
(FP7 614100)

# D4.3 Resource Management & Access Scheduler

## 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_D4_3_ Resource_Management_and_access_scheduler_internal_review_version.docx
**Document version:** 1.0
**Document owner:** VTT

**Work package:** WP4
**Task**: T4.3 Resource Management & Access Scheduler
**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 | Jussi Kiljander | 10/09/2015 | Table of contents |
| 0.2 | Jussi Kiljander | 17/09/2015 | Initial content for sections 3 and 4. |
| 0.3 | Jussi Kiljander | 28/09/2015 | Initial content for sections 1, 2 and 6. |
| 0.4 | Jussi Kiljander | 09/10/2015 | Updates to sections 3 and 4 |
| 0.9 | Jussi Kiljander | 19/10/2015 | Updates to all sections. Draft version for internal review. |
| 1.0 | Jussi Kiljander | 27/10/2015 | Modifications according to the review comments. |

**Internal review history:**

| Reviewed by | Date | Summary of comments |
|---|---|---|
| Carlos Kamienski | 25/10/2015 | Approved with minor corrections |
| José Ángel Carvajal Soto | 23/10/2015 | Approved with some suggestions and comments. |

# Index:

# 1. Executive summary

This deliverable describes the overall approach, architecture and reference implementation of the mixed criticality resource management component of the IMPReSS platform. This is a Prototype deliverable and the source code with examples can be downloaded from the IMPReSS git and installed as specified in the Appendix I. Mixed criticality is a new topic in the field of the Internet of Things (IoT) and although this is a Prototype type deliverable the approach is also explained in detail.

The approach covers both application and device-level mixed criticality resource management features. At the application-level the idea in the mixed criticality resource management is to manage how IoT applications can access IoT Resources (i.e., sensors and actuators), which are provided by the IMPReSS platform. At the device-level the goal in turn is to manage device-level resources. To this end, we have decided to focus on device energy management and developed features for managing which devices are provided with energy in the case there is a power outage in the electricity network.

The IMPReSS mixed criticality middleware consists of two-level architecture. At the system level the Global Resource Manager is responsible for selecting suitable resources for applications and selecting which application can access which resource. It also manages which devices need to be turned off if there is not enough energy to power all the devices for the duration of the power outage. At the resource level there is one Local Resource Manager for each device that interacts with the Global Resource Manager to make sure that only authorized applications access the resource and schedules the requests send by the applications when needed. Additionally, the Local Resource Manager turns off/on the devices whenever requested by the Global Resource Manager.

The deliverable is organized as follows. Chapter 2 provides motivation and background for the deliverable. Chapter 3 provides an overall approach and architecture for mixed criticality resource management middleware. In chapters 4 and 5, respectively, the application and device-level mixed criticality management aspects are described in detail. The chapter 6 concludes the deliverable.

# 2. Introduction

## 2.1 Purpose, context and scope of this deliverable

Smart phones and tablets have revolutionized people's life by bringing different kinds of services and applications on handheld devices. Although these devices make it possible to utilise digital services wherever the people are this is still far from the vision of ubiquitous computing (Weiser 1999) in which the digital services are embedded into our everyday living environments.

In order to fully realise the visions of ubiquitous computing and the Internet of Things (IoT)[1] there is a need for technologies that make it possible to develop applications that utilise resources provided by various kinds of devices embedded into our everyday living environments. To this end, the IMPReSS project develops a System Development Platform (SDP) that provides developers with means to create IoT applications for building automation, facility management and energy management application domains.

An important part of the IMPReSS SDP is a middleware that makes it possible to support applications and resources with mixed criticality requirements. Without this type of middleware, the IoT systems would need to be carefully designed by a single party and it would not be possible to achieve an open computing platform that supports 3rd party applications and resources.

Mixed criticality in the context of IoT is a novel topic and to the best of our knowledge no existing approaches or solutions for it exist in the literature. Therefore, it is very important to first identify the needs for mixed criticality resource management and clarify its purpose in this context. In the IMPReSS we divide the mixed criticality resource management into two levels, referred to as device-level and application-level. The goal of the application-level mixed criticality resource management is to make it possible to deploy and execute multiple independent IoT applications on the same IoT platform without compromising the behavior of the more critical applications. The aim of the device-level mixed criticality resource management in turn is to make sure that the most critical IoT devices are supplied with power in the case of power outage.

To further clarity the concept the IMPReSS domain model for mixed criticality resource management is depicted in the Figure 1.

[1] At the moment IoT is a more popular term but the idea is basically the same as in ubiquitous or pervasive computing.

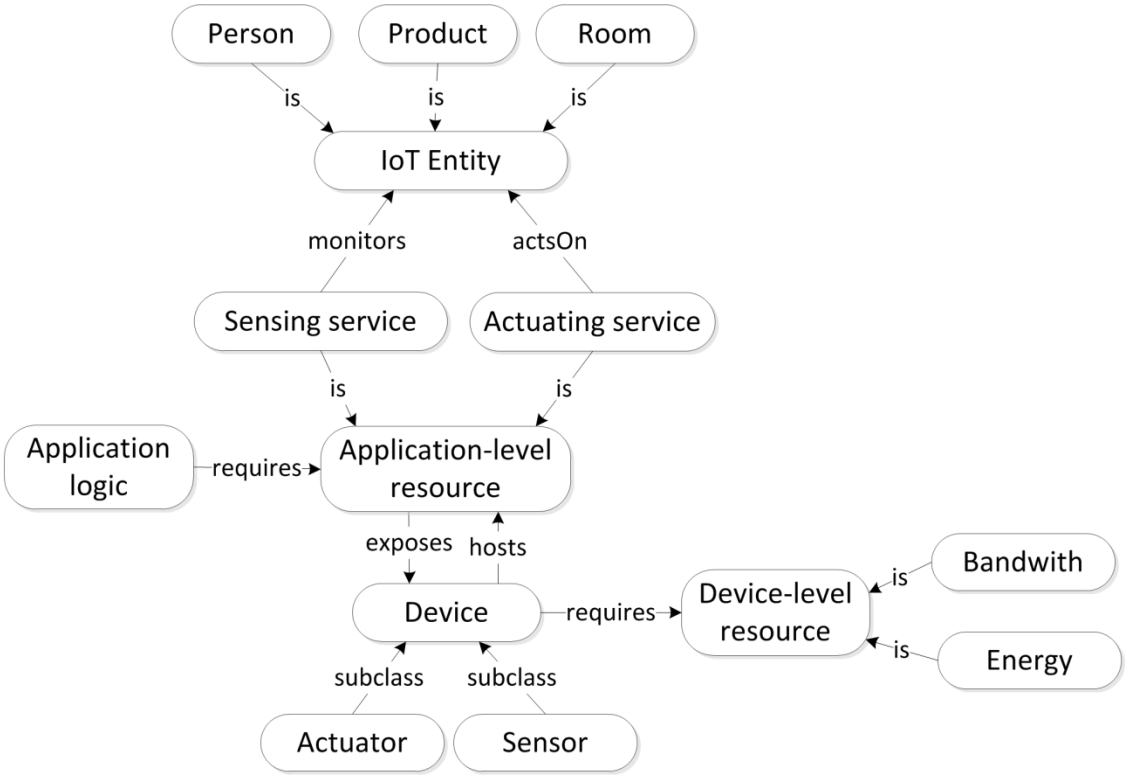


Figure 1. Domain model for mixed criticality in Internet of Things.

The aim of the IoT is to provide users with means to interact with the physical world (Bassi et al. 2013). In the domain model (presented in the Figure 1) the physical world objects that are relevant for the given IoT system are called IoT Entities. The IoT Entities can be human beings (e.g., eHealth related IoT system), inanimate non-ICT objects (e.g. food products), places (rooms, buildings, etc.) or devices (home appliances, servers, etc.). The only requirement is that they are somehow in the interest of the IoT system and need to be monitor and/or controlled by the IoT devices. When compare to the IoT-A Architectural Reference Model (ARM) (Bauer et al. 2013) the IoT Entity is basically equivalent to the Physical Entity (or Virtual Entity) of the IoT-A ARM domain model.

The business logic (or domain logic) that defines how the user can interact with the IoT Entities in a specific use case is realized by applications developed with the IMPReSS SDP. In practice, the interaction between the application and the IoT Entities is achieved through sensors and actuators that provide applications with means to monitor and manipulate the physical world, respectively. Consequently, sensors and actuators are referred to as *Application-level resources* in the IMPReSS mixed criticality resource management domain model. When compared to the IoT-A DM for the Application-level resources cover the functionality provided by the IoT Service and IoT Resource.

In addition to the application-level resources, more low-level resources such as energy and bandwidth are also vital for IoT systems. In the domain model we call these resources *Device-level resources* because they are needed by devices that are either IoT Entities (e.g. server, freezer, etc.) or devices (i.e., sensor and actuator platforms) that host *Application-level resources*. The main focus in the IMPReSS project has been on the application-level mixed criticality resource management, but we have also developed a solution that provides means to handle energy management between devices with differing criticality levels.

## 2.2 Background

In the literature (Burns and Davis 2013) systems that can support applications with differing criticality requirements are called mixed-criticality systems (MCS). The term MCS has been traditionally used within application domains such as cars and air planes (e.g. AUTOSAR[2] and

[2] http://www.autosar.org/

ARINC[3]) where the main challenge is that different kinds of software applications (or tasks) run on the same computer sharing the Central Processing Unit (CPU) and memory. Consequently, the MCS research has mainly focused either to 1) MCS scheduling approaches for single or multicore processors (Vestal 2007) (Huber et al. 2008) or to 2) system partitioning approaches to provide protection for safety critical systems (Hill and Lake 2000).

In IoT domain the challenges of building MCS are completely different however. The four main differences between traditional and IoT specific MCS are listed below:

1. Instead of traditional computer resources such as the processor and the memory, the resources that are important in IoT are the devices (e.g. sensors and actuators) utilised by IoT applications and the resources needed by these devices (e.g. energy).[4]
2. Traditional MCS (cars, airplanes, etc.) are typically developed (or at least) integrated by single party whereas our vision of IoT is an open platform in which 3rd party applications could be deployed.
3. In traditional MCS the resources and applications to be deployed into the system are typically known at design time whereas the idea in IoT is that the system constantly evolves as the user upgrades it with new application and devices.
4. IoT systems rely on best-effort networks (e.g. the Internet, ZigBee, Bluetooth, etc.) which means both that it is not possible to provide deadlines for communication latencies and that the resources can even become occasionally totally unavailable.

## 2.3 Structure of the Deliverable

The rest of the deliverable is structured as follows. In the Chapter 3 the overall approach and architecture for mixed criticality resource management is depicted. The Chapters 4 and 5 elaborate the application and device-level mixed criticality management aspects, respectively. Chapter 6 summarizes and concludes the deliverable.

3 http://www.arinc.com/

4 In addition to sensors and actuators, the IoT applications, of course, needs traditional resources such as processor and memory, but since it is not typically relevant where the logic of the IoT application runs (i.e. the application logic of different application do not need to run on same device) this is not a relevant problem for typical IoT systems.

# 3. Mixed criticality resource management architecture

## 3.1 Architecture overview

The mixed criticality resource management architecture (with some other closely related IMPReSS components) is depicted in the figure 2. It consists of following entities: applications, IoT resources, Global Resource Manager (GRM), Local Resource Manager (LRM) and the Development & Management Tools. These components work in co-operation with the other IMPReSS modules in order to provide mixed criticality resource management functionality at the application and device levels. In addition to these component an important part of the mixed criticality resource management approach are the application and resource descriptions. The application and resource description formats are introduced in more detail in the section 3.2.

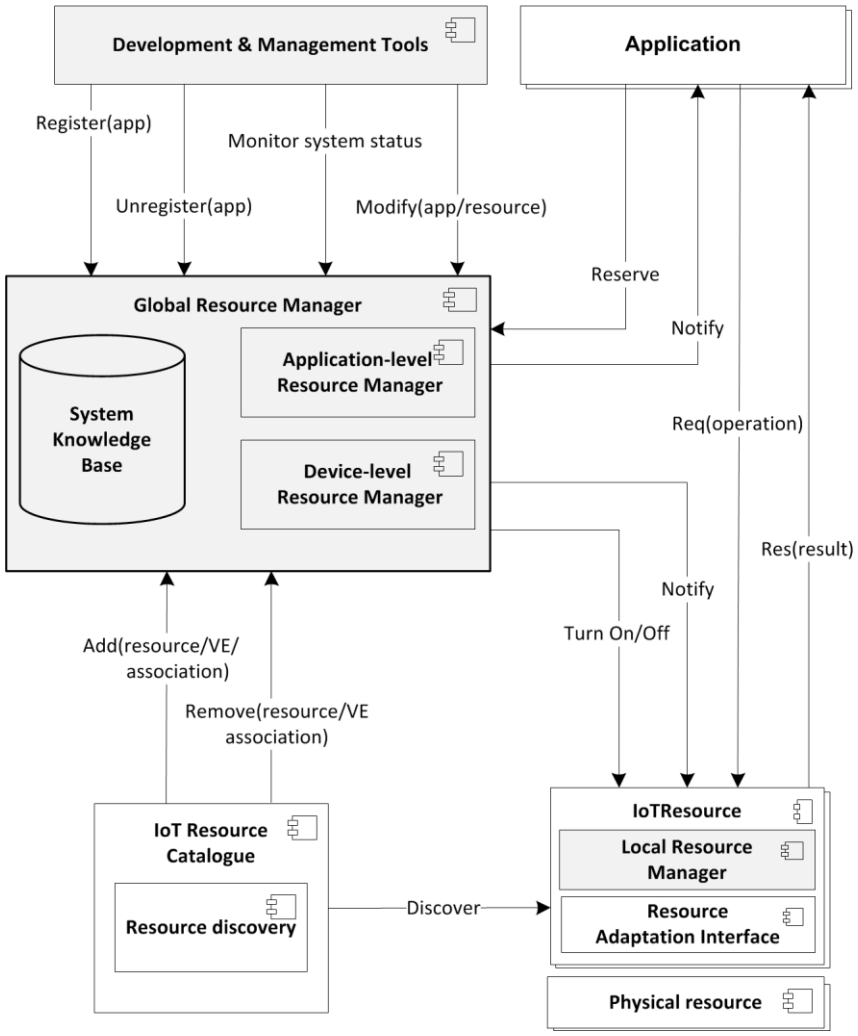


Figure 2. Mixed criticality resource management architecture. Grey components are responsible for realizing the mixed criticality resource management features.

