(FP7 614100)

D3.5 Templates and Integration Support Tool

30.03.2016 – 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: IMPRESS D3.5_templates_and_integration_support_tool_v1.0.docx
Document version: 1.0
Document owner: Enrico Ferrera (ISMB)

Work package: WP3 Resource Abstraction and IoT Communication Infrastructure
Task: T3.1 Resource Adaptation Interface and Integration Support tool
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 (ISMB)	08.01.2016	First draft
0.2	Davide Conzon (ISMB)	20.01.2016	Introduction and Integration Support Tool section
0.3	Enrico Ferrera (ISMB)	21.02.2016	Explanation about the deviation from the DOW point of view
0.4	Enrico Ferrera (ISMB)	15.03.2016	Sections finalization. Deliverable ready for peer review.
0.5	Enrico Ferrera (ISMB)	30.03.2016	Documents modified according to comments from peer review.
1.0	Enrico Ferrera (ISMB)	30.03.2016	Deliverable ready to be submitted.

Internal review history:

Reviewed by	Date	Summary of comments
Trine F. Sørensen	29.03.2016	Checked Lessons Learned and Requirements section. Accepted.
Peeter Kool (CNet)	30.03.2016	Accepted with minor comments

Index:

1. Executive summary

The goal of this deliverable is twofold: firstly, it aims to describe Resource Adaptation Interface (RAI) in its improvements respect to the description reported into D3.1 – Resource Adaptation Interface Framework. Secondly, it describes the model the RAI uses for describing resources both for development of RAI's Device Manager and registration of virtual IoT Resource on Resource Catalogue (see D3.3 – Resource and Service Discovery Solutions).

2. Introduction

2.1 Background

The Internet-of-Things (IoT) consists in a holistic interaction among objects, devices, systems, services and people based on a general-purpose infrastructure on top of which is possible to develop complex applications and services enabling "smart society". It leverages a plethora of heterogeneous objects, each one providing specific functionalities that are accessible through specific communication protocols. For this reason, an *abstraction and adaptation layer* is necessary, in order to blend the access of many different resources in a common language and set of procedures. Standing to [1], an IoT platform architecture is layered as shown in Figure 1. The Object Abstraction component is the very low layer of an IoT platform and it is located just before the IoT objects, taking part of the IoT ecosystem.

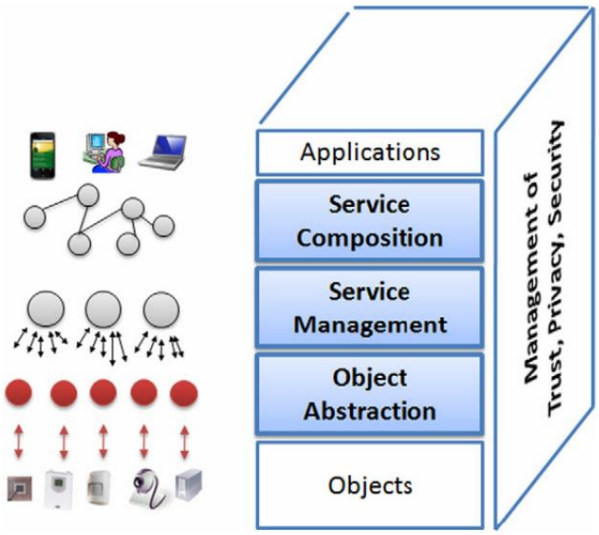

Figure 1: IoT platform architecture

Resource Adaptation Interface, or RAI, aims at cover the role of such abstraction layer. More specifically, RAI is a framework that collects a set of existing classes and methods that can be used in order to monitor and control application-level resources. In fact, RAI aims to abstract the concept of resource (i.e. physical devices or third-party systems), providing a virtual "device" that can be used to seamlessly communicate with resources despite of technology-specific implementation details. RAI aims to ease and speed up the integration of application-level resources within IMPReSS platform. The main goal of the Resource Adaptation Interface is to integrate and expose the features provided by heterogeneous physical and non-physical resources (e.g. sensors or actuators, personal devices of engaged citizens, services providing public data), providing to the IMPReSS platform a uniform way to communicate with them, despite the differences in term of hardware and software.

