Target Outcome: b) Sustainable technologies for a Smarter Society

(FP7 614100)

D4.2 Device and Subsystem Resource Management

31 January 2014 – Version 1.1

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: D4 2 Device and Subsystem Resource Management_v1.docx
Document version: 0.4
Document owner: Enrico Ferrera (ISMB)

Work package: WP4 – Mixed Criticality Resource Management
Task: T4.2 Device and Subsystem Resource Management
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	Davide Conzon, Enrico Ferrera		Table of content
0.2	Jussi Kiljander		Contributions to sections 1 and 4
0.3	Davide Conzon, Enrico Ferrera	2014/03/30	First draft
0.4	Enrico Ferrera	2014/04/12	Ready for internal review
1.0	Enrico Ferrera	2014/04/30	Document modified according to the internal review
1.1	Davide Conzon	2014/10/11	Document modified with the API of the LRM modified to handle several resources.

Internal review history:

Reviewed by	Date	Summary of comments
Ferry Pramudianto	2014/04/28	Requires some clarification on the interaction between GRM & LRM.

Index:

1 Executive summary

The goal of this deliverable is to give a definition of Service Proxy and describe how it is composed and works. Furthermore it is a compendium of the prototypical implementation of the interface between the Resource Management components and the Service Proxies. Particularly, the interface represents a software layer that interacts with the Application Level Resources, which are abstracted by the Resource Adaptation Interface and exposed as services through the proxies.

In Chapter 2 there is an introduction to the general architecture of the Resource Management module of the IMPReSS platform, which is composed by both global and local components. The local ones are part of Service Proxies, which are defined and described in Chapter 3. In Chapter 4 there is the description of the interface between Service Proxies and Resource Management, which is the core of this document. According to the fact that D4.2 actually consists in the prototypical implementation of the previously mentioned interface, this document is considered just a compendium with the interface specification and for this reason has been kept with a restricted number of pages.

2 Introduction

We believe that in the future IoT systems will be open computing platforms that support applications developed by 3rd party developers much in the same way as PCs, tablets, and mobiles phones at the moment. The main difference between IoT platforms and these traditional computing platforms is that the IoT systems are highly distributed and consist of numerous heterogeneous devices that monitor and interact with the physical world in order to provide the necessary computing facilities for applications. The heterogeneous nature of this type of IoT platforms imposes many challenges that need to be tackled before the functionality of the devices can be provided as a uniform computing platform. In particular, managing the access to resources between mixed criticality 3rd party applications is a big challenge in these types of open IoT systems.

In IMPReSS project we provide a solution to this challenge by abstracting the resources access and providing platform components for managing the access to application level resources (i.e. sensors and actuators).

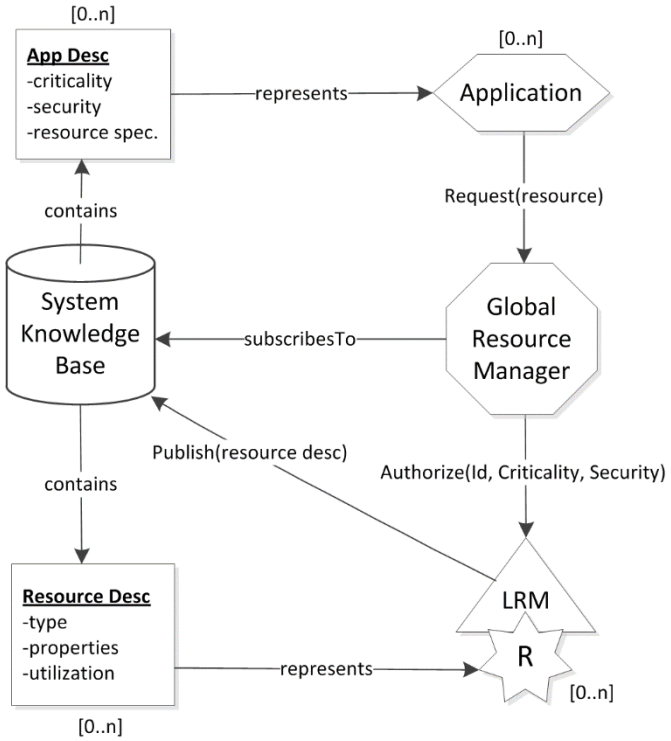

Figure 1.Resource management architecture for mixed critical IoT applications.