Applications are software processes that provide a certain service for the user by utilising IoT resources (i.e., sensors and actuators) available in the given IoT environment. The IMPReSS SDP provides two options for the developers. The preferred way to create applications s to use the IMPReSS Context Management module and tools developed in the WP6 and WP7. However, it is also possible to write applications manually in any programming language. In either case

the application needs to utilise the mixed criticality middleware in order to be able to access the IoT resources provide by the IMPReSS platform.

In the IMPReSS architecture the component responsible for virtualising the resources and exposing the resource functionality for applications is called IoTResource. There is one IoTResource for each physical resource in the IoT system. The IoTResource consists of LRM and Resource Adaptation Interface (RAI) components of which the LRM is the one responsible for mixed criticality management aspects. The RAI component is described in the more detail in the deliverables IMPReSS deliverable D3.1. The interface provided by the LRM is presented in the Table 1. The LRM component is introduced in more detail in the section 3.3.

Table 1. Local Resource Manager interface.

| Operation | Description |
|---|---|
| **Authorize application** | This operation is used by the Global Resource Manager when it authorizes new application to access a resource. |
| **Deauthorize application** | This operation is used by the Global Resource Manager when it de-authorizes an application from using a resource. |
| **Request** | This is a generic envelope operation for all resource specific operations (e.g. get temperature, turn On/Off, etc.). The LRM processes the envelope and passes the actual operation to the RAI if the application is authorised to access the resource. |
| **Shut down** | This operation is used by the Global Resource Manager to shut down a device during power outage. |
| **Turn on** | This operation is used by the Global Resource Manager to turn on a device after a power outage. |

At the system level mixed criticality resource management is executed by the Global Resource Manager. Its main goal is to optimize the behaviour of the IoT system by providing resource management functionality at application and device levels. At the application-level the GRM discovers suitable resources for each application and controls which applications can access which resources in order to make sure that the behaviour of more critical applications is not compromised. At the device-level the role of the GRM is to make sure that more critical devices are supplied with power in the case of power shutdown. The device-level resource manager is based on the assumption that in the case of a power outage the devices are supplied with power from backup a battery or a generator. Whenever the available energy in the supply drops below a certain limit the application-level resource manager turns of devices with a criticality level below a predefined threshold. The Global Resource Manager interface is presented in the Table 2. The Global Resource Manager is introduced in more detail in the section 3.4.

Table 2. Global Resource Manager interface

| Operation | Description |
|---|---|
| **Register application** | Provides means to registers an application. The parameters include application ID and the application description. |
| **Unregister application** | Provides means to uninstall an application. The application ID parameter specifies the application to be unregistered. |
| **Add resource** | New resources are registered to the Global Resource Manager through this interface. |
| **Remove resource** | Obsolete resources are removed from the Global Resource Manager through this interface. |
| **Add virtual entity** | New virtual entities are registered to the Global Resource Manager through this interface. |
| **Remove virtual entity** | Obsolete virtual entities are removed from the Global Resource Manager through this interface. |
| **Add association** | New associations are registered to the Global Resource Manager through this interface. |

| | |
|---|---|
| **Remove association** | Obsolete associations are removed from the Global Resource Manager through this interface. |
| **Reserve resource** | Provides means to make a persistent reservation to a resource matching a resource specification (defined by resource specification ID). The GRM responds with a resource ID that matches the specification. The GRM will notify the client whenever a different resource is allocated for it. |
| **Release resource** | Provides means to release a resource for other (applications. Resource specification ID specifies the resource to be released. |
| **Notify power outage** | This interface is used to notify Global Resource Manager about a power outage. |
| **Notify power outage over** | This interface is used to notify Global Resource Manager that a power outage is over. |
| **Notify current energy level** | This interface is used to notify the Global Resource Manager whenever the energy level in the supply is changes. |
| **Set threshold** | Provides means to set criticality threshold(s) for device-level resource manager. A criticality threshold consists of available energy (percent) and criticality thresholds value pairs. |
| **Delete threshold** | Provides means to delete criticality thresholds from the device-level resource manager. |
| **Get thresholds** | Provides a list of defines criticality thresholds. |

In addition to the mixed criticality resource management middleware components, the architecture includes tools to help the development and management of mixed criticality applications. The tools are introduced in more detail in the section 3.5

## 3.2 Application and resource descriptions

### 3.2.1 Resource descriptions

One of the key design choices in the mixed criticality resource management approach is to represent the resource descriptions and specifications using Semantic Web technologies (Berners-Lee et al. 2001). This choice was made so that we can better address the challenge related to the heterogeneity of IoT resources and the need for future extendibility as IoT systems need to evolve and adjust to changes in the environment. Semantic Web technologies also provide very advanced query and pattern matching functionality which is important as the mixed criticality middleware needs to make it as easy as possible for the applications to access the resources available in the system.

In the IMPReSS project we divide the IoT resource representations into two parts: common model and domain model. The common model provides the overall framework and ontology for resource descriptions. It is based and aligned with the common architecture reference model, i.e., the IoT-A ARM (Bauer et al. 2013). The domain model in turn contains all domain specific parts of the IoT resource description. The resource management approach is agnostic to the used domain specific ontology and the only requirement is that the application developer utilizes the same ontology in the resource specifications. The common model ontology for IoT resources (and virtual entities) is illustrated in the Figure 3.

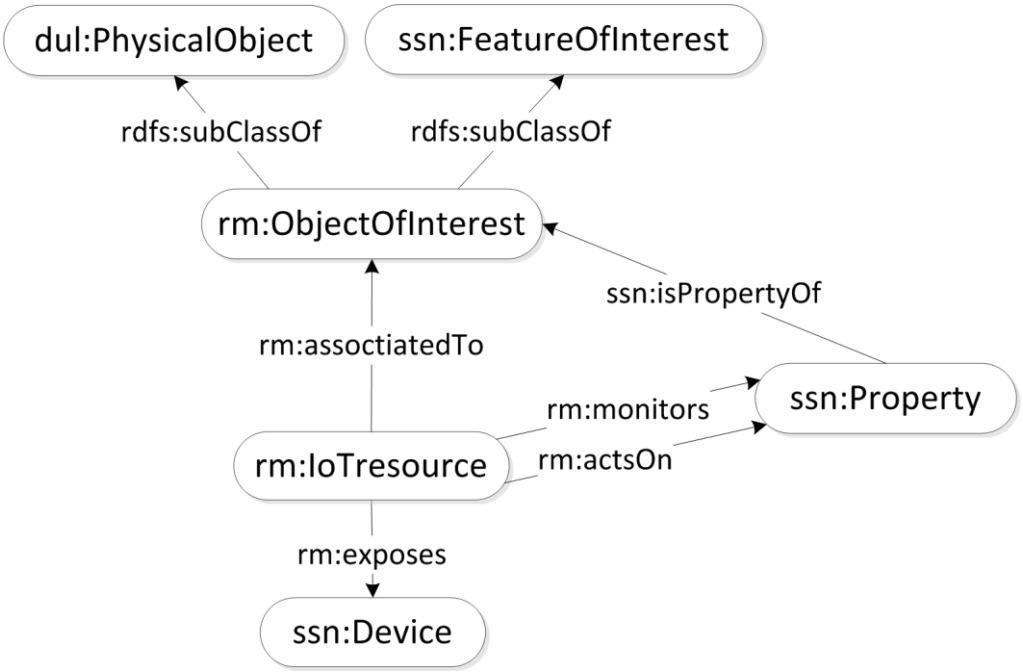


Figure 3. IoT resource ontology.

The IoT resource ontology has two main concepts: IoTresource and ObjectOfInterest. In the centre of all IoT systems are different kinds of physical world objects (e.g. rooms, items, appliances, products, people, etc.) that need to be somehow monitored and interacted with via the IoT technology. In the IoT resource ontology these physical world objects are instances of the class ObjectOfInterest (corresponds with the IoT Entity in the domain model). The corresponding entity for the ObjectOfInterest class in the IoT-A ARM domain model is the Virtual Entity (VE). In order to align the IMPReSS resource ontology with popular ontologies, the ObjectOfInterest class is modelled as a subclass of ssn:FeatureOfInterest (Compton et al. 2012) and dul:PhysicalObject (Gangemi 2007) classes. The idea is that whenever more domain specific information about the object of interest is needed any domain specific ontology can be utilised (or created) for this purpose.

The IoTresource class models a sensing or actuating capability of a device that is exposed for IoT applications. That is, it is the base class for all devices that are able to monitor and/or interact with the object of interests. When compared to the IoT-A ARM model the IMPReSS IoTresource maps directly to the IoT ARM Resource entity. The link between an IoTresource instance (e.g. temperature sensor) and an ObjectOfInterest instance (e.g. Freezer) is modelled with associatedTo object property. An IoTresource instance can either monitor or modify certain properties of the ObjectOfInterest; the link between the IoTresource instance and the ssn:Property instance are modelled with monitors and actsOn object properties. Again the idea is that whenever more specific description about the IoT resource is needed domain specific ontologies can be used for this purpose. Our resource management approach is totally agnostic to the used ontology. The only requirement is that the same ontology needs to be, of course, used in the resource specifications (inside the application descriptions) in order to be able discover the given resources.

An example instance of the IoT resource ontology is presented below. The example contains four ObjectOfInterest (or virtual entities); the classroom 10 and rows in the classroom 10. For simplicity reasons only four rows are used. For each row there is an occupancy sensor that monitors whether people are present in the row. Each row also has its own light resource (PhilipsHue bulb). Again for the sake of clarity only one bulb is used for each row. In addition to the simple occupancy sensor there is also a Kinect sensor in the room that can provide occupancy data. The same sensor is associated with all the rows in the classroom.

```
@prefix rm: <http://purl.oclc.org/impress/resource# .

# Classroom 10.
<urn:classRoom10> a re:Place, re:ObjectOfInterest;
                    re:name   "Classroom 10" .

# Four rows of the classroom.
<urn:row1inClassRoom10> a re:Place, re:ObjectOfInterest;
                        re:name "Row 1 in classroom 10" .

<urn:row2inClassRoom10> a re:Place, re:ObjectOfInterest;
                        re:name "Row 2 in classroom 10" .

<urn:row3inClassRoom10> a re:Place, re:ObjectOfInterest;
                        re:name "Row 3 in classroom 10" .

<urn:row4inClassRoom10> a re:Place, re:ObjectOfInterest;
                        re:name "Row 4 in classroom 10" .

# Four occupancy sensors. One associated for each row in the classroom.
<http://purl.oclc.org/impress/rai/occupancysensor1> a re:OccupancySensor,
                                                      re:Resource;
                                                re:name "Occupancy sensor nro 1";
                                                re:associatedTo <urn:row1inClassRoom10> .

<http://purl.oclc.org/impress/rai/occupancysensor2> a re:OccupancySensor,
                                                      re:Resource;
                                                re:name "Occupancy sensor nro 2";
                                                re:associatedTo <urn:row1inClassRoom10> .

<http://purl.oclc.org/impress/rai/occupancysensor3> a re:OccupancySensor,
                                                      re:Resource;
                                                re:name "Occupancy sensor nro 3";
                                                re:associatedTo <urn:row1inClassRoom10> .

<http://purl.oclc.org/impress/rai/occupancysensor4> a re:OccupancySensor,
                                                      re:Resource;
                                                re:name "Occupancy sensor nro 4";
                                                re:associatedTo <urn:row1inClassRoom10> .

<http://purl.oclc.org/impress/rai/philipsHue1> a re:PhilipsHue,
                                                 re:LightingSystem,
                                                 re:Resource;
                                           re:name "PhilipsHue nro 1";
                                           re:associatedTo <urn:row1inClassRoom10>.

<http://purl.oclc.org/impress/rai/philipsHue2> a re:PhilipsHue,
                                                  re:LightingSystem,
                                                  re:Resource;
                                           re:name " PhilipsHue nro 2";
                                           re:associatedTo <urn:row1inClassRoom10>.

<http://purl.oclc.org/impress/rai/philipsHue3> a re:PhilipsHue,
                                                 re:LightingSystem,
                                                 re:Resource;
                                           re:name " PhilipsHue nro 3";
                                           re:associatedTo <urn:row1inClassRoom10>.

<http://purl.oclc.org/impress/rai/philipsHue4> a re:PhilipsHue,
                                                 re:LightingSystem,
                                                 re:Resource;
                                           re:name " PhilipsHue nro 4";
                                           re:associatedTo <urn:row1inClassRoom10>.

# Kinect sensor (provides also occupancy data) associated to every row in the classroom.
<http://purl.oclc.org/impress/rai/kinectsensor> a re:KinectSensor,
                                                  re:OccupancySensor,
                                                  re:Resource;
                                              re:name "Kinect sensor";
                                              re:associatedTo <urn:row1inClassRoom10>,
                                                              <urn:row2inClassRoom10>
                                                              <urn:row3inClassRoom10>
                                                              <urn:row4inClassRoom10>
```

### 3.2.2 Application descriptions

Application description represents relevant information about IoT applications running on top of the IMPReSS mixed criticality middleware. The information available in the application descriptions is used for following purposes:

1. Find and select suitable IoT resources for each application so that the behaviour of the whole IoT system can be optimised.
2. Ensure that more critical applications get access to resources over less critical ones (exclusive access scheme).
3. Schedule the resource access so that more critical applications are served before less critical ones (shared access scheme).
4. Ensure that only trustworthy applications can access confidential resources.
5. Visualise the IoT system status in terms of applications, resources and associations between them.

