI (Figure 4) allows the system integrators to manage and monitor an instance of the IMPReSS platform. IOT-PIC GUI 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.

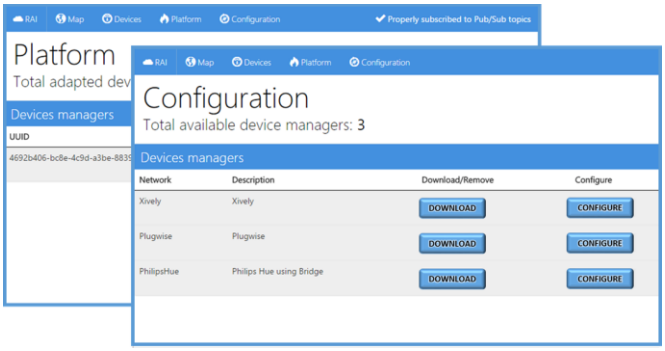

Figure 4 – IoT-PIC web interface

An installation wizard procedure has been designed to guide the user into the different configuration tasks. Once launched, the IoT-PIC 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 IoT-PIC 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 click the download button, the PIC_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 uninstall the previous module and replace it with the new one. Start/stop: this operation allow to start and stop 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 IoT-PIC 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 PIC_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 4. Through this web page, it is possible to interact with the PIC_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 PIC_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 in the bundle, through the PIC_M.

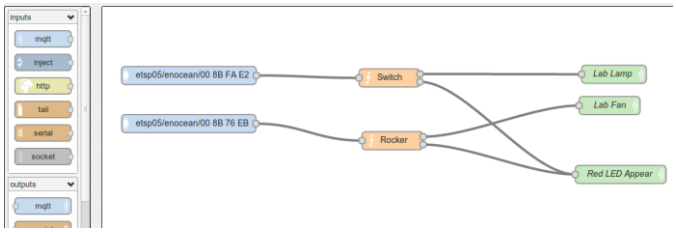

Figure 5: IoT-PIC GUI – Commissioning view

The IMPReSS toolkit is completed by a model driven development (MDD) tool for IoT applications, see Figure 5. The architecture of this tool is similar to the one described in [29], with the difference that this one uses the features provided by the IoT-PIC, to discover 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 IoT-PIC. 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.

VI. USE CASE

The IoT-PIC presented in this paper has been implemented and tested in an energy efficiency scenario. In the beginning of the scenario, an integrator, using the GUI presented in previous section, installs all the components of the IMPReSS platform. The instance of IMPReSS includes two applications (energy saver and alarm system) and three types of IoT resources (Presence sensors, Lights, Smart plugs) deployed into the system. The Energy saver manages the lights, in order to save energy (i.e. It turns off the lights where there is no class scheduled). When there is a class, presence sensors detect if a row of seats in the classroom is empty. For the empty areas, the lights are automatically switched off. In the proposed scenario, in the aforementioned condition, is it possible to imagine the following situation: presence detection done only through presence sensors often gives false positive and so, in order to reduce the number of errors, other sensors need to be integrated. At this point, it is possible to see how the IMPReSS platform allows integrating new devices, without requiring modifications to the existing applications. For example, a system integrator can decide to add a Kinect sensor to improve presence detection. The integrator, using the IoT-PIC and the web interface, searches for a driver available for the Kinect (assuming has been already developed and published by another developer) and uses it for the integration. The driver is installed at runtime in the RAI, which discovers the presence of the new device. Through the PIC_A, the device creates on the server the hierarchy of pub/sub nodes, which represent its functions and data produced. The component is therefore automatically added in the model driven tool. Using this tool, the user can connect, for example, the output of this component to the input of a hypothetical lights management one. Because of this connection, the PIC_M indicates to the PIC_A of the lights management component to subscribe itself to the pub/sub node of the device, in order to receive its events. Finally, using the web interface the user can configure the various components of the platform; the interface can now convert the XML structure returned by the ad-hoc command of the PIC_M in a form that can be compiled by the user, in order to tune, at runtime, the behavior of the components, to satisfy his/her needs.