The resource management architecture (presented in the Figure 1.) consists of two levels: global and local. At the global level the role of the resource management is twofold. First, solve conflicts between applications that either request exclusive access to a same resource or request access to different resources that might interfere with each other in the real word (e.g. lights, heating systems, etc.). Second, optimize the usage of resources shared between applications so that the performance of the whole IoT system is as optimal as possible.

The global level resource management architecture consists of two components, namely *System Knowledge Base* and *Global Resource Manager*. The *System Knowledge Base* is a shared memory of the system that provides publish/subscribe interface for other components of the IMPReSS platform to share information about resources, applications and devices in a machine-interpretable format. The *Global Resource Manager* is the functional component responsible for the actual resource management. It subscribes to the application and resource descriptions published to the *System Knowledge Base* and is this way aware of the current state of the system and thus able to manage the resource access.

At local level the resource management is handled by a *Local Resource Managers* (LRMs) assigned for each resource (or Service Proxy). It is the responsibility of the LRM to guarantee that more critical applications are served before less critical ones. Additionally, the LRM ensures that the applications accessing the resources have has been authorized by the GRM. In this deliverable, we will describe the interfaces and functionality of the LRM in detail.

3 Service Proxy

The D4.1.1 (IMPReSS, 2014) has presented the domain model of mixed critical IoT systems as reported in Figure 2.

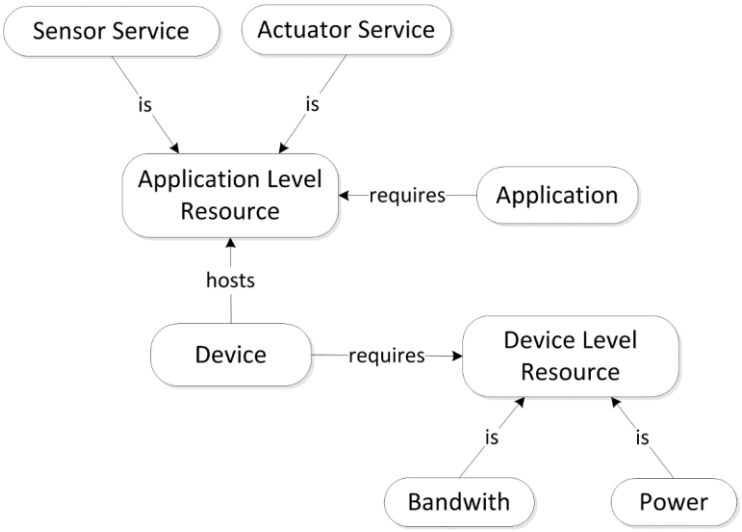

Figure 2. Domain model for mixed criticality in Internet of Things.

In the Internet of Things – Architecture (IoT-A) project (Carrez, 2013) is stated that the main goal of IoT is to enable users, *i.e.* humans or software agents, to interact with the physical world. The interaction is possible leveraging on means, i.e. devices and/or external systems, which are able to monitor, control and retrieve information about the physical world status. These means can be classified in two categories:

- Sensors: which measure physical phenomenon and provide enhanced information regarding the current context of external world. Temperature sensors, smoke detectors and third-party systems providing current energy price or meteorological information are examples of sensors.
- Actuators: which modify the state of the physical world by taking commands from other parts of the system. Air condition systems, automated doors and lights are examples of typical actuators.