The application description is serialised with JSON and SPARQL 1.1 syntax (W3C 2013a) is used for representing the functional specifications of resources. The Table 3 presents the parameters of the applications description. The parameters for resource specifications in turn are presented in the Table 4.

Table 3. Application description parameters.

| Parameter | Possible values | Description |
|---|---|---|
| **application ID** | String (globally unique) | Unique identifier of the application. |
| **application criticality** | Number (non-negative integer) | Specifies the criticality of the application. |
| **application name** | String | Short name of the application. |
| **description** | String | Description of the application. |
| **resources** | List of resource specification objects | List of resource specifications. The parameters of resource specification are presented in the table below. |

Table 4. Resource specification parameters.

| Parameter | Possible values | Description |
|---|---|---|
| **resource specification ID** | String (unique within the application description) | Unique identifier of the resource specification. |
| **access scheme** | String (either Shared or Exclusive) | Defines whether the resource access is exclusive or shared. |
| **significance** | String (either obligatory or useful) | Defines the importance of the resource for the application. If the application cannot function without the resource use "obligatory", otherwise use "useful". |
| **query** | String (SPARQL SELECT query with single query variable ?resource) | Domain specific specification of the resource using SPARQL SELECT query syntax. In principle the only requirement is that same ontology is used as for resource descriptions. |

An example of the application description is provided below. It represents an energy saver application that controls the lights in a classroom in order to save energy. That is, lights are turned on only when there is a person present in the row. The application has eight resource specifications one for occupancy sensor and lighting system associated to each row in the class room (only four rows are used in the example).

```
{"application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634",
 "application criticality": 100,
 "application name": "Energy saver app",
 "description": "The application controls lights in order to save energy.",
 "resources": [
          {"resource specification ID" : "1",
            "access scheme": "Shared",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:OccupancySensor ;
                                         rm:associatedTo <urn:row1inClassRoom10>.}"
          },
          {"resource specification ID" : "2",
            "access scheme": "Shared",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:OccupancySensor ;
                                         rm:associatedTo <urn:row2inClassRoom10>.}"
          },
          {"resource specification ID" : "3",
            "access scheme": "Shared",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:OccupancySensor ;
                                         rm:associatedTo <urn:row3inClassRoom10>.}"
          },
          {"resource specification ID" : "4",
            "access scheme": "Shared",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:OccupancySensor ;
                                         rm:associatedTo <urn:row4inClassRoom10>
          },
          { "resource specification ID" : "5",
            "access scheme": "Exclusive",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:LightingSystem ;
                                         rm:associatedTo <urn:row1inClassRoom10>.}"
          },
          { "resource specification ID" : "6",
            "access scheme": "Exclusive",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:LightingSystem ;
                                         rm:associatedTo <urn:row2inClassRoom10>.}"
          },
          { "resource specification ID" : "7",
            "access scheme": "Exclusive",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:LightingSystem ;
                                         rm:associatedTo <urn:row3inClassRoom10>.}"
          },
          { "resource specification ID" : "8",
            "access scheme": "Exclusive",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:LightingSystem ;
                                         rm:associatedTo <urn:row4inClassRoom10>.}"
          },
        ]
}
```

### 3.3 Local Resource Manager

At local level the resource management is performed by the LRM. The role of the LRM is twofold. At the application-level it 1) controls that only applications that are authorised by the GRM access the given resources and 2) schedules that request send by the applications (shared access scheme) so that the most critical applications are served first. At the device-level it provides an interface for Global Resource Manager to control which devices are provided with power in the case of a power outage.

The LRM is implemented with Java programming language using Jersey[5] RESTful Web Services framework. The internal architecture of the IoTResource component (including the LRM) is presented in the Figure 4.

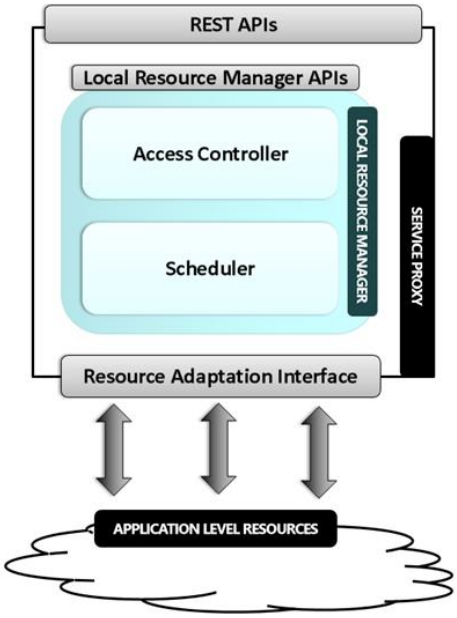


Figure 4. IoTResource architecture.

The Local Resource Manager provides a HTTP interface presented Table 5 (as part of the IoTResource interface) for the Global Resource Manager to control its operation.

Table 5. Local Resource Manager HTTP interface.

| Method | URI | Payload type |
|---|---|---|
| **POST** | /<resource_id>/authorize_access | A |
| **POST** | /<resource_id>/deauthorize_access | B |
| **GET** | /<resource_id>/turnOff | None |
| **GET** | /<resource_id>/turnOn | None |

The parameters for authorise access request are presented in the Table 6.

Table 6. Parameters for payload type A.

| Parameter | Type | Description |
|---|---|---|
| **appID** | String | ID of the App that is authorized to access the Resource |

[5] https://jersey.java.net/

| | | |
|---|---|---|
| **criticality** | Integer | Level of priority assigned to the application. This is used to manage concurrent access requests to a Resource. |
| **securityLevel** | String | Level of security of the communication channel between RAI and the Resource. |

An example of the authorise access request is presented below:

```
Request:
POST http://130.192.85.32:8080/f061b5da-7ddd-3510-95fc-15d61e870f26/authorize_access

Content-Type: application/json

Payload:
{
    "appID" : "4ad26797-7bb9-470a-b2a7-17533ae42f48",
    "priority" : 100,
    "securityLevel" : "Medium"
}
```

The parameters of the de-authorise access request are presented in the Table 7.

Table 7. Parameters for payload type B.

| Parameter | Type | Description |
|---|---|---|
| **appID** | String | ID of the App that is authorized to access the Resource |

An example of the deauthorise access request is presented below:

```
Request:
POST http://130.192.85.32:8080/f061b5da-7ddd-3510-95fc-15d61e870f26/deauthorize_access

Content-Type: application/json

Payload:
{
    "appID" : "4ad26797-7bb9-470a-b2a7-17533ae42f48"
}
```

### 3.4 Global Resource Manager

The Global Resource Manager internal architecture is depicted in the Figure 5. It consists of three functional software components, called Application-level Resource Manager, Device-level Resource Manager and System Knowledge Base, and three interface modules called Resource Catalogue Interface, Global Resource Manager Protocol and System Knowledge Base Protocol. The Application-level Resource Manager and Device-level Resource Manager components are described in more detail in the chapters 4 and 5. The other components are introduced in sections 3.4.1-3.4.3.

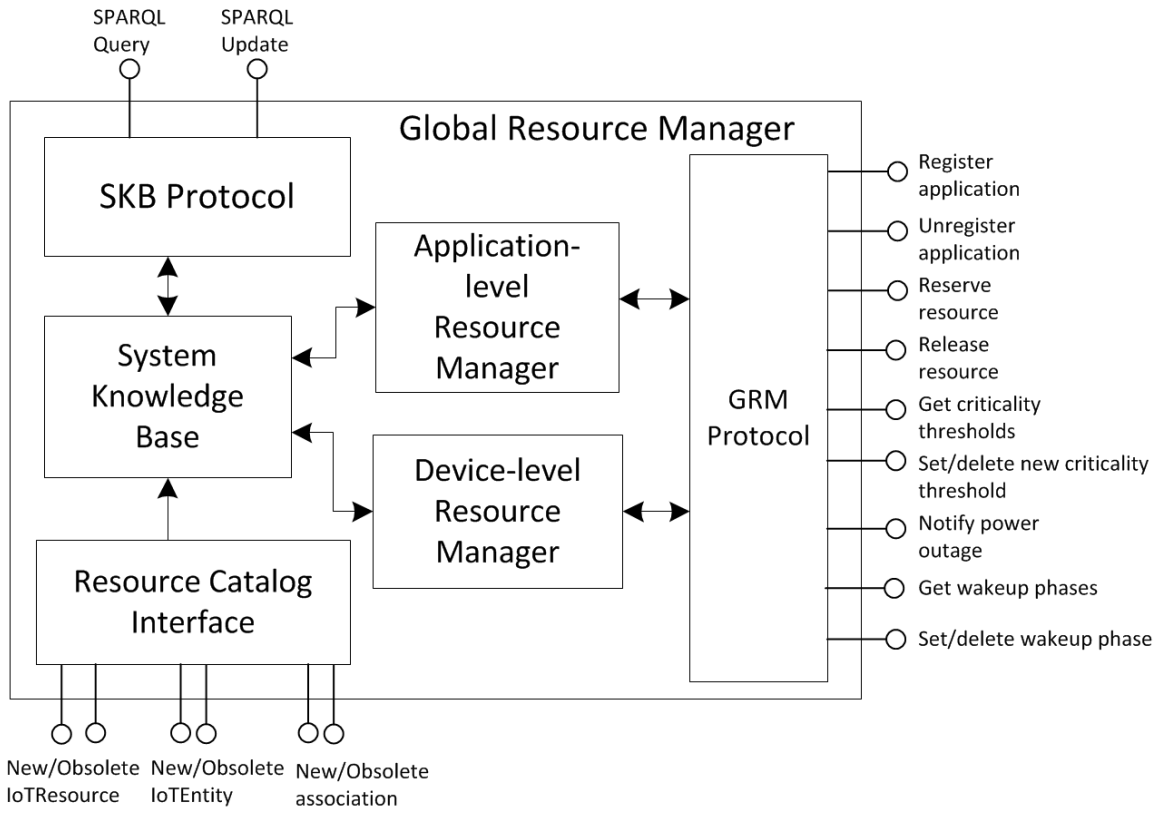


Figure 5. Global Resource Manager internal software architecture.

### 3.4.1 Global Resource Manager Protocol

The role of the Global Resource Manager Protocol module is to provide an interface for applications to access the resource management services, as well as, an interface for Application-level Resource Manager and Device-level Resource Manager components to control the Local Resource Manager. The Global Resource Manager Protocol module is implemented with Python programming language. In order to provide full-duplex communication channel for the applications we utilise Websockets (Fette and Melnikov 2011) for communication. The Websocket interface is implemented on top of Autobahn websocket library[6]. The Autobahn-framework in turn is based on an event-driven networking engine, called Twisted, and for this reason the Global Resource Manager Protocol is based on event-driven programming paradigm.

The Global Resource Manager Protocol reacts to two types of events. First, messages sent by the applications and IoT resources need to be forwarded to the Application-level Resource Manager and the Device-level Resource Manager, respectively. Second, the applications and IoT resources need to be notified about the events created by the Application-level Resource Manager and the Device-level Resource Manager whenever the status of the reservations changes or some of the IoT resources need be shut down due to the power outage.

The Global Resource Manager Protocol utilises following JSON based message format on top of Websockets:

```
[operation type, message type, {dataObject}]
```

6

The *operationType* is an integer between (0-9 reserved for application-level resource manager and 10- for device-level resource manager) and defines the type of the operation. Possible operation types are presented in the Table 8.

Table 8. Global Resource Manager Protocol – Operation types.

| Operation type | Description | dataObject type |
|---|---|---|
| ***0*** | *Reserve resource:* Sent by application to reserve resource. This operation is persistent and the application needs to terminate the operation with the release resource operation. | A |
| **1** | *Release resource*: Sent by application to release reserved resource and terminate the persistent resource reservation operation. | B |
| **2** | *Register application*: Sent by application deployment tool (or application) to register an application. | C |
| **3** | *Unregister application*: Sent by application deployment tool (or application) to unregister an application | D |
| **10** | *Power outage*: Sent by IoT resource to notify GRM about a power outage. | E |
| **11** | *Power outage over*: Sent by IoT resource to notify GRM that the power outage is over. | F |
| **12** | *Set threshold:* Sent by the Management Tool to add new device-level mixed criticality threshold. | G |
| **13** | *Delete threshold:* Sent by the Management Tool to delete a device-level mixed criticality threshold. | H |
| **14** | *Get threshold:* Sent by the Management Tool to get a list device-level mixed criticality thresholds. | I |
| **15** | *Set wakeup:* Sent by the Management Tool to add new device wakeup phase. | J |
| **16** | *Delete wakeup:* Sent by the Management Tool to delete a wakeup phase. | K |
| **17** | *Get wakeup phases:* Sent by the Management Tool to retrieve a list of wakeup phases configured for the Device-level Resource Manager. | L |

The *message type* is an integer between 0 and 2 that defines the type of the message. Possible message types are presented in the Table 9.