RAI is located on the extreme edge of the IMPReSS platform, just before the hardware and software resources, and can cooperate with the LinkSmart middleware, which the platform is based on. The heterogeneous nature of physical devices requires finding a way to interact with the resources. For this reason, the architecture of the RAI has been designed to be modular, in order to make it extensible through the addition of new resource drivers abstracting new, previously unhandled, entities. RAI can abstract both physical resources (e.g. sensors, actuators) and third-party services (e.g. existing platform such as Xively [2], weather forecast web service, etc.).

2.2 Resource Adaptation Interface – RAI

The RAI architecture is composed by three layers, completely decoupled from each other. In this way, it is possible to change one of them, without requiring many modifications to the others.

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 interfaces provided by RAI are used by the Local Resource Manager (see D4.2 – Device and Subsystem Resource Management) to interact with the available resources. The methods defined by the specific resources interfaces are passed as parameters to the method *requestResource(String appID, String operation)* provided by the LRM APIs.

In Figure 2 is shown the layered architecture of the Resource Adaptation Interface.

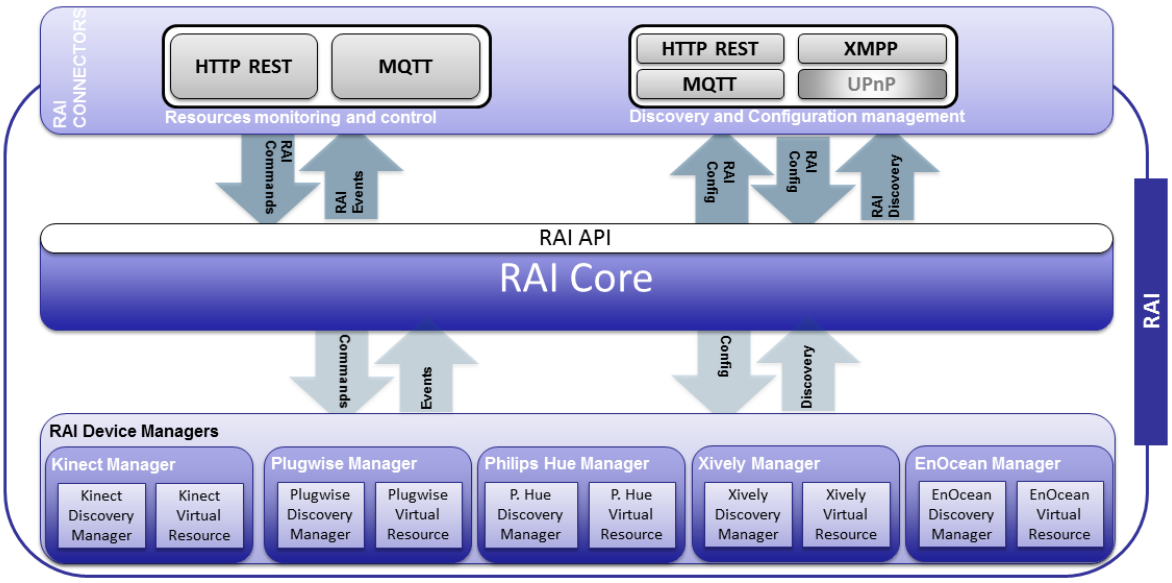

Figure 2 - Resource Adaptation Interface Architecture

The lower layer of the architecture consists in a set of technology-specific *DeviceManager*(s) 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. The device-level discovery topic is deeply described in D3.2 – Resource and Service Discovery.

A set of application-level resource models, contained in a dedicated repository, are used for the virtual device interface definition. The modelled interfaces are implemented with specific commands depending on the specific resource to be integrated. For the first implementation of the RAI, the models consist just in a bunch of Java interfaces that define a basic number of methods/services provided by the most common device types. For next versions of RAI, it is going to be investigated the use of ontology descriptions for resource models.