Sensors and Actuators are the resources available to the applications (i.e. software programs developed with the IMPReSS platform) in order to realize a specific domain logic. For this reason, we refer to Sensors and Actuators as *Application Level Resources (ALRs).*

The formal interfaces between the applications and the ALRs are defined as Services. A **Service Proxy** is an IMPRESS platform component that is responsible for the implementation of the interface between the ALRs and their users. In other words, a Service Proxy is a software component that is responsible for the abstraction of ALRs, hiding the various implementation details (e.g. communication protocols and data format), and instead providing a uniform virtualization for them within the IMPRESS platform. In conclusion, when applications need to interact with the physical world through ALRs, they can do it through virtual resources (i.e. Service Proxies) that provide a bridge to the actual Sensors and Actuators, exposing their offered services.

3.1 Description of the service proxy architecture

In Figure 3 is shown the internal architecture of a Service Proxy. A Service Proxy is composed by different sub-components:

- Resource Adaptation Interface (RAI)
- Local Resource Manager (LRM)
- REST interface

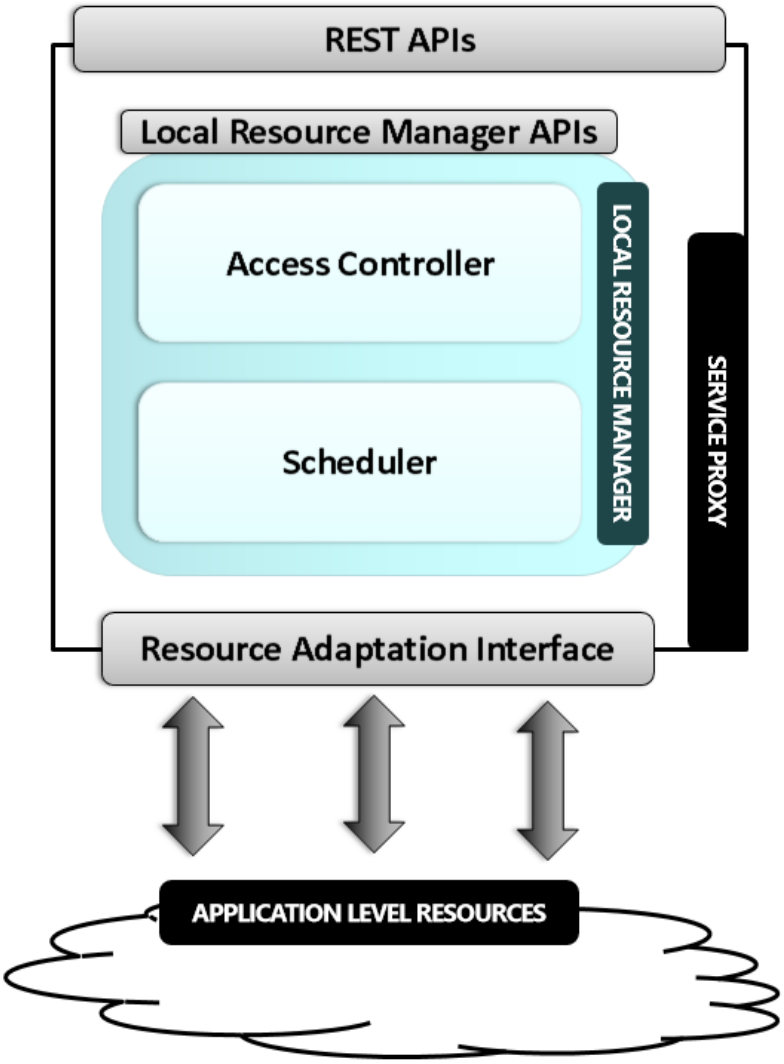

Figure 3: Service Proxy architecture

The **Resource Adaptation Interface** is a software layer that allow to integrate different ALRs (i.e. devices or sub systems) in the physical world to the IMPReSS platform. RAI acts as the glue between the IMPReSS system and the physical world. One can see this layer as the hardware drivers needed for IMPReSS in order to communicate with the ALRs. It is the lowest layer of the Service Proxy and is used to hide technology-specific details for interaction with ALRs. The abstracted ALRs are virtualized within IMPReSS platform by the Service Proxies themselves and used by the Applications in order to implement suitable domain logics. RAI is defined and implemented in Task 3.1 and will be described more deeply in D3.1.