Table 9. Global Resource Manager Protocol – Message types.

| Message type | Description |
|---|---|
| **0** | *Request*: Sent by application or application deployment tool to GRM. |
| **1** | *Response*: Immediate response for request-message. Sent by GRM to application or application distribution tool. |
| **2** | *Notification*: Sent by GRM to notify application about the changes in resources e.g. resource matching a specification available and reserved for the application. |

DataObject is JSON Object storing the actual data about the message. The content of the dataObject field depends on the operation and message types. In the Table 10 - Table 18 the parameters for different type dataObject types are presented. In order to make the structure of the messages more clear each table is also followed by an example of the given message.

**Reserve resource messages (dataObject A)**

Table 10. Request message parameters for dataObject type A (Reserve resource).

| Parameter | Possible values | Description |
|---|---|---|
| **application ID** | String (globally unique) | Unique identifier of the application. |
| **resource specification ID** | String (unique within the application description) | Unique identifier of the resource specification. |
| **resource count (optional)** | Number (positive): Default value is one. | Number of resources requested by the app. Default value is one. |

```
[0, 0, {"application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634",
        "resource specification ID": "1",
        "resource count": 2}]
```

Table 11. Response message parameters for dataObject type A (Reserve resource).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |
| **resource specification ID** | String (unique within the application description) | Unique identifier of the resource specification. |
| **resource ID** | Resource ID list | List of resource IDs reserved for the application. |

```
[0, 1, {"status": 0,
        "resource specification ID": "1",
        "resource ID": [
              "http://purl.oclc.org/impress/rai/occupancysensor54",
              "http://purl.oclc.org/impress/rai/kinectsensor1"]}]
```

Table 12. Notification message parameters for dataObject type A (Reserve resource)

| Parameter | Possible values | Description |
|---|---|---|
| **resource specification ID** | String (unique within the application description) | Unique identifier of the resource specification. |
| **resource ID** | String (URI of the resource) or null if no suitable resource is available. | ID of a new resource reserved for the application. |
| **old resource ID** | String (URI of the resource) or null if no resource were removed or replaced. | ID of a resource that is not anymore reserved for the application. |

```
[0,2,{"resource specification ID": "identifier",
      "resource ID": null,
      "old resource ID": "http://purl.oclc.org/impress/rai/occupancysensor54"}]
```

***Release resource* messages (dataObject type B)**

Table 13. Request message parameters for dataObject type B (Release resource).

| Parameter | Possible values | Description |
|---|---|---|
| **application ID** | String (globally unique) | Unique identifier of the application. |
| **resource specification ID** | String (unique within the application description) | Unique identifier of the resource specification. |

```
[1, 0, {"application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634",
```

```
        "resource specification ID": "1"}]
```

Table 14. Response message parameters for dataObject type B (Release resource).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |
| **resource specification ID** | String (unique within the application description) | Unique identifier of the resource specification. |

```
[1, 1, {"status": 0,
        "resource specification ID": "1"}]
```

**Register application messages (dataObject type C)**

Table 15. Request message parameters for dataObject type C (Register application).

| Parameter | Possible values | Description |
|---|---|---|
| **application ID** | String (globally unique) | Unique identifier of the application. |
| **application name** | String | Short name of the application. |
| **description** | String | Description of the application. |
| **application criticality** | Number (non-negative integer) | Specifies the criticality of the application. |
| **resources** | List of resource specification objects | List of resource specifications. The parameters of resource specification are presented in the section 3.2. |

```
[2, 0, {"application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634",
        "application criticality": 100,
        "resources": [
          {"resource specification ID" : "1",
            "access scheme": "Shared",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:OccupancySensor ;
                                       rm:associatedTo <urn:row1ClassRoom10>.}"
          },
          { "resource specification ID" : "2",
            "access scheme": "Exclusive",
            "significance": "Obligatory",
            "query" : "PREFIX rm: <http://purl.oclc.org/impress/resource#>
                       SELECT ?resource
                       WHERE { ?resource a rm:LightingSystem ;
                                        rm:associatedTo <urn:row1ClassRoom10>.}"
          }
       ]}
]
```

Table 16. Response message parameters for dataObject type C (Register application).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |
| **application ID** | String (globally unique) | Unique identifier of the application. |

```
[2, 1, {"status": 0, "application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634"}]
```

***Unregister application* messages (dataObject type D)**

Table 17. Request message parameters for dataObject type D (Unregister application).

| Parameter | Possible values | Description |
|---|---|---|
| **application ID** | String (globally unique) | Unique identifier of the application. |

```
[3, 0, {"application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634"}]
```

Table 18. Response message parameters for dataObject type D (Unregister application).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |
| **application ID** | String (globally unique) | Unique identifier of the application. |

```
[3, 1, {"status": 0, "application ID": "7d1331c0-cf29-4d52-8089-9e0418dcc634"}]
```

**Power outage messages (dataObject type E)**

The power outage request messages no dot have a payload (i.e., dataObject field). An example of the power outage request message is presented below.

```
[10, 0, {}]
```

Table 19. Response message parameters for dataObject type E (Power outage).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |

```
[10, 0, {"status": 0}]
```

**Power outage over messages (dataObject type F)**

The power outage over request messages no dot have a payload (i.e., dataObject field). An example of the power outage over request message is presented below.

```
[11, 0, {}]
```

Table 20. Response message parameters for dataObject type F (Power outage over).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |

```
[11, 0, {"status": 0}]
```

**Set threshold messages (dataObject type G)**

Table 21. Request message parameters for dataObject type G (Set threshold).

| Parameter | Possible values | Description |
|---|---|---|
| **energy level** | Integer between 0 to 100 | Specifies an energy level in terms of how many percent is left in the battery or generator. If this energy level is reached all the devise below the criticality level defined in the second parameter are turn off. |
| **criticality level** | Positive integer | Specifies a criticality threshold. If the energy level specified in the first parameter is reached all the devices below this level are turned off. |

```
[12, 0, {"energy level": 75, "criticality level": 150}]
```

Table 22. Response message parameters for dataObject type G (Set threshold).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |

```
[12, 0, {"status": 0}]
```

**Delete threshold messages (dataObject type H)**

Table 23. Request message parameters for dataObject type H (Delete threshold).

| Parameter | Possible values | Description |
|---|---|---|
| **energy level** | Integer between 0 to 100 | Specifies an energy level in terms of how many percent is left in the battery or generator. If this energy level is reached all the devise below the criticality level defined in the second parameter are turn off. |
| **criticality level** | Positive integer | Specifies a criticality threshold. If the energy level specified in the first parameter is reached all the devices below this level are turned off. |

```
[13, 0, {"energy level": 75, "criticality level": 150}]
```

Table 24. Response message parameters for dataObject type H (Delete threshold).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |

```
[13, 0, {"status": 0}]
```

**Get thresholds messages (dataObject type I)**

The get thresholds request message does not have parameters (i.e., dataObject). An example of the request message is given below:

```
[14, 0, {}]
```

Table 25. Response message parameters for dataObject type I (Get thresholds).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |
| **levels** | DataObject list. | A list of individual energy level criticality level pairs. |

```
[14, 0, {"status": 0, "levels": [{"energy level": 75, "criticality level": 100},
                                 {"energy level": 50, "criticality level": 200},
                                 {"energy level": 25, "criticality level": 300}
                                ]
        }
]
```

**Set wakeup messages (dataObject type J)**

Table 26. Request message parameters for dataObject type J (Set wakeup)

| Parameter | Possible values | Description |
|---|---|---|
| **criticality level** | Positive integer | Specifies a criticality threshold. |
| **wait time** | Positive integer (seconds) | Specifies in seconds the time that is waited before devices below the criticality level are turned back on. |

```
[15, 0, {"criticality level": 150, "wait time": 15}]
```

Table 27. Response message parameters for dataObject type J (Set wakeup).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |

```
[15, 0, {"status": 0}]
```

**Delete wakeup messages (dataObject type K)**

Table 28. Request message parameters for dataObject type K (Delete wakeup).

| Parameter | Possible values | Description |
|---|---|---|
| **criticality level** | Positive integer | Specifies a criticality threshold. |
| **wait time** | Positive integer (seconds) | Specifies in seconds the time that is waited before devices below the criticality level are turned back on. |

```
[16, 0, {"criticality level": 150, "wait time": 15}]
```

Table 29. Response message parameters for dataObject type K (Delete wakeup).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |

```
[16, 0, {"status": 0}]
```

**Get wakeup phases messages (dataObject type L)**

The get wakeup request message does not have parameters (i.e., dataObject). An example of the request message is given below:

```
[17, 0, {}]
```

Table 30. Response message parameters for dataObject type L (Get wakeup phases).

| Parameter | Possible values | Description |
|---|---|---|
| **status** | 0 if request was understood by GRM, otherwise -1 | Status of the operation. |
| **phases** | DataObject list. | A list of wakeup phases. |

```
[17, 0, {"status": 0, "phases":[{"criticality level": 240, "wait time": 0},
                                {"criticality level": 0, "wait time": 30}
                               ]
}]
```

### 3.4.2 System Knowledge Base and SKB Protocol

The role of the System Knowledge Base is to act as an internal database for the Global Resource Manager components. That is, it used to store information about applications, IoT Resources, IoT Entities and associations in Resource Description Framework (RDF) (W3C 2014) format. The SKB reference implementation is based on Smart-M3 Semantic Information Broker called Red-SIB[7], which is available as open-source software. The Red-SIB is a RDF-database which provides clients, called Knowledge Processors (KPs), with publish-subscribe based interface to access and manipulate RDF data. The publish-subscribe architecture decouples efficiently the different components that need to access and manipulate date.The communication between the Red-SIB and the KP is executed with proprietary protocol called Smart Space Access Protocol (SSAP). In our proof-of-concept implementation the Global Resource Manager communicates with the Red-SIB directly by using the SSAP. In practise this interface is implemented with a Python-library called m3_kp_api.

In order to provide more standard and widely supported interface for external clients and tools (e.g. Web browser) we have implemented a Websocket-interface for the SKB on top of the Red-SIB SSAP communication. As already mentioned above Websocket provides a full-duplex communication channels and is therefore feasible for implementing the publish-subscribe pattern. For the messages we have designed a simple JSON message format with three parameters as follows:

```
[operation type, message type, transaction ID, payload]
```

The operation type is an integer between 0 and 2. The possible operation types are presented in the Table 31. The transaction ID is a Universally Unique Identifier (UUID) that identifies each request and corresponding response/notifications messages. It is used to map the response and notification messages to the right request (i.e., the same id used in the request is used in the response and notification messages related to that request). The payload in the request message is presented with SPARQL 1.1 Query/Update language syntax (W3C 2013a) (W3C 2013b). In the response and notification message the payload is JSON serialised SPARQL results as presented in the W3C Recommendation (W3C 2013c).

Table 31. SKB Protocol – Operation types.

| Operation type | Description |
|---|---|
| **0** | *Query*: Provides means to query the status of the system using SPARQL 1.1 SELECT queries. |
| **1** | *Subscribe*: Provides means to monitor the system status by performing persistent SPARQL 1.1 SELECT queries. The Red-SIB will create events to the Twisted engine whenever the results of the subscription operation change. The results are encapsulated in SPARQL JSON format and send to the client. |
| **2** | *Update:* Provides means to modify the system status. The exact operation to be performed is presented in SPARQL 1.1 Update syntax. |
| **3** | *Unsubscribe*: Terminates a subscription operation. Transaction ID of the subscription to be terminated is defined in the payload. |

Table 32. SKB Protocol – Message types.

| Message type | Description |
|---|---|
| **0** | *Request*: Used in the messages sent by the client. |
| **1** | *Response*: Immediate response for a request-message. |
| **2** | *Notification*: Used when the results of a SPARQL subscription operation are |

[7] http://sourceforge.net/projects/smart-m3/

| modified. |
|---|

### 3.4.3 Resource Catalogue Interface

As the name implies the role of the Resource Catalogue Interface component is to provide interface for Resource Catalogue to update information about the system status into the Global Resource Management components. This includes information about the IoT resources, IoT Entities (i.e., object of interests) and associations between them. The communication between the Global Resource Manager and the Resource Catalogue is executed on top of MQTT. Consequently, the Resource Catalogue Interface contains an MQTT client that that is subscribed. The MQTT client is implemented with python programming language using Paho-mqtt 1.1[8] library. The MQTT topics used for Resource Catalog - Global Resource Manager interaction are presented in the Table 33.

Table 33. MQTT topics for resource, virtual entity and association notifications.

| Topic | Description | Payload type |
|---|---|---|
| **IMPRESS/System/NewIoTResource** | Topic for new IoT Resource notifications | Payload A |
| **IMPRESS/System/ObsoleteIoTResource** | Topic for obsolete IoT Resource notifications | Payload B |
| **IMPRESS/System/NewIoTEntity** | Topic for new IoT Entity notifications | Payload C |
| **IMPRESS/System/ObsoleteIoTEntity** | Topic for obsolete IoT Entity notifications | Payload D |

The different payload types for the MQTT topics are presented in the Table 35 - Table 37. As serialisation format we utilise JSON. An example of each payload is given after each table to further clarify the payload structure.