VII. EVALUATION

The solution presented in this paper has been evaluated during a test organized by the IMPReSS project consortium at Universidade Federal de Pernambuco (UFPE). Ten people, among students and ICT technicians, have tested the components of the platform, filling out a user experience questionnaire. Particularly, the questionnaire requires an evaluation on the following aspects: *attractiveness* – general impression towards the component*; efficiency* – how fast and efficient the component is and how the user interface look organized*; perspicuity* – how much is easy to understand how to use the component and how does it work*; dependability* – how much the user feels while using the component (i.e. is the interaction with the product secure and predicable?)*; stimulation*: how much interesting and exciting people perceived the component; *novelty* – how much the design of the component is innovative and creative.

The results of the evaluation about the IoT-PIC (Figure 6) show a good appreciation of the component, particularly regards its usability.

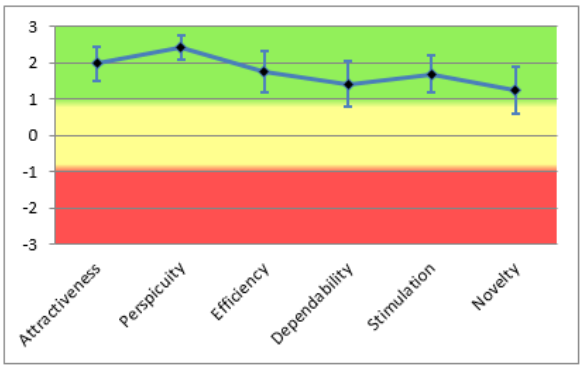

Figure 6 – Evaluation results

VIII. CONCLUSION AND FUTURE WORKS

The paper has presented a novel IoT Network Management infrastructure, based on the XMPP protocol, and it has described how this solution is leveraged in the IMPReSS architecture, as

part of a complete Model Driven Development toolkit. The solution proposed allows applying Network Management features, typical of IP network, in the IoT scenario. Through this framework, the gateways, sensors and actuators involved can be configured and monitored in real-time, using a standard IoT communication protocol, without requiring to support a specific additional protocol only for this task; this aspect is particularly important in the IoT scenario, because of the use of resource-constrained devices. Features provided by the IoT-PIC enable the easy composition of IoT application, allowing the automatic discovery of new components and the interconnection of IoT entities. The proposed solution guarantees a high-level data security using TLS encryption and SASL authentication, embedded in the XMPP protocol.

In the next future, the authors will investigate the possibility to enhance the IoT-PIC, with some useful features typical of Network Management solution, like the possibility to evaluate the quality of service and the round trip time or other parameters used to monitor the status of a network.

Furthermore, a set of application-level resource models are used for the virtual device interface definition. The modelled interfaces, implemented with specific commands depending on the resource to be integrated, are defined through Java interfaces. Those interfaces define a basic number of methods/services provided by the most common device types, but the usage of ontology descriptions will be investigated.

Finally, currently the XMPP community, while promoting the use of XMPP in IoT scenario [31], is working to a set of extensions including an experimental extension regarding to IoT Discovery [32]. The authors of this paper are going to investigate how to integrate these extensions in the presented solution.

REFERENCES

[1] The ebbits Project. (Online), http://www.ebbits-project.eu/news.php.

[2] GreenCom Project. [Online], http://www.greencom-project.eu/.

[3] Jalali, R.; El-Khatib, K.; McGregor, C., "Smart city architecture for community level services through the internet of things," in Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on , vol., no., pp.108-113, 17-19 Feb. 2015

[4] Sarkar, Chayan, et al. "A scalable distributed architecture towards unifying iot applications." Internet of Things (WF-IoT), 2014 IEEE World Forum on. IEEE, 2014.

[5] IMPReSS web site (online) http://impressproject.eu/news.php.