The **Local Resource Manager** is a Service Proxy component responsible for the local management of ALRs. It is composed by two sub-components:

- The *Scheduler*, which schedules the access to ALRs. Scheduling is only needed when multiple applications have been authorized to access the same resource (i.e. when the applications use a shared resource access scheme). The basic principle in LRM's Scheduler is to guarantee that more critical applications are served before less critical ones. The definition of a scheduling scheme will be more deeply addressed in deliverable D4.3 related to Task 4.3. A possible approach that can suit IMPReSS goals can be a priority based pre-emptive scheduling, by assigning a fixed priority to the mixed critical applications. This approach guarantees the more critical applications are served before less critical ones. The main issue of this approach is that it can happen that the less critical applications are not served at all, determining the starvation

of lower priority applications when there are many high critical applications. In some cases can be more suitable a round-robin scheduling algorithm, which allow an application to access a resource following the order of the requests. In this case, the scheduler will perform the pre-emption of the running application placing it at the end of a queue of waiting applications and continuing giving access to the next application within the queue.

- The *Access Controller*, which is responsible for publishing the resource description of the ALR it manages into the System Knowledge Base. In this way, the ALRs are "registered" to the Global Resource Manager (GRM). Once GRM grants the access to the ALR (see D4.1.1 for more details), the Access Controller is also responsible for checking if the Application who requests to interact with ALR is the one previously authorized by GRM. If the Application has been authorized, it can proceed calling the specific service provided by the abstracted ALR. The interaction between the applications, Local Resource Managers and Global Resource Manager are illustrated with two example scenarios in Section 3.2. This interaction is made possible by the implementation of Local Resource Manager APIs. The description of the APIs for the intercommunication with LRM is reported in Section 4.1.
- REST APIs are the access door to the services offered by Service Proxy. They consist in a mapping of LRM APIs in REST commands for accessing ALRs services. The description of the REST APIs are reported in Section 4.2.

3.2 Interaction of the LRM with the global components for resources management

When a new resource (and LRM) is added to the IMPRESS platform, it needs to be first discovered by the Resource Discovery Manager component. The actual methods for resource discovery will be investigated in Task 3.2 and reported in Deliverable 3.2. Once a new resource is found, the Resource Discovery Manager requests the description of the resource and inserts it into the System Knowledge Base. This way other components of the IMPRESS platform are aware of the resources in the network. One of these components is the Global Resource Manager that is subscribed to the resource specifications defined by the applications and will be notified every time a new resource matching the specification is found. This interaction is depicted in the Figure 4.