Table 34. Resource description (Payload A)

| Parameter | Possible types | Description |
|---|---|---|
| **resource id** | Dereferencable URI | Unique ID of the resource. |
| **type** | String list | A list of types (classes) resource belongs to. This is used to classify resources into certain categories. |
| **friendly name** | String | Human readable description of the resource. |

```
{
     "resource id": "http://purl.oclc.org/impress/rai/kinectsensor_1",
     "type": ["OccypancySensor", "KinectSensor"],
     "friendly name": "Kinect sensor"
}
```

Table 35. Resource description ID (Payload B)

| Parameter | Possible types | Description |
|---|---|---|
| **resource id** | Dereferencable URI | Unique ID of the resource. |

```
{"resource id": "http://purl.oclc.org/impress/rai/kinectsensor_1"}
```

[8] https://pypi.python.org/pypi/paho-mqtt/1.1

Table 36. Entity description (Payload C)

| Parameter | Possible types | Description |
|---|---|---|
| **entity id** | Any unique string | Unique ID of the IoT Entity. |
| **type** | String list | The type of the entity (e.g. place, person, device). |
| **friendly name** | String | Human readable description of the IoT Entity. |

```
{
     "entity id": "urn:uuid:f47ac10b-58cc-4372-a567-0e02b2c3d479",
     "type": "Place",
     "friendly name": "Row 10 in the classroom 10"
}
```

Table 37. IoT Entity ID (Payload D)

| Parameter | Possible types | Description |
|---|---|---|
| **entity id** | Any unique string | Unique ID of the IoT Entity. |

Whenever the Resource Catalogue Interface receives new MQTT event is parses the parameters from the JSON payload, transforms the data into semantic format modelled according the ontology presented in the section 10, and then updates the corresponding information (i.e., resource description, association description, association, etc.) into the SKB using the *m3_kp_ap* python library. This way the Global Resource Manager has all the time up to date view of the IoT resources, IoT entities and their associations.

## 3.5 Tools for mixed criticality resource management

The mixed criticality resource management framework provides also two tools; one for the application developer and one for the system administrator/maintenance personnel. The Application description generator tool reference implementation is introduced in the section 3.5.1. The Mixed criticality resource management tool in turn is briefly described in the section 3.5.2.

### 3.5.1 Application Description Generator Tool

The Application description generator tool (illustrated in Figure 6) provides developers with simple means to create the application descriptions so that they do not need to be familiar with the application description syntax presented in section 3.2.2. Similarly to the other IMPReSS tools the Application Description Generator is a Web-based tool. It is implemented with Java programming on top of AngularJS[9]. AngularJS is an open source JavaScript-based web framework that has been designed to support single-page applications (SPAs). Applying a MVC pattern AngularJS enables developers to separate the application logic from the DOM. The application logic is handled should be handled so called 'controllers' and further so called 'scope' objects need to be defined which act as a glue to share things between the controller and the view. In order to manipulate the Document Object Model (DOM) AngularJS offers developers the use of so called 'directives'. For further technical details the reader is asked to read the official documentation at https://docs.angularjs.org/guide .

[9] https://angularjs.org/.

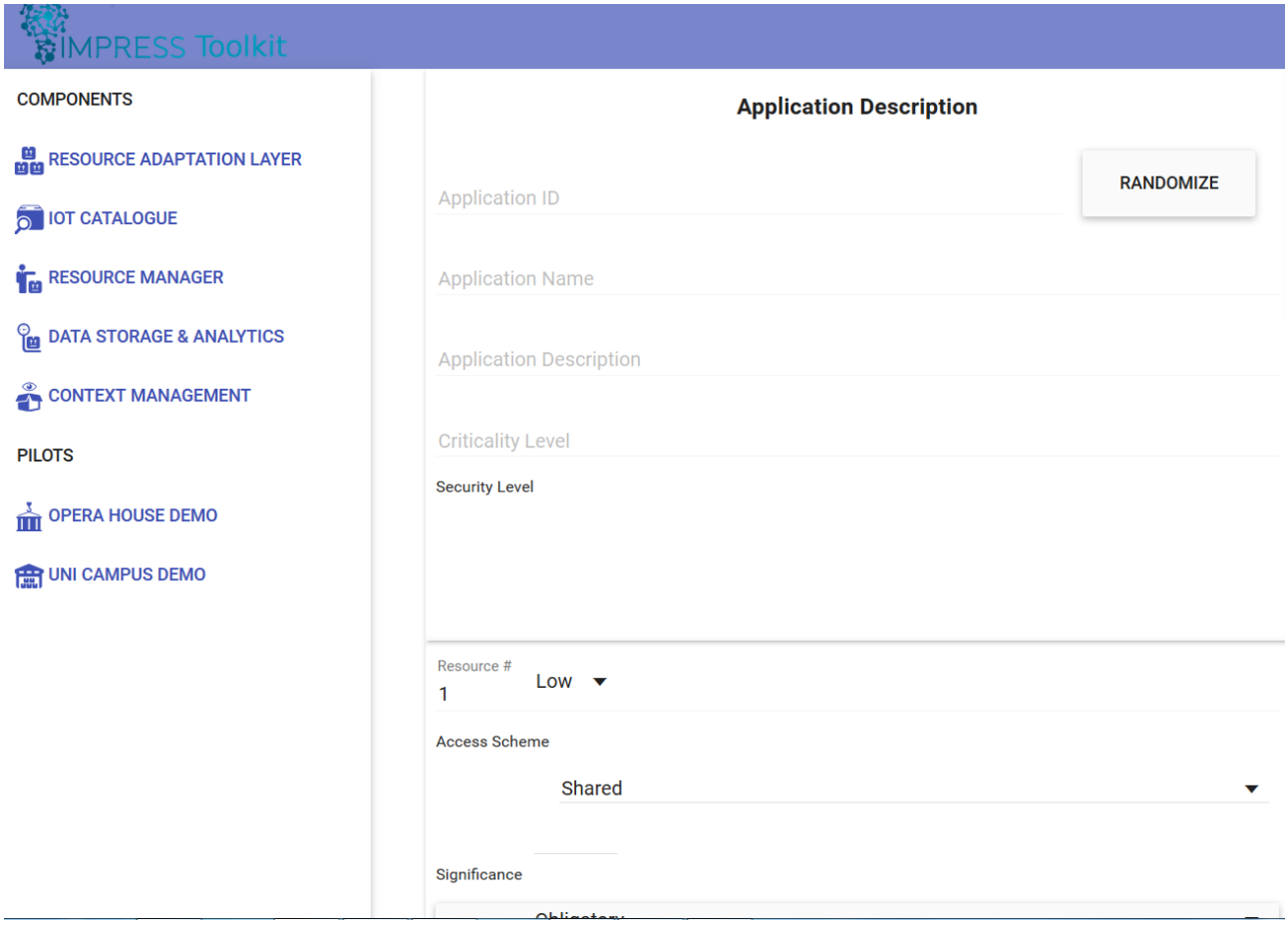


Figure 6. Snapshot of the Application Description Generator tool user interface.

More information about the Application Description Generator Tool is provided in the chapter 4 where practical example is used to demonstrate how the tool works in practise.

### 3.5.2 Mixed Criticality System Management Tool

The Mixed Criticality System Management Tool provides system administrators, integrators and maintenance personnel with means to monitor and manage their IMPReSS based IoT systems. Similarly to the Application Description Generator Tool the Mixed Criticality System Management Tool provides a Web-based user interface as illustrated in the Figure 7. The tool is implemented with JavaScript and D3.js[10] visualisation framework.

[10] http://d3js.org/

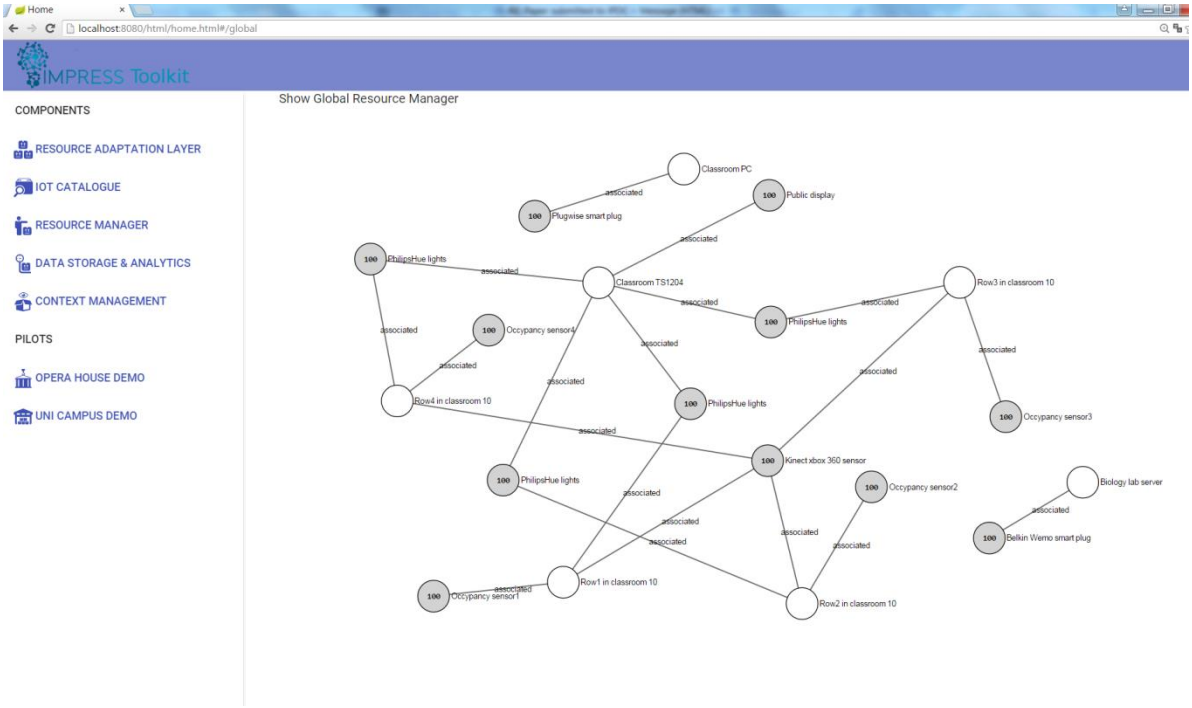


Figure 7. IoT Resources and Entity view of the Mixed Criticality System Management Tool.

The tool can be used for both application and device-level mixed criticality resource management. At the application-level the tool visualises the system status in terms of applications, IoT resources and IoT entities. The tool also visualises which application is using which resource at the moment and displays the associations between IoT resources and entities. It is also possible to adjust the criticality of applications and unregister applications when necessary. At the device-level the tool provides means to adjust the criticality of IoT resources and to manage criticality thresholds set for the Device-level Resource Manager. More detailed examples of using the tool in device-level mixed criticality resource management is presented in the chapter 5.

# 4. Application-level resource management

In this section the application-level resource management features are introduced in more detail. In order to concretise the mixed-criticality resource management technologies practical examples are used to demonstrate the approach in three phases: development, deployment and run-time. The development phase is described in section 4.1. Section 4.1.2 describes the functionality of the mixed criticality resource management architecture during deployment and run-time phases. It should be noted that although these phases are sequential from individual application point of view, for the IMPReSS mixed criticality middleware all these phases occur in parallel since the idea is that new applications and IoT Resources can be developed and deployed during run-time of the whole IMPReSS system.

As an example application the Energy saver application introduced in the section 3.2.2 is used thorough this section.

## 4.1 Development phase

For the development phase the IMPReSS mixed criticality middleware provides 3rd party application developers with 1) a Python API library, which makes it easier to access the Global Resource Manager and 2) an Application Description Generator Tool that can be used to generate application descriptions on the behalf of developers. A short introduction to the Application Description Generator Tool is presented in the section 4.1.1. A brief introduction to application development with the Python API is presented in the section 4.1.2.

It should be noted that if the developer does not use Python programming language (or does not want to use the API for some other reason) they can also access directly the Global Resource Manager WebSocket/JSON interface described in the section 3.4.1. Also the application descriptions can be written manually using the specification described in the section 3.2.2.

### 4.1.1 Application Description Generator Tool

When making a new application description with the Application Description Generator Tool the developer needs to first provide generic information about their application as presented in the Figure 8. This generic part includes the ID (can be also generated by pressing the randomize button), name, description, criticality and security level[11] of the application. In the picture this part of the application description creation is highlighted with the redlined rectangle number 1. When the user inputs the data the tool generates and displays the generated application description (inside the redlined rectangle number 2).

After inserting generic information about the application the user needs to add descriptions of the resources needed by the application. This process is illustrated in the Figure 9, Figure 10 and Figure 11. In the Figure 9 (inside the redlined rectangle) it is shown how the developer makes a specification of a resource. First she selects that she does not have the ID of the resource and wants to make a more generic specification. For resource type she selects the Occupancy sensor and associates it to the IoT Entity - "row 1 in the classroom 10". That is, the application needs to access an IoT resource that provides occupancy information about the row 1 in the classroom 10.