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.

3. IoT Resource templates and Integration Support Tool

3.1 Lesson Learned and deviation from the DoW

The description of WP3 reported into the IMPReSS Document of Work, or DoW, states the following:

"This work package is responsible for providing software components and tools to the IMPRESS platform to enable seamless and easier interoperability with the available application-domain heterogeneous resources. The objectives of WP3 are related to the implementation of a resource adaptation framework and a relevant development toolkit as well as the design and the implementation of a lightweight network communication and management infrastructure. To enable seamless and automatic virtualization of such heterogeneous resources (i.e. devices, systems and services), this WP will exploit the results of the ebbits project and extend the LinkSmart middleware with a lightweight standardized abstraction layer, namely Resource Adaptation Interface (RAI). Such adaptation layer will be designed in order to be executed in devices with limited resources e.g., ARM-based hardware platforms. In addition, this WP will provide the necessary Development Toolkit allowing developers to rapidly integrate devices, systems and sub-systems present in the reference smart society environment into the overall IoT platform. Such toolkit will leverage on an extendable collection of RAI templates to easy the implementation of technology-specific RAI modules. The provided templates will support technologies in the energy efficiency domain including legacy technologies such as PLCs, fieldbus metering devices, emerging technologies such as ZigBee, 6LowPAN and classical IoT solutions like EPC Global based RFID tags and readers, and NFC tags. These technologies will be iteratively and further identified in the requirement engineering process of the project to ensure that the results of the project will have a direct substantial impact for the modern and smart society development. Besides templates, IMPRESS platform will support the developers with a model driven development tool allowing them to design and implement the layer of resources adaptation rapidly."

The original idea at project proposal time was to provide to the users of the IMPReSS platform a development tool that would have to ease the integration process of physical resources. More specifically, other than the resource abstraction layer itself, the idea was also to create a tool for writing code for a faster integration of resources leveraging predefined resource templates modelling the specific technology of the new resource to be integrated. These templates would have to be used for automatically generate most part of the code, which was supposed to be the same for all physical devices sharing common communication technology. The work of the developer would have consisted just in writing few lines for complementing the code with device-specific implementations.

In D2.1.2 – Requirement and Lessons Learned report, the two lessons learned reported in the following table are described.

LL No.	Experience and knowledge gained	Lesson Learned
WP3-3	The classification of resources with respect to the type of wireless communication technology adopted is not useful the design the RAI modules. The integration is made at Gateway level.	Gateway communication protocol classification is more useful
WP3-4	Device Managers handling different technologies still share some part of source code	Automatic generation mechanisms can be used to support the development of Device Managers. As a result, a new requirement has been created: IMP-41 RAI shall provide a tool that help developer while creating new device managers.

These two Lessons Learned were identified one year ago, in February 2015, and are still true in their *Experience and knowledge gained* description but while working on the requirements resulting from them we outlined some thoughts, which are described in the following statements:

- The classification of resources with respect to the type of wireless communication technology adopted is not useful for developing the RAI drivers. The integration is made considering the gateway communication technology.
- The majority of commercial devices have a gateway that does not expose a standard application profile but just a proprietary way to interact with the WSAN that it manages. For example:
    - Plugwise -> ZigBee network with proprietary gateway REST protocol
    - PhilipsHue -> ZigBee network with proprietary gateway REST protocol
- Some commercial devices implement standard application profiles that are exposed by their gateways. In these cases, once a Device Manager has been developed it can be fully reused for every single device implementing that standard application profile. There is no need to have a tool that automatically generates a bunch of common code to be complemented with device specific implementations.
- The common lines of source code shared by different Device Managers are static code that can be moved from Device Managers classes to RAI core classes in order to furtherly reduce the code size of the Device Manager reducing consequently the effort requested to the developer.
- Making some code optimization, the Device Managers can be reduced to just the minimum amount of code needed to develop specific technology-dependant implementations.
- The tool that minimizes the amount of code to be implemented during physical device abstraction is the RAI itself. The common code is contained in RAI core. The technology-specific code is the Device Manager.
- We still need some device models in order to describe the services exposed by themselves. This models, or templates, must be used for generating the APIs that must be used in order to interact with integrated devices. For example:
    - A temperature sensor must expose a getTemperature service;
    - A smart plug must expose switchOn/switchOff and getPower services;
- The templates describe devices according to their type and services that are able to provide, independently from the implemented communication technology or standard.