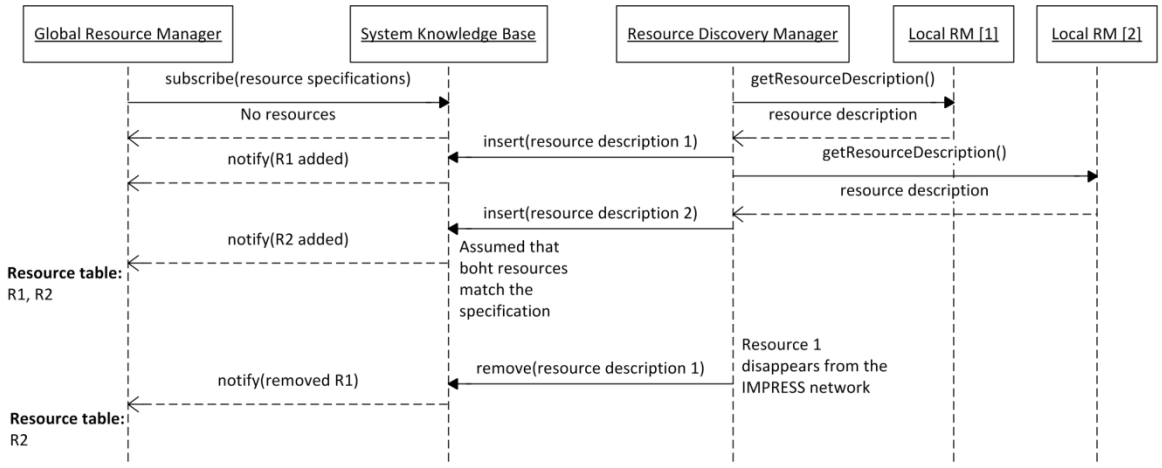

Figure 4. Resource discovery and registration.

Application get access to resources by sending a *reserveResource()* request to the GRM. The message parameters define the resource specification of interest. The GRM will select the most suitable resource for the application from all the resources matching the given specification and notifies the application about the resource. In order to assign the more suitable resources to the applications, the resource description must be constantly updated with informations such as the total service execution time, packet loss rate, service utilization rate, etc. Such kind of information are going to be retrieved by the IMPReSS Network Manager, which will operate at both resource and Service Proxy levels

in order to update the resource description on the System Knowledge Base. The role of the Network Manager will be more deeply analysed and defined within Deliverable D3.4. An extended version of the Service Proxy interface will be defined in order to provide relevant services to the Network Manager. Additionally, the GRM informs the LRM about the application authorized to access the resource. This is done with the *authorizeAccess()* message which defines the application ID, the criticality, and the required security level. Once the application and the LRM have been notified about the pairing the application can start requesting domain specific services provided by the resource. This happens be sending a *requestResource()* message to the LRM with the actual domain specific operation as payload. If the LRM receives multiple simultaneous requests it schedules them based on the criticality of the application. This way the more critical applications are always served before less critical ones. The Figure 5 illustrates message exchange between applications, GRM and LRMs in an example scenario where two applications need to access a sensor resource using shared access scheme.