[11] The security related aspects of the mixed criticality resource management are described in the D4.4 – Security architecture for resource-constrained devices

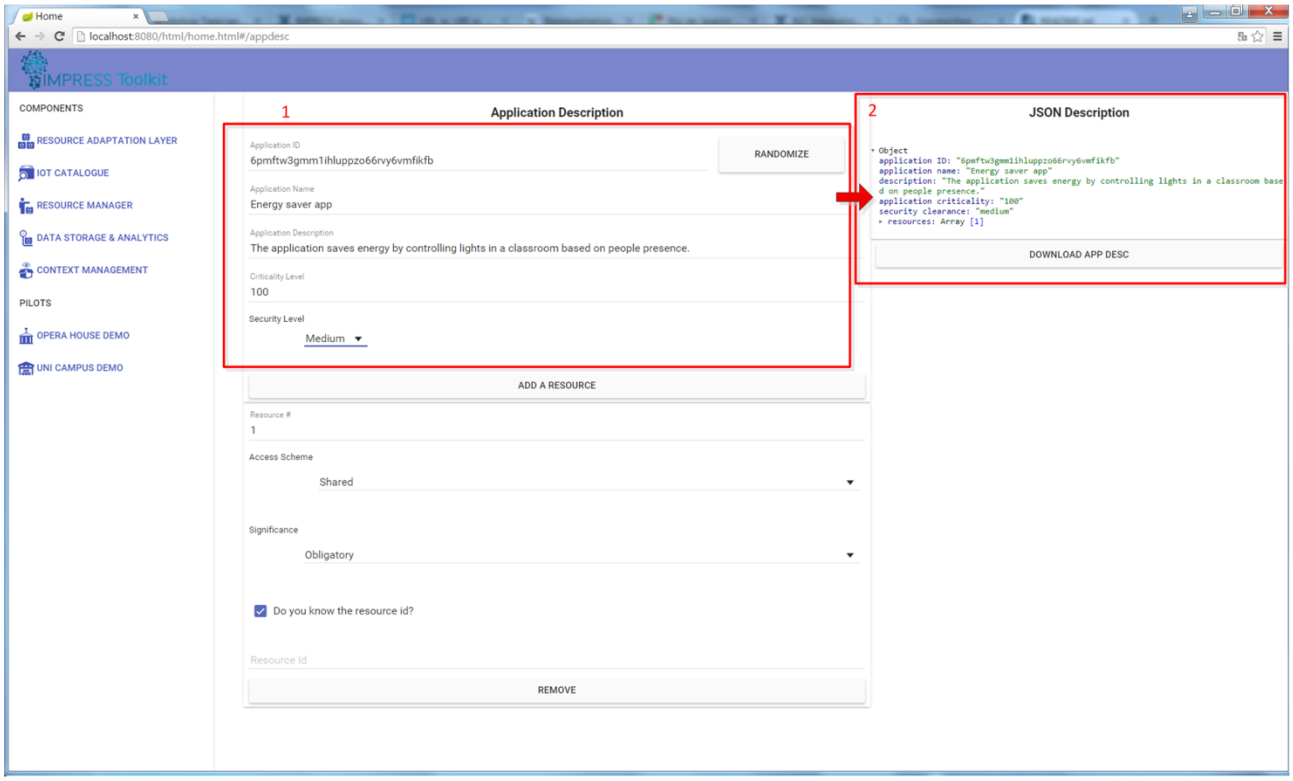


Figure 8. Application Description Generator Tool – Generic information about the application.

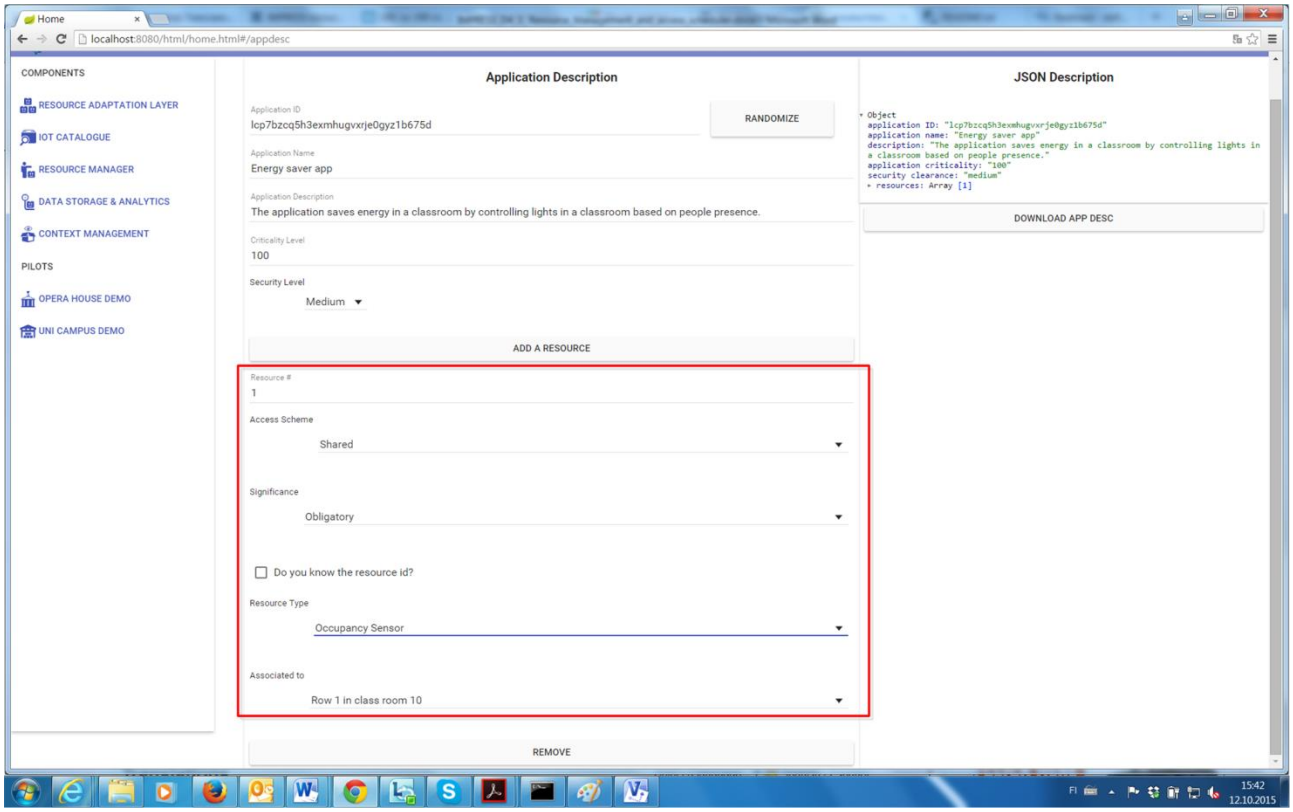


Figure 9. Application Description Generator Tool – Adding resource specifications part 1.

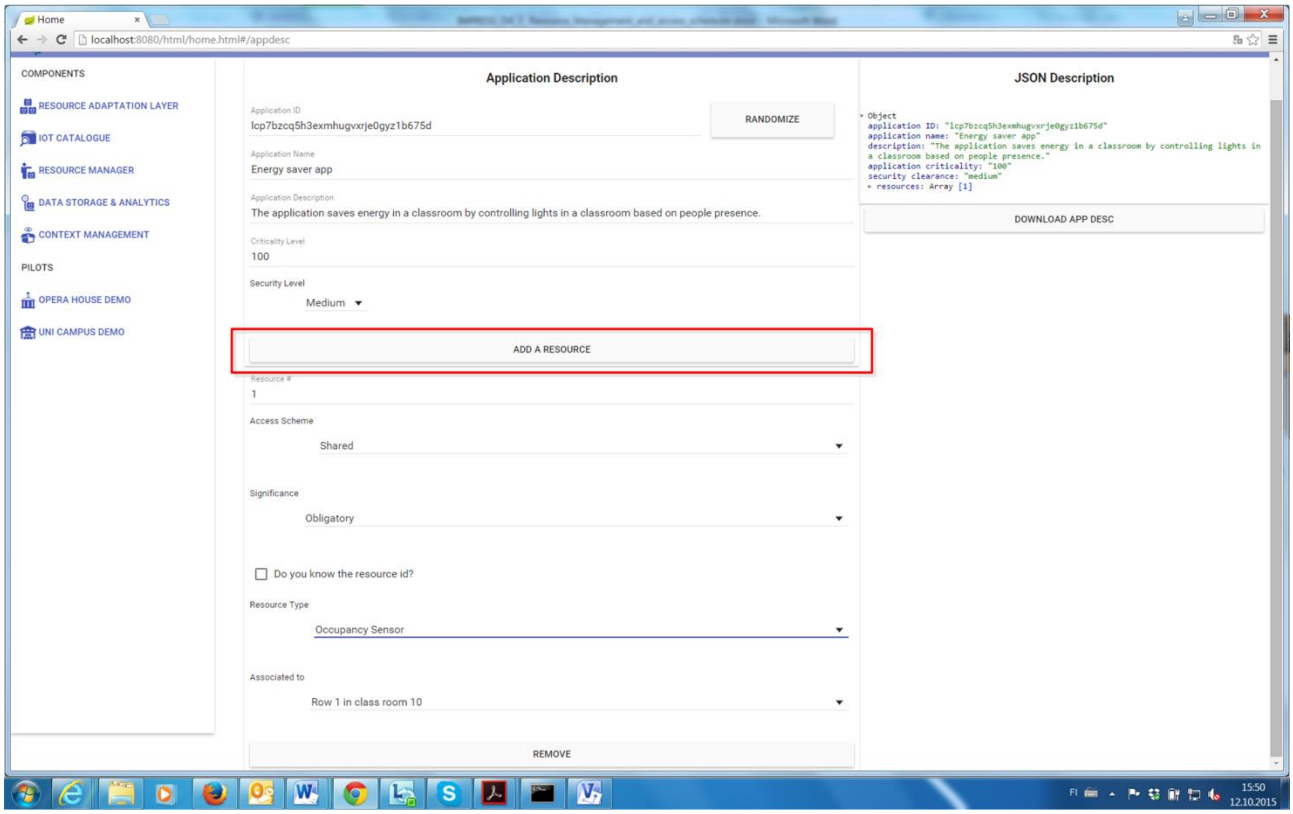

Figure 10. Application Description Generator Tool – Adding resource specifications part 2.

New resource specifications can be added by pressing the ADD A RESOURCE button as illustrated in the Figure 10. In the Energy saver app example, the developer needs to create total of eight resource specifications; one for occupancy sensor resource in each row (four rows) and one for each lighting system resource in each row. All the occupancy sensor (and lighting system) specifications are otherwise identical except to which IoT Entity the sensor (or light) is associated to. When making the resource specification the user can select the IoT Entity from a combo box as illustrated in Figure 11 in which the developer creates a specification of the Lighting System resource for row 2 in the classroom 10. The Application Description Tool fetches the available IoT Entities from the IoT Resource Catalogue using a REST interface and displays the friendly names of the entities for the user. This way the tool has up to date view of the available IoT Entities in the system.

As already mentioned, the tool generates the application description in real-time and displays it in the right side of the user interface so that the developer can also monitor this process if needed. When the application description is ready the JSON serialised format can be exported by pressing the DOWNLOAD APP DESC as illustrated in the Figure 12.

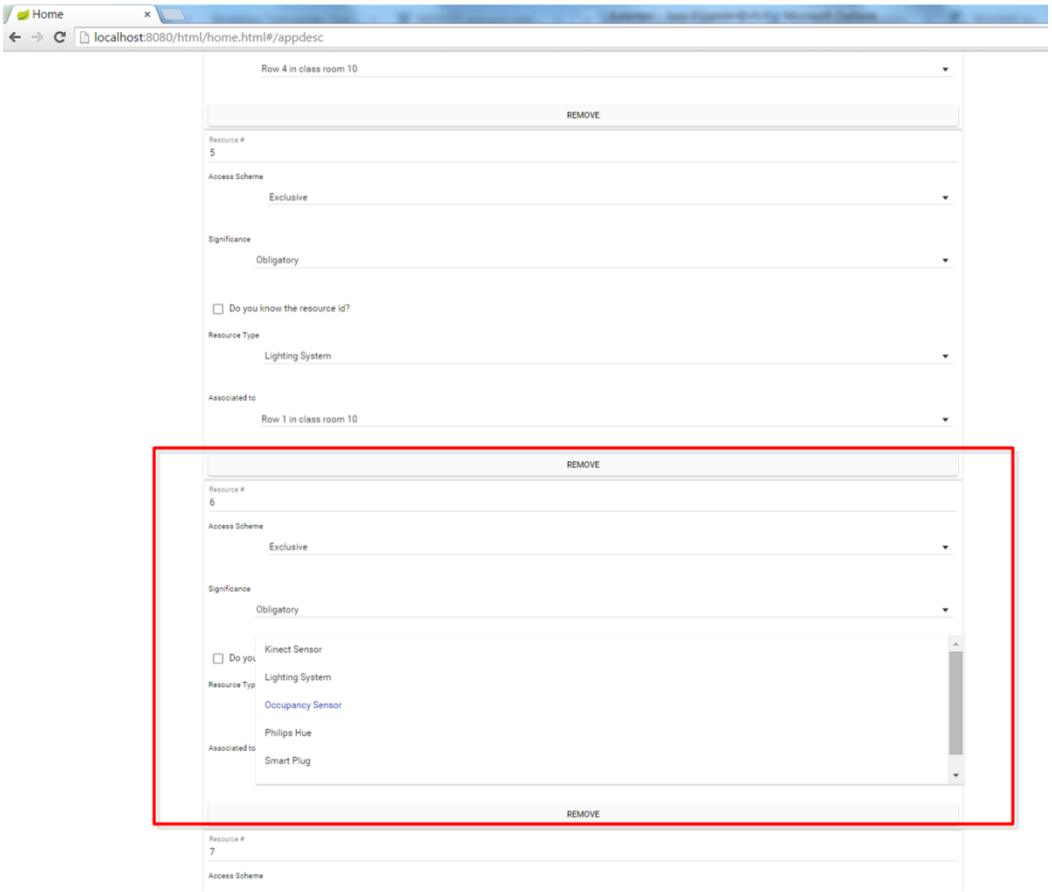


Figure 11. Application Description Generator Tool – Adding resource specifications part 3.