[6] Frank, M. et al. (2007), "State-of-the-Art Review for Commissioning Low Energy Buildings: Existing Cost/Benefit and Persistence Methodologies and Data, State of Development of Automated Tools and Assessment of Needs for Commissioning ZEB", NISTIR 7356, 2007

[7] LonMaker.(Online), http://www.echelon.com/products/tools/integration/lonmaker/.

[8] Friedman, H. et al. (2010), "Commissioning Cost-Benefit and Persistence of Savings", Cost-Effective Commissioning of Existing and Low Energy Buildings, Energy Conservation in Buildings and Community Systems (ECBCS) Program.

[9] BUILD UP web portal. (Online), http://www.buildup.eu/home

[10] IREEN Project. (Online), http://www.ireenproject.eu/

[11] Visier, J. C. et al. (2004), "Commissioning tools for improved energyperformance", Energy Conservation in Buildings and Community Systems(ECBCS) Program.

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

[13] ITU X.711 (1997) Common management information protocol: Specification (online) http://www.itu.int/rec/T-REC-X.711/en/

[14] ITU X.710 (1997) Management Communication Service and Protocol (online) http://www.itu.int/rec/T-REC-X.710-199710-I/en

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

[16] 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/.

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

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

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

[20] XMPP Standard Foundation – XMPP Extensions (online) http://xmpp.org/xmpp-protocols/xmpp-extensions/

[21] XMPP Standard Foundation – XEP 0030 Service Discovery (online) http://xmpp.org/extensions/xep-0030.html

[22] XMPP Standard Foundation – XEP 0050 Ad-Hoc Commands (online) http://xmpp.org/extensions/xep-0050.html

[23] XMPP Standard Foundation – XEP 0060 Publish-Subscribe (online) http://xmpp.org/extensions/xep-0060.html

[24] XMPP Standard Foundation – XEP 0248 PubSub Collection Nodes (online) http://xmpp.org/extensions/xep-0248.html

[25] EU Commision. ICT 30 2015: Internet of Things and Platforms for Connected Smart Objects (online) http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=8269.

[26] LinkSmart Middleware Portal (online) https://www.linksmart.eu/redmine.

[27] Conzon, D., T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M. A. Spirito, "The VIRTUS Middleware: An XMPP Based Architecture for Secure IoT Communications", 21st International Conference on Computer Communications and Networks (ICCCN), pp. 1 -6, 07/2012.

[28] OSGi Alliance. RFC 196 Device Abstraction Layer (online) https://github.com/osgi/design/raw/master/rfcs/rfc0196/rfc-0196-DeviceAbstractionLayer.pdf

[29] Conzon, D.; Brizzi, P.; Kasinathan, P.; Pastrone, C.; Pramudianto, F.; Cultrona, P., "Industrial application development exploiting IoT vision and model driven programming," in Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on , vol., no., pp.168-175, 17-19 Feb. 2015

[30] Benslimane D., Dustdar S., and Sheth A., "Services Mashups: The New Generation of Web Applications," IEEE Internet Comput., vol. 12, no.5, pp. 13–15, Sep. 2008

[31] XMPP Standard Foundation – XMPP–IoT (online) http://wiki.xmpp.org/web/Tech_pages/IoT_systems.

[32] XMPP Standard Foundation – XEP 0347 Internet of Things – Discovery (online) http://xmpp.org/extensions/xep-0347.html.

[33] Roh, Inho, and Ilsoo Ahn. "CMIP based Light MIB: Design & Implementation."

[34] Chatzimisios, Periklis. "Security issues and vulnerabilities of the SNMP protocol." 1st International Conference on Electrical and Electronics Engineering. 2004.

[35] Lawrence, Nigel, and Patrick Traynor. "Under New Management: Practical Attacks on SNMPv3." WOOT. 2012.

[36] XMPP perfomances [online] https://iotprotocols.wordpress.com/2015/03/31/performance-tests-xmpp/