- Anyway, an IMPReSS platform user still needs a tool that eases and speeds the usage of Resource Adaptation Interface according to the actual physical devices that are meant to be added to the platform.

Following these statements, the Lessons Learned can be declined in the following way:

LL No.	Experience and knowledge gained	Lesson Learned
WP3-3	The classification of resources with respect to the type of wireless communication technology adopted is not useful the design the RAI modules. The integration is made at Gateway level.	Gateway communication protocol classification is more useful
WP3-4.1	Device Managers handling different technologies still share some static source code	RAI code has to be optimized in order to move static code out from Device Managers. Device Managers have to contain just code relevant to the specific technology to be integrated.

WP3-4.1	In order to plug just needed Device Managers in RAI, the platform managers need programming skills. This makes the platform deployment error prone.	Support mechanisms can be used to help the integration of relevant Device Managers in the platform. As a result, a new requirement has been created: IMP-41.1 RAI shall provide a tool that help managers while integrating relevant devices during platform deployment.

During the last year of IMPReSS project, we aimed to work on these statements. More specifically, we refactored RAI improving it in order to reduce to the minimum the amount of code needed to implement Device Managers. All the specific code for the relevant device technology communication is delegated to the development of just one class: DeviceManager class.

In the remaining part of this deliverable, will be described how we dealt with the open issues stated above in this document. Furthermore, the templates used for modelling devices will be described. Finally, the revised Integration Support Tool will be shown.

3.2 IoT Resource Templates

RAI uses a set of templates in order to describe resources. These templates are used to define the interface of virtual devices, or IoT Resources, generated by RAI and at the same time the same templates is sent to the IoT Resource Catalogue in order to register the resources available in the IMPReSS platform. IoT Resource Catalogue provides the means to store metadata regarding the services provided by IoT Resources. IoT Resource services descriptions are expressed in an extended version of SCPD (Service Control Protocol Description) which is the standard for service descriptions in DLNA/UPnP. An example of the SCPD description is shown below, see Figure 3.

The reason for using the extended SCPD format is that it is well defined and used for service discovery as well that it is possible to describe services independently of their implementation. This makes it possible to describe REST based services which do not really have any formal description language.
There are two ways to register an IoTResource, i.e. service, with the IoT Resource Catalogue:

- UPnP Discovery using SSDP and SCPD
- SELF Registration using HTTP or MQTT

IMPReSS users are supported while creating new SCPD templates by a dedicated tool. This tool has been developed during the course of the project and is described in D7.2.2 – Integrated Component Platform prototypes.

If an IoTResource supports the UPnP Protocol the IoTResource will register automatically with the IoT Resource Catalogue. If a service is integrated using the developer tools this information will be created mostly automatically and the service will be discovered dynamically by UPnP as well. But it also possible to manually register the service in the IoT Resource Catalogue if one prefers by using the RegisterResource action of the catalogue service of the IoTResourceCatalogue.SCPD templates of all device available in the IMPReSS platform can be fetched through an HTTP GET at the following URL:

http://{RAI_IP}:{RAI_PORT}/devices/{Resource_ID}/scpd

```
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <!-- UPnP Elements -->
    <deviceType>urn:schemas-upnp-org:IoTdevice:smartplug:1</deviceType>
    <presentationURL>http://130.192.86.221:8888/</presentationURL>
    <friendlyName>Plugwise SmartPlug B1BDE9-INSTANT_POWER</friendlyName>
    <manufacturer>Plugwise</manufacturer>
    <manufacturerURL>http://www.ismb.it</manufacturerURL>
    <modelDescription>
      A SmartPlug device connected to a Plugwise network.
    </modelDescription>
    <modelName>smartplug</modelName>
    <modelNumber>1</modelNumber>
    <UDN>uuid:0021b7d9-c33a-3d60-8dd6-c181a9b35aae</UDN>
    <!--  Resource Catalogue Fix for OGC SensorThings API  -->
    <metadata xmlns="IoT">
      {"id":"6d9001be7db18de0217cf93039d53e1ba92e033dab319ff6277fe80a206a63f9","Description":"The SmartPlug connected to the Plugwise network.","Locations":[{"Time":"2016-03-28T22:31:42.709Z","Geometry":
      {"type":"Point","coordinates":[577.0,500.0]}}],"Datastreams":[{"id":"f5988d738d80b1bd89af0bdfe06efbdad8ed9c04dfbbb533527f33f2bc1802f9","ObservedProperty":
      {"id":"e2e030918130ece249734df50811cff54ba86e86fbbe24eb6bf6eceb867a10ad","UnitOfMeasurement":"Watts"}},{"id":"cee16aa6a3c4331ce7404870175cfbff8be25a4046645a90708b9e84ab87da682","ObservedProperty":
      {"id":"0f4eccd5989bb9fbc372d1d4204a84940629ae0d12fc112aa5c09c21eed64380","UnitOfMeasurement":"Watts"}},{"id":"e89f50c82b8ecdc73bcc969ad34fa5af7fe5b44d00016e46b96f024e5a42135c","ObservedProperty":
      {"id":"c3d1b98854ccb096f241654c98d9f77d2aeb3342f6e550f6bd30ff2fbb8f2680","UnitOfMeasurement":"Watts"}},{"id":"51a99e043d900bcec769be563a44b78ec3233d1943906a6fc3327f1f27022592","ObservedProperty":
      {"id":"a1d3534918f1a04948a1c39fef88cfb2b10731e3617f9f78afe0943aafbe460b","UnitOfMeasurement":"Watts"}}]}
    </metadata>
    <!-- CNet extension Elements -->
    <IoTresourceid xmlns="IoT">0021b7d9-c33a-3d60-8dd6-c181a9b35aae</IoTresourceid>
    <gateway xmlns="IoT">130.192.86.221:8888</gateway>
    <errormessage xmlns="IoT"/>
    <networkType xmlns="IoT">Plugwise</networkType>
    <pwalId xmlns="IoT">0021b7d9-c33a-3d60-8dd6-c181a9b35aae</pwalId>
    <power xmlns="IoT">0.0</power>
    <id xmlns="IoT">B1BDE9-INSTANT_POWER</id>
    <unit xmlns="IoT">Watts</unit>
    <updatedAt xmlns="IoT">2016-03-28T22:31:39.497Z</updatedAt>