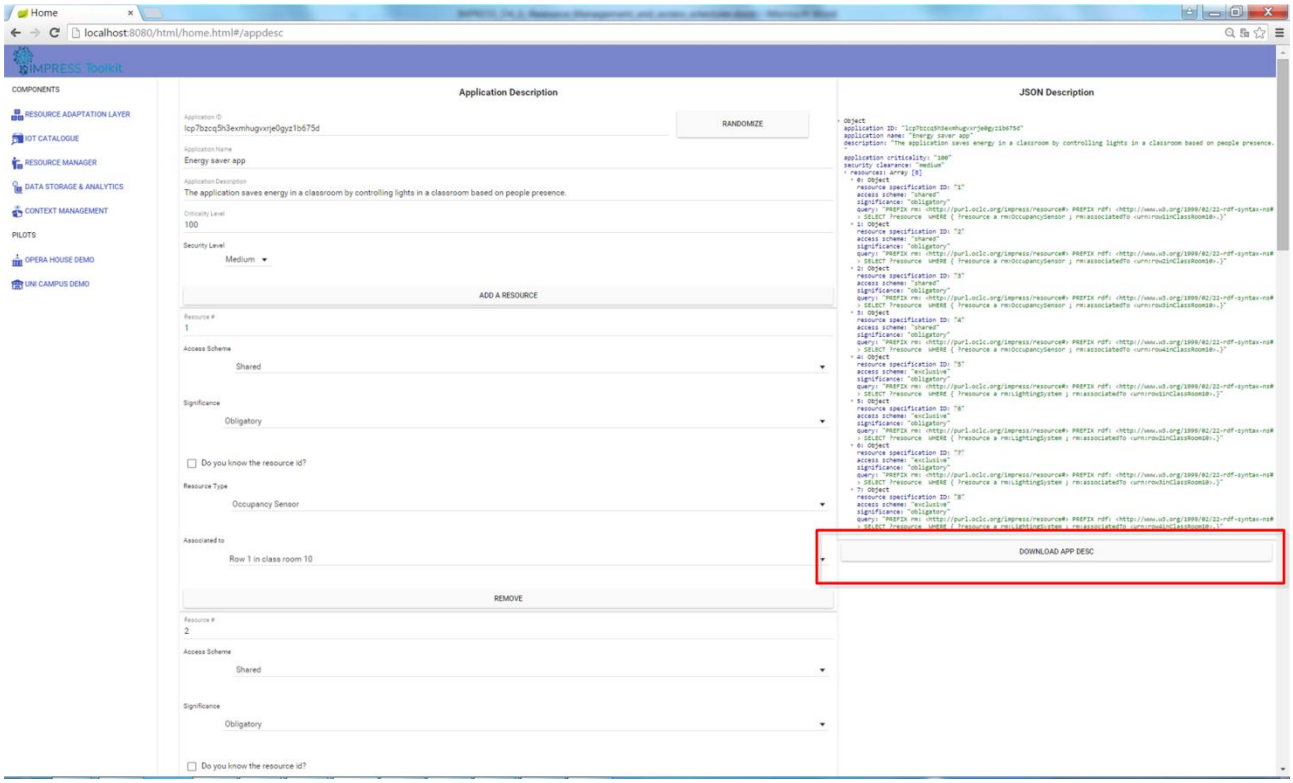


Figure 12. Application Description Generator Tool – Exporting the application description.

### 4.1.2 Developing mixed critical applications with Python

Communication with the Application-level Resource Manager can be done with any programming language that supports Websockets. However, we also provide a Python library, called grmClientLib, to ease the development of mixed criticality IoT applications.

Similarly to Global Resource Manager Protocol reference implementation the grmClientlib is based on the Twisted event-driven networking engine. Consequently, the whole mixed criticality resource management application needs to be developed around the event driven computing paradigm as will be illustrated in the example described in this section.

In the core the grmClientlib is are Handler and Factory classes. The grmClientlib provides following callback methods the user needs to implement:

```
def grmConnectionOpenedHandler(self):
"""Callback for opening the grm connection """

def grmReserveResourceResponseHandler(self, resource_spec_id, resources, status):
"""Callback for ReserveResource """

def grmReserveResourceNotificationHandler(self,  resource_spec_id, resource_id, old_resource_id):
"""Callback for ReserveResourceNotification """

def grmReleaseResourceResponseHandler(self, resourceSpecID, status):
"""Callback for ReleaseResource """

def grmRegisterApplicationResponseHandler(self, applicationID, status):

"""Callback for RegisterApplication """

def grmUnregisterApplicationResponseHandler(self, applicationID, status):
"""Callback for UnregisterApplication """
```

Factory is responsible for making protocol for each incoming connection. For the developer it provides following methods for reserving resources, releasing resources, registering applications and unregistering applications:

```
def reserveResource(self, resourceSpecID, applicationID, resource, resource_count = 1):

def releaseResource(self, resourceSpecID, applicationID):

def registerApplication(self, criticality, applicationID, resourceSpecifications):

def registerApplicationStr(self, applicationDescriptionJSON):


def unregisterApplication(self, applicationID):
```

In order to use the grmClientLib following libraries need to be first imported as follows:

```
from grmClientLib.grmClientFactoryBase import grmClientFactoryBase
from grmClientLib.grmClientProtocolBase import grmClientProtocolBase
from grmClientLib.config import *
from twisted.internet import reactor
```

The next step is to setup Twisted engine concepts[12] such as factory, protocol and reactor in the python main and then link the Handler class instance to the factory and the reactor instances.

The Factory can be created as follows:

```
    factory = grmClientFactoryBase("ws://" + grm_ip + ":" + str(grm_port), debug=False)
```

[12] https://twistedmatrix.com/trac/.

Twisted protocol handles data in an asynchronous manner. The protocol responds to events as they arrive from the network and the events arrive as calls to methods on the protocol. The protocol type for the factory can be set as follows.

```
factory.protocol = grmClientProtocolBase
```

The grmClientProtocolBase is an implementation of the Websocket/JSON protocol introduced in the section 3.4.1.

The reactor entity is core of the Twisted engine as it provides the event loop that handles all incoming events from network, keyboard, interrupts etc. Websockets operate on top of TCP and the reactor can be set to listen TCP/IP connection as follows:

```
reactor.connectTCP(grm_ip, grm_port, factory)
```

In addition to the Twisted engine concepts the developer needs to create an instance of the Handler class and set it to handle the callbacks from the Application-level Resource Manager. This can be done as follows:

```
handler = EventHandler(factory, reactor)
factory.createCallback(handler)
```

Finally, the developer needs to start the reactor event loop as follows:

```
reactor.run()
```

In the end the main loop should look like this:

```
if __name__ == '__main__':

    # Initializing the GRM client factory and protocol.
    factory = grmClientFactoryBase("ws://" + grm_ip + ":" + str(grm_port), debug=False)
    factory.protocol = grmClientProtocolBase
    reactor.connectTCP(grm_ip, grm_port, factory)

    # Create and set handler for GRM notification callbacks.
    handler = EventHandler(factory, reactor)
    factory.createCallback(handler)

    # Start the event loop.
    reactor.run()
```

Once the main loop is setup the developer needs to implement the callbacks introduced above. Next examples of the *grmConnectionOpenendHandler* and *grmRegisterApplicationResponseHandler* methods are provided. An example of the grmConnectionOpenendHandler implementation for the Energy saver application is presented below:

```
def grmConnectionOpenedHandler(self):
    """Callback for opening the grm connection """

    if self.registered is False:

        self.app_id = 'nbxb9q4zv9fhqjyk8ujvf6ti4u78j37y'

        json_dict = json.loads(app_desc)

        # Register application
        self.factory.registerApplicationStr(json_dict)
```

This callback is called whenever connection to the Global Resource Manager is established. First the code checks whether the application is already registered and registers the application if this is not the case. The actual application description is stored into the *app_desc* variable

and its content is not shown here for simplicity reasons. The application description is the same as presented in the section 3.2.2.

An example of the *grmRegisterApplicationResponseHandler* implementation for the Energy saver application is presented below.

```
def grmRegisterApplicationResponseHandler(self, applicationID, status):
    """Callback for RegisterApplication """

    if status == 0:
        self.registered = True

        # Reserve the resources
        self.factory.reserveResource(resourceSpecID=1,
                                     applicationID=self.app_id,
                                     resource_count=2)

        self.factory.reserveResource(resourceSpecID=2,
                                     applicationID=self.app_id,
                                     resource_count=2)

        self.factory.reserveResource(resourceSpecID=3,
                                     applicationID=self.app_id,
                                     resource_count=2)

        self.factory.reserveResource(resourceSpecID=4,
                                     applicationID=self.app_id,
                                     resource_count=2)

        self.factory.reserveResource(resourceSpecID=5,
                                     applicationID=self.app_id)

        self.factory.reserveResource(resourceSpecID=6,
                                     applicationID=self.app_id)

        self.factory.reserveResource(resourceSpecID=7,
                                     applicationID=self.app_id)

        self.factory.reserveResource(resourceSpecID=8,
                                     applicationID=self.app_id)
    else:
        raise ValueError('Error unable to register')
```

The *grmRegisterApplicationResponseHandler* callback is called when register application response message is received from the Application-level Resource Manager. First the code checks that the registration has been successful and then reservations to the eight resource specifications are made. Two resources of the occupancy sensor type and one lighting system resource are reserved for each row in the classroom.

## 4.2 Deployment and runtime phases

From the mixed criticality resource management point of view the deployment phase includes the deployment of new applications, IoT Resources and IoT Entities, as well as, registration of associations between IoT Resources and IoT Entities. The runtime phase in turn covers activities related to resource reservation and releasing. In practise, the deployment and runtime phases are highly intertwined (i.e., new IoT Resources can be deployed during the life cycle of an IoT application and vice versa) and therefore the description of these phases is combined into this section.

At the application deployment phase the application is registered to the mixed criticality resource management middleware by making a *register application* –request to the Global Resource Manager Protocol. The Global Resource Manager Protocol passes the request to the Application-level Resource Manager, which is a python module providing the actual resource management logic. Immediately after the application has been registered, the Application-level Resource Managers starts to search suitable resources for the application by subscribing

to the resource descriptions stored inside the SKB that match the resource specifications presented in the application description. This is an important design choice that makes it possible to tackle to performance limitations of Semantic Web technologies by executing the IoT Resource discovery/matching constantly in the background. This way the Application-level Resource Manager does not need to perform any queries or semantic matchmaking when the applications actually make reservations to the resource specifications and therefore this operation can be executed in performance efficient way.

For each resource specification in the application description, the Application-level Resource Manager creates a thread that subscribes to the resource specification represented with SPARQL. Whenever the SKB informs Application-level Resource Manager about a resource that matches a specification it is added to a list of suitable resources. That is, each thread keeps a track of suitable resources for the given resource specification. In similar manner the Application-level Resource Manager is informed whenever a resource matching a resource specification is removed from the System Knowledge Base. If the resource is new a thread for monitoring the status of the resource is also created. The application registration and resource discovery process is illustrated in the Figure 13.

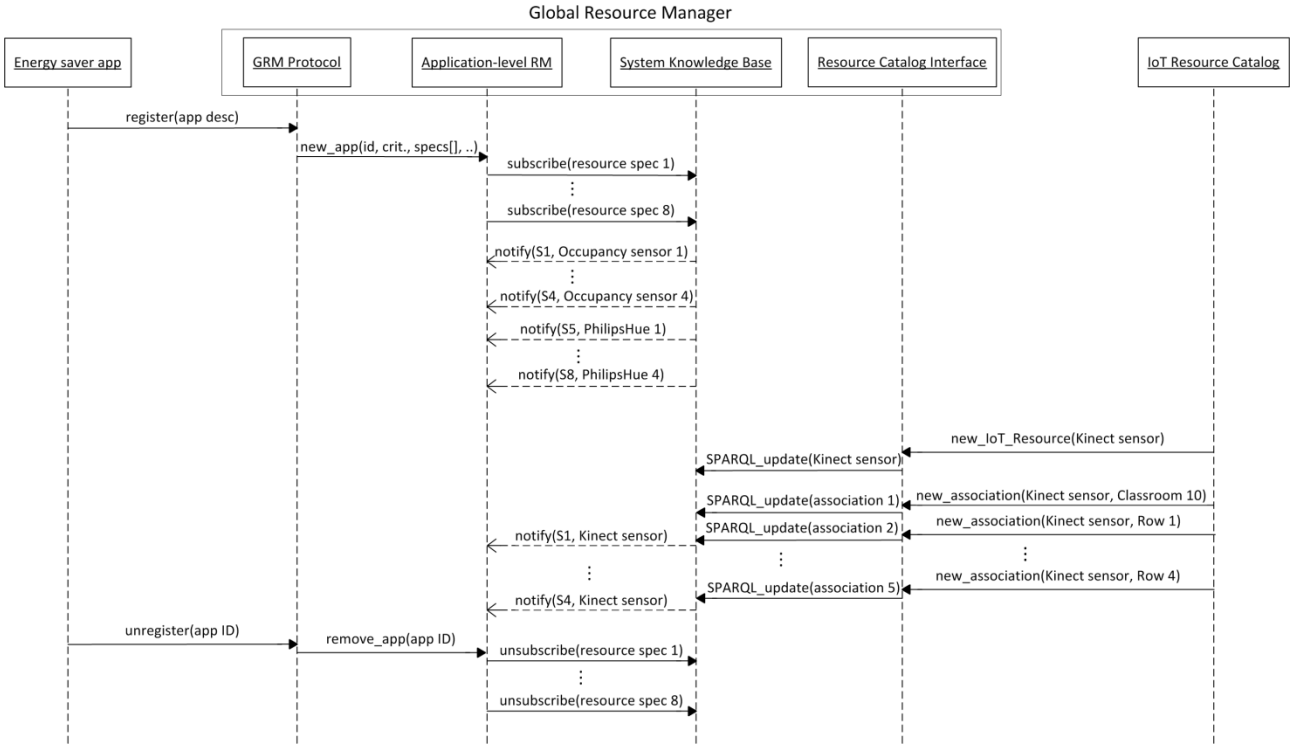


Figure 13. Example of application and IoT Resource deployment.