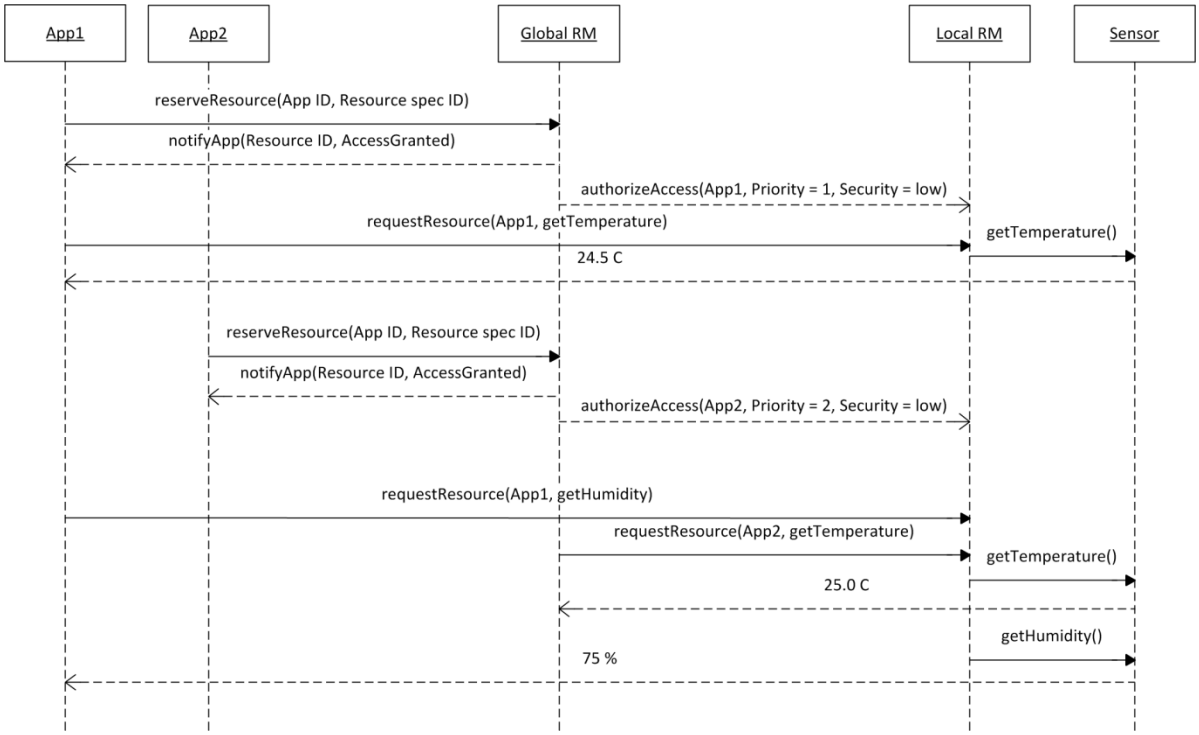

Figure 5. Message exchange between applications, global RM, local RM, and a resource.

4 Local Resource Manager API

In this Section, the APIs exposed by the LRM to the GRM for the control of ALRs are described. The section is subdivided in two parts, in the first one there is the description of the Java implementation of the interface between LRM and GRM, with also some examples of specific domain APIs; in the second one there is the description of the REST interfaces exposed by the Service Proxy, with the mapping between these and the Java ones.

4.1 Java API

Java Interface of the LRM to the GRM and some examples of domain specific APIs of sensors and actuators.

4.1.1 Interface LRMInterface

public interface **LRMInterface**

This is the interface implemented by the LRM to regulate the access to local resources. These methods are called (through the REST interfaces) by the GRM, which is in charge to control the resource allocation to the apps at global level.

Methods

requestResource

Object **requestResource**(String resourceID. String appID, String operation)

Parameters:

appID: ID of the application that has requested to execute an operation on the resource.

resourceID: required: ID of the resource on which the operation has to be executed.

operation: specific operation to be executed on the resource.

Returns:

Resource/Operation specific value.

Description:

This operation is used by apps to request a specific service from the application level resource. The LRM uses the Application ID to schedule the access to the resource (indicated by the resource ID) and passes the resource specific operation to the resource that responses to the app directly.

authorizeAccess

void **authorizeAccess**(String resourceID, String appID, int priority, String securityLevel)

Parameters:

resourceID: required: ID of the resource on which the operation has to be executed.

appID: ID of the application that is authorized to access to the resource.

priority: level of priority assigned to the application (used to schedule the access to the resource, among different applications, with different priorities).

securityLevel: level of security required to access to the resource (e.g. low, medium, high).

Description:

Informs the LRM (resource) about an app that is allowed to access the resource indicated by the resourceID.

deauthorizeAccess

void **deauthorizeAccess**(String resourceID, String appID)

Parameters:

resourceID: required: ID of the resource on which the operation has to be executed.

appID: ID of the application that is no longer authorized to access to the resource.

Description:

Informs the LRM that the app cannot access the resource (indicated by the resource ID) anymore.

getResourceDescriptions

DescriptionList **getResourceDescriptions**()

Returns:

Descriptions of the resources.

Description:

Used to obtain the descriptions of the resources

getServices

ServiceList getServices()

Returns:

List of services provided by the resources.

Description:

Used to obtain the list of service provided by the resources.

4.2 REST API

REST interfaces and mapping of these ones with the Java APIs presented before.