    <supportedCommand xmlns="IoT">
      {turnOff=it.ismb.pertlab.pwal.api.devices.commands.impl.meter.TurnOffMeter@18c5f02, getPower=it.ismb.pertlab.pwal.api.devices.commands.impl.meter.GetPowerMeter@da95f1,
      turnOn=it.ismb.pertlab.pwal.api.devices.commands.impl.meter.TurnOnMeter@294453, isOn=it.ismb.pertlab.pwal.api.devices.commands.impl.meter.IsOnMeter@183da4}
    </supportedCommand>
    <location xmlns="IoT">577.0 500.0</location>
    <type xmlns="IoT">SmartPlug</type>
    <expiresAt xmlns="IoT">2016-03-28T22:31:40.497Z</expiresAt>
    <configuration xmlns="IoT">it.ismb.pertlab.pwal.api.xmpp.PicForm@1380600</configuration>
    <lastMeasurement xmlns="IoT">2016-03-28T22:31:39.497Z</lastMeasurement>
    <mqttBrokerIP xmlns="IoT">tcp://130.192.85.32</mqttBrokerIP>
    <mqttBrokerPort xmlns="IoT">1883</mqttBrokerPort>
    <mqttTopics xmlns="IoT">
      <mqttTopic xmlns="IoT" type="metadata">
        /impress/metadata/iotentity/0021b7d9-c33a-3d60-8dd6-c181a9b35aae
      </mqttTopic>
      <mqttTopic xmlns="IoT" type="observation">
        /impress/observation/iotentity/0021b7d9-c33a-3d60-8dd6-c181a9b35aae
      </mqttTopic>
      <mqttTopic xmlns="IoT" type="observationOGC">
        /v2/observation/6a577d707c12429bffa2b63f17ada8a4044d7642fd4f11553783ce3bc842dfe1/2a14e826901fd396897825e5e75497ac263deca76dc4bc6a790df07efb81da22
      </mqttTopic>
      <mqttTopic xmlns="IoT" type="observationOGC">
        /v2/observation/6a577d707c12429bffa2b63f17ada8a4044d7642fd4f11553783ce3bc842dfe1/e08a1164aa930a02000d7a2909fe69e968a4efaad0cf8d884ce45b5021d564cf
      </mqttTopic>
    </mqttTopics>
    <!-- UPnP Elements -->
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:smartplugservice::1</serviceType>
        <serviceId>
          urn:schemas-upnp-org:smartplugservice_0021b7d9-c33a-3d60-8dd6-c181a9b35aae:1
        </serviceId>
        <SCPDURL>
          _urn:schemas-upnp-org:tsmartplugservice_0021b7d9-c33a-3d60-8dd6-c181a9b35aae:1_scpd.xml
        </SCPDURL>
        <controlURL>
          _urn:schemas-upnp-org:smartplugservice_0021b7d9-c33a-3d60-8dd6-c181a9b35aae:1_control
        </controlURL>
        <eventSubURL>
          _urn:schemas-upnp-org:smartplugservice_0021b7d9-c33a-3d60-8dd6-c181a9b35aae:1_event
        </eventSubURL>
      </service>
    </serviceList>
  </device>
</root>
```