At the beginning of the scenario there are simple occupancy sensors and PhilipsHue lights associated for each row in the classroom. When the Energy saver application is deployed into the system it registers itself to the Global Resource Manager through the GRM Protocol interface. The application is added to the Application-level Resource Manager that subscribes to resource specifications. Individual thread is assigned for each specification as mentioned above. The Application-level Resource Manager is notified about the suitable resources for each specification.

Later in the scenario Kinect Xbox360 sensor is deployed into the classroom and associated with the room and all its rows. Among other features this device can also provide occupancy data and it is therefore also classified as occupancy sensor. Therefore it matches the specifications 1-4 of the Energy saver app and the Application-level Resource Manager will be notified about

these IoT Resources. It should be noted that the Energy saver application is not anyway notified about these resources at this phase. The Application-level Resource Manager just stores this information so that it can immediately respond when the application actually needs to use these resources.

Whenever an application needs to access a resource, it sends a *reserve resource* –message to the Application-level Resource Manager and specifies how many resources matching the specification it wants to reserve. It is important to notice that the *reserve resource* requests are persistent and the application needs to release the resources if they are not needed anymore. After receiving the *reserve resource* –message the Application-level Resource Manager first searches suitable resources from free resources pool; if free resources are not available, suitable resources used by the least critical applications are selected (assuming of course that the application making the new reservation is more critical). If suitable resources are found the Application-level Resource Manager will return URIs for the application in the *reserve resource* response message. The Application-level Resource Manager also notifies the corresponding Local Resource Manager(s) that the application is authorised to access the resource. Additionally, if the selected resources are used by other application(s) in exclusive access mode these applications and the corresponding LRMs are notified and new resources (if possible) are assigned to the application(s). It should be again noted that since the Application-level Resource Manager is all the time aware about the most suitable resources for each application it is able to quickly respond to the reservations made by the applications (*i.e.*, no match making is needed at this phase). In the Figure 14 an example runtime mixed criticality resource management scenario is depicted.

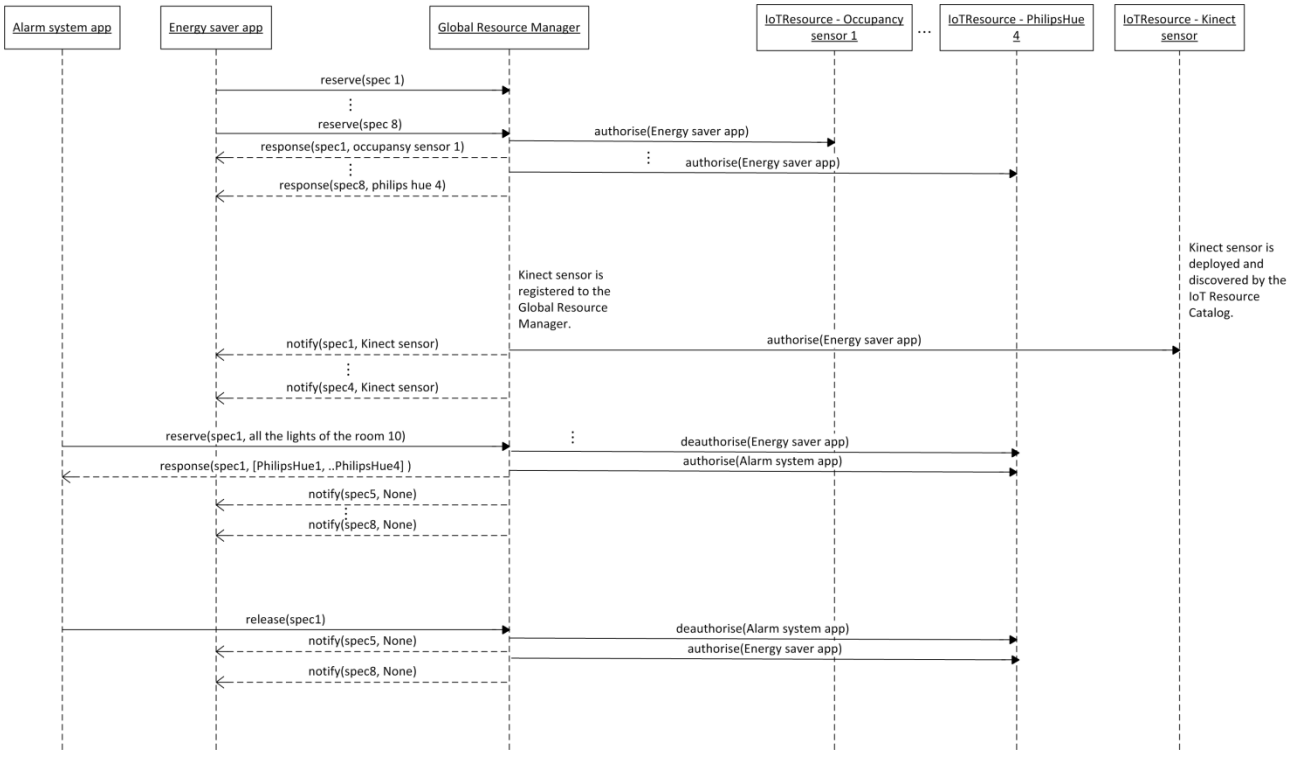


Figure 14. An example resource allocation scenario with alarm system and energy saver apps.

At the beginning of the scenario the system set-up is the following. There are two applications deployed into the system: Energy saver app and the Alarm system app. There are also simple occupancy sensors and Philips Hue lights associated for each row in the classroom. For the sake of clarity only one Occupancy sensor and Philips Hue light resource is illustrated in the sequence chart.

The Energy saver app makes a reservation to all the eight resource specifications and receives a response from the Application-level Resource Manager. Global Resource Manager also notifies corresponding LRM(s) that the application is authorised to access the given resource. Later in the scenario a Kinect sensor is deployed into the IMPReSS platform and the Application-level Resource Manager automatically reserves the resource for the Energy saver application and notifies the application and the LRM about the reservation.

Later in the scenario the Alarm system application receives a notification about smoke detector sensors and makes reservations to all the Light System resources in the classroom in order to notify people about the danger. It uses exclusive access scheme and is more critical application than the Energy saver app and therefore the resources are assigned for it and the corresponding Local Resource Managers are notified about the situation. In the end of the scenario the alarm ends and the Alarm system application releases the Lighting system resources. Because the reservations are persistent and there are now available resources for the Energy saver application it and the LRMs are notified about the situation.

# 5. Device-level resource management

When compared to the application-level resource management the device-level mixed criticality resource management is relatively simple process. However, it is a concrete problem in existing buildings (especially in countries where power outages are common) and for this reason this features was also incorporated into the IMPReSS mixed criticality resource management middleware.

The basic idea in the device-level mixed criticality management is to control the power of devices based on device criticality so that enough energy is available for the most critical applications for the duration of a power outage. The approach is based on the assumption that all the devices[13] are powered by the same emergency supply (e.g. generator or battery) in the case of a power outage.

The Global Resource Manager module responsible for providing the device-level resource management logic is called Device-level Resource Manager. Similarly to the rest of the Global Resource Manager components it is implemented with the Python programming language. In addition to the Application-level Resource Manager the device-level resource management approach includes the Local Resource Managers of all IoT Resources, as well as, two special types of IoT Resources that need to be tailored depending on the type of the IoT system. The first IoT Resource type monitors the electricity network and notifies the Application-level Resource Manager about the power outages. The second IoT Resource type monitors the energy level in the emergency supply and notifies the Application-level Resource Manager whenever the level changes.

## 5.1 Initialisation and deployment phase

The Device-level Resource Manager needs to be configured for each individual IoT system so that it meets the requirements of the given environment. During configuration the system administrator defines the order and situation in which devices are turned off during power outages and the schedule in which devices are turned back on after the power outage is over. In practise the power outage mode is configured by defining energy level (specified in terms of how many percent of energy is available in the backup supply) and device criticality level thresholds. When certain energy level is reached all devices below the criticality threshold are turned off. When power outage ends it is important that all devices are not turned on at the same time in order to avoid a situation where the grid becomes unstable. For this purpose the system administrator can define how many seconds are waited before devices below specific criticality level are turned back on. The initial configuration is done at the installation phase but it is also possible to change the settings during runtime. In practise the configuration is done with the Mixed Criticality System Management Tool.

Whenever new devices are deployed into the IMPReSS system they are discovered by the IoT Resource Catalog component that notifies the Global Resource Manager about these IoT Resources as already illustrated in the section 4.2. The Device-level Resource Manager is subscribed to the all IoT Resource descriptions stored into the System Knowledge Base. This way it is aware whenever new IoT Resources are deployed into or removed from the IMPReSS platform. The interaction between components during the deployment phase is illustrated in the Figure 15.

[13] In principle it is also possible to support several different emergency supplies each powering a separate pool of devices using the approach presented in this deliverable. However, this feature is not yet implemented to the mixed criticality middleware.

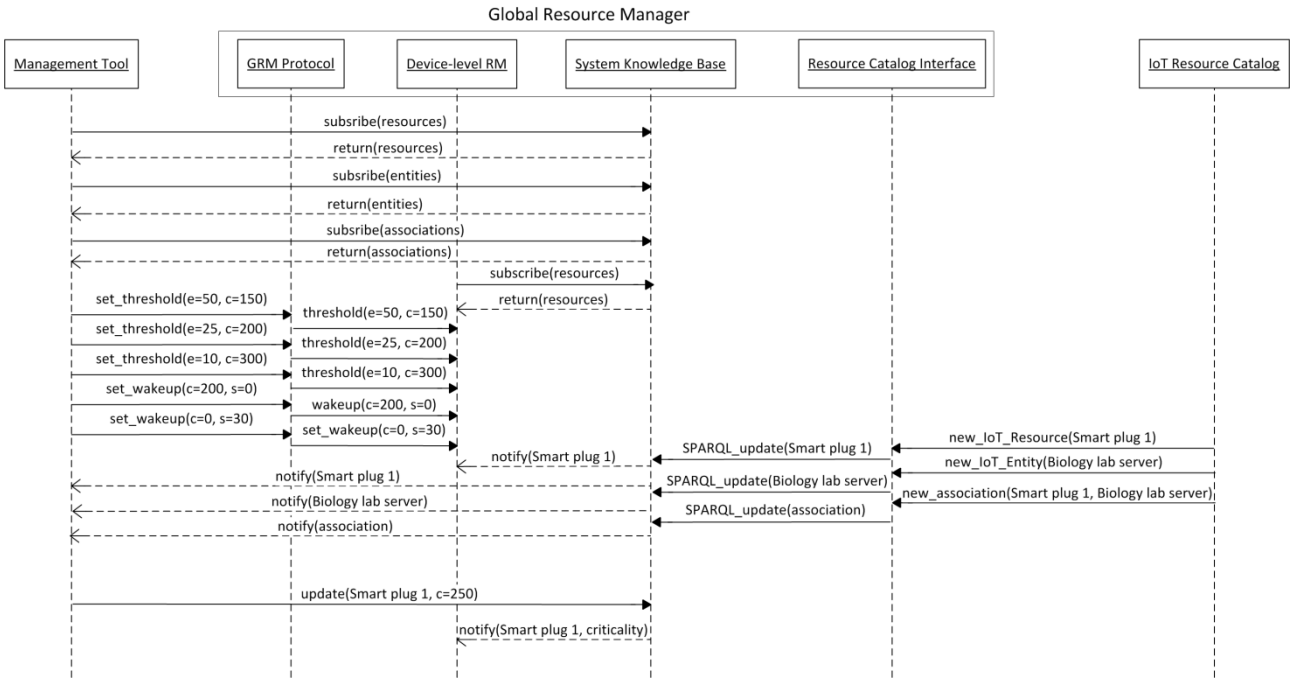


Figure 15.Example of mixed criticality component interaction during initialisation and deployment.

In this scenario the system administrator sets following three energy thresholds:

1. When 50% of the energy is available all the devices below criticality value 150 will be turned off.
2. When 25% of the energy is available all the devices below criticality value 200 will be turned off.
3. When 10% of the energy is available all the devices below criticality value 300 will be turned off.

The administrator also defines that once power outage is over the wakeup is executed in two phases. Immediately after the power outage is over all devices above criticality value 200 will be turned on. After 30 seconds all devices above critical value 0 (i.e., all the rest of the devices) will be turned on.

New devices registered to the Global Resource Manager are assigned with criticality value 100 by default and the system administrator needs to assign proper value for the devices during the deployment. This is also done with the Mixed Criticality System Management Tool. As illustrated in the Figure 15 the tool is subscribed to all IoT Resources in the system through the SKB Websocket interface and can thus display the system status for the user. When user sets a new criticality value for a device it is updated to the SKB and the Device-level Resource Manager is notified about the new value. In the Figure 16 it is demonstrated how new criticality value for IoT Resources can be set with the tool in practise.