4.2.1 LRM REST interface

GET lrm/request_resource/:resourcedID/:operation

POST lrm/authorize_access

POST lrm/deauthorize_access

GET lrm/get_descriptions

GET lrm/get_services

Services

GET lrm/request_resource/:resourceID/:operation

Java Interface method:

requestResource (see the Java interface for method description).

Description:

The GET method has been chosen for this operation, because is idempotent on the status of the LRM. The appID and resource ID are passed as custom header and not as sub-resource or parameter of the URL, because it does not involve the name of the resource (the url), the state of the resource (the body), or parameters directly affecting the resource (parameters).

Parameters:

appID: *required.* ID of the application that has requested to execute an operation on the resource. Example value: App124

resourceID: required: ID of the resource on which the operation has to be executed. Example value: 4d62f720-5309-11e4-8247-0002a5d5c51b3&appID:app12

operation: *required.* Specific operation to be executed on the resource. Example value: getTemperature.

Returns:

Resource/Operation specific value.

Example request:

GET http://127.0.0.1/lrm/request_resoure/4d62f720-5309-11e4-8247-0002a5d5c51b3/getTemperature

POST lrm/authorize_access

Java Interface method:

authorizeAccess (see the Java interface for method description).

Description:

The POST method has been chosen for this operation because it updates the status of the LRM, the parameters are passed as POST data.

Parameters:

resourceID: required: ID of the resource on which the operation has to be executed. Example value: 4d62f720-5309-11e4-8247-0002a5d5c51b3

appID: *required.* ID of the application that is authorized to access to the resource. Example value: App124

priority: *required.* Level of priority assigned to the application (used to schedule the access to the resource, among different applications, with different priorities). Example value: 1

securityLevel: *required.* Level of security required to access to the resource. Example value: low

Example request:

POST http://127.0.0.1/lrm/authorize_access

POST Data: resourceID:4d62f720-5309-11e4-8247-0002a5d5c51b&ID:app124&priority=1&securityLevel=low

POST lrm/deauthorize_access

Java Interface method:

deauthorizeAccess (see the Java interface for method description).

Description:

The POST method has been chosen for this operation for this operation because it updates the status of the LRM, the parameters are passed as POST data.

Parameters:

resourceID: required: ID of the resource on which the operation has to be executed. Example value: 4d62f720-5309-11e4-8247-0002a5d5c51b3

appID: ID of the application that is no longer authorized to access to the resource.

Example request:

POST http://127.0.0.1/deauthorize_access

POST Data: resourceID:4d62f720-5309-11e4-8247-0002a5d5c51b3&appID:app12

GET lrm/get_descriptions

Java Interface method:

getDescriptions (see the Java interface for method description).

Description:

The GET method has been chosen for this operation, because is idempotent on the status of the LRM.

Returns:

Description of the resources.

Example request:

GET http://127.0.0.1/lrm/get_descriptions

GET lrm/get_services

Java Interface method:

getServices (see the Java interface for method description).

Description:

The GET method has been chosen for this operation, because is idempotent on the status of the LRM.

Returns:

Services provided by the resources.

Example request:

GET http://127.0.0.1/lrm/get_services

5 Summary & Conclusion

In this deliverable, we described a Service Proxy with its architecture and functionalities.

To interact with the IMPReSS platform, a resource from the physical world is abstracted by the Resource Adaptation Interface and virtualized within the platform as a provider of services by the Service Proxy. Every single call for services from an Application if filtered by the Local Resource Manager, which check a previously granted authorization to access the resources.

The interaction with the Service Proxy is made possible through the utilization of REST APIs that allow to communicate with the Local Resource Manager. The Local Resource Manager APIs and the REST interface, which are the focus of task T4.2, are described at the end of this document.

6 Bibliography

Carrez, F. (2013). *IoT-A project deliverable D1.5 – Final architectural reference model for the IoT v3.0.* Retrieved from http://www.iot-a.eu/public/public-documents/d1.5/view

IMPReSS. (2014). *IMPReSS project deliverable D4.1.1 – Initial application classification language.*