Figure 3: Example of a service description in SCPD

In Figure 4 is shown a list of the devices modelled so far and supported by the RAI.

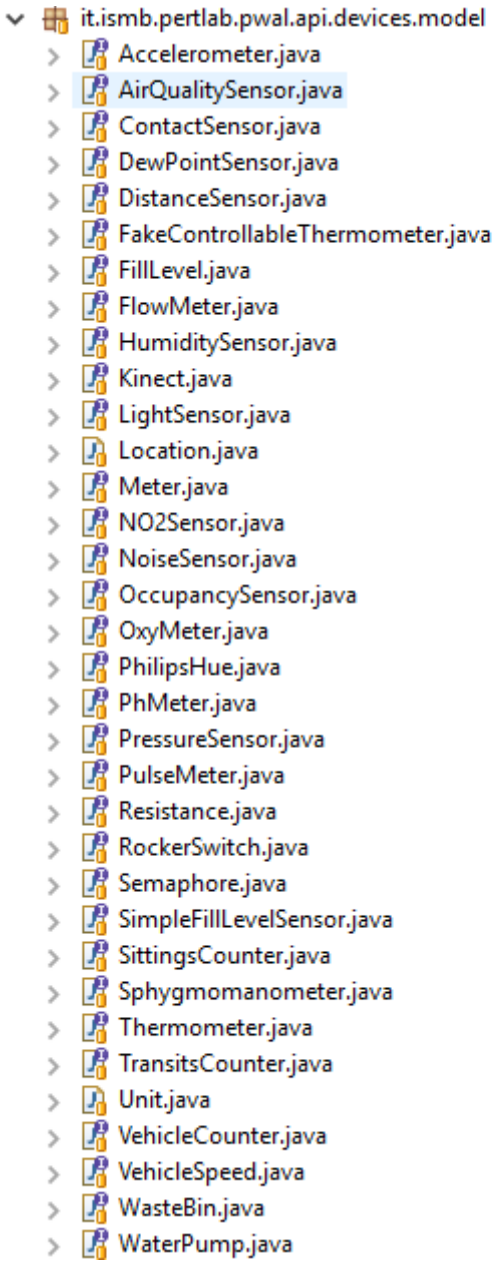

Figure 4: List of modelled devices

3.3 Integration Support Tool

The Integration Support Tool allows IMPReSS platform users to integrate easily and rapidly physical devices, needed to implement the desired IoT application, into the local IMPReSS platform instance. This tool consists in a local web interface that allows configuring the adaptation layer of the IMPReSS platform, i.e. the RAI.
Web interface allows users to configure, monitor and test the relevant RAI instance through a web browser.

As indicated in D4.2 – Device and Subsystem Resource Management, the RAI is a sub-component of the IoT Resource, which acts as a service proxy. The IoT Resource architecture with the RAI is shown in Figure 5. The role of the RAI is to abstract physical resources and expose their functionalities in a common way through the provided API. The Local Resource Manager can interact with the RAI using the exposed API, handling thus, the requests received by the IMPReSS applications. Other than REST APIs useful for its monitoring and control, the

IoT Resource provides also a Web Interface used for its setup and configuration. Particularly, it allows to plug and play specific Resource Managers in order to drive physical resource that IMPReSS platform users have previously supplied in order to develop the desired IoT application.

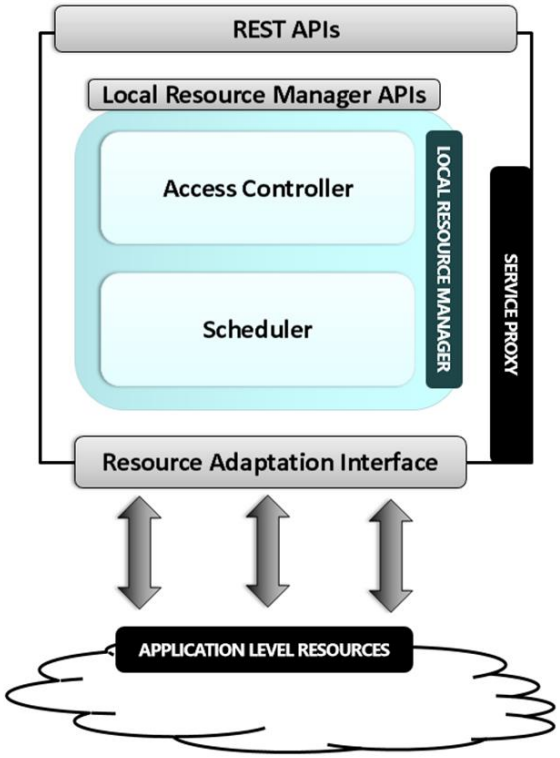

Figure 5 - Service proxy architecture

You may consider an IMPReSS IoT Resource as a box that is initially empty.
Through the Integration Support Tool web interface (see Figure 6), the IMPReSS platform users can download desired Device Managers and rapidly install them locally, without any programming skills. In the same way, users can remove no more useful Device Managers in order to optimize performance and resource allocation. She/he can stop Device Managers without necessarily remove them from the local IoT Resource instance and then restart them in a second time; this feature is useful mainly for maintenance purposes. Through this web application is even possible to manage Resource Manager updates, whenever they are available.

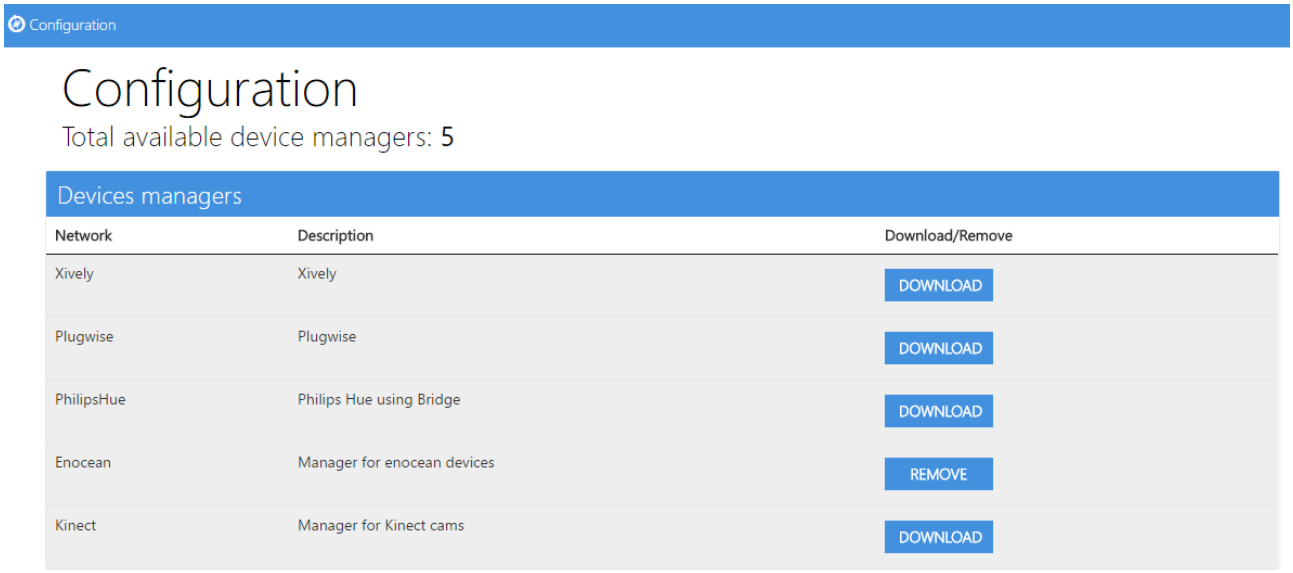

Figure 6: IMPReSS IoT Resource Integration Support Tool web interface

This web interface leverages the communication infrastructure described in D7.3.2 – Final Design and Implementation of the Configuration and Composition Manager and in [3].
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 Device Managers. 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 Integration Support Tool tries to connect to the local XMPP server. If the connection fails, the web interface 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 Resource Managers downloaded from the web link. The GUI provides, for each bundle, the following operations:

- Download/remove: when the user clicks the download button, the management module of the configuration infrastructure (i.e. CC_M) download the Device Managers from the remote repository and install it in the local RAI 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 Device Managers, in order to inform in real-time the user if the Device Manager is correctly running, or if there is some error in its execution.

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.

4. Conclusions

In this deliverable is reported how physical devices have been modelled through SCPD templates. Furthermore, Integration Support Tool has been presented. It has been designed targeting IMPReSS platform managers and maintainers.
Integration Support Tool has been evaluated by students and IT managers during the evaluation meeting organized in Recife, Brazil, in November 2015. It received a very good feedback in terms of attractiveness, efficiency, perspicuity, dependability, stimulation and novelty, according to the User Experience Questionnaire (UEQ), which consists of twenty-six statements or items plus six additional statements that have been added in order to stress certain quality aspects that were important to be evaluated. More information about evaluation process and results can be found in D8.5 – Platform Analysis and Feedback Report.

5. References

[1]. L. Atzori, A. Iera e G. Morabito, «The Internet of Things: A survey», *Computer networks 54(15),* pp. 2787-2805, 2010.
[2]. «Xively,» 2016. [Online].
[3]. E. Ferrera, D. Conzon, P. Brizzi, L. Gomes, M. Jentsch, P. Kool, «XMPP-based Network Management Infrastructure for Agile IoT Application Deployment and Configuration», *ICIN 2016 conference on Innovations in Clouds, Internet and Networks*, 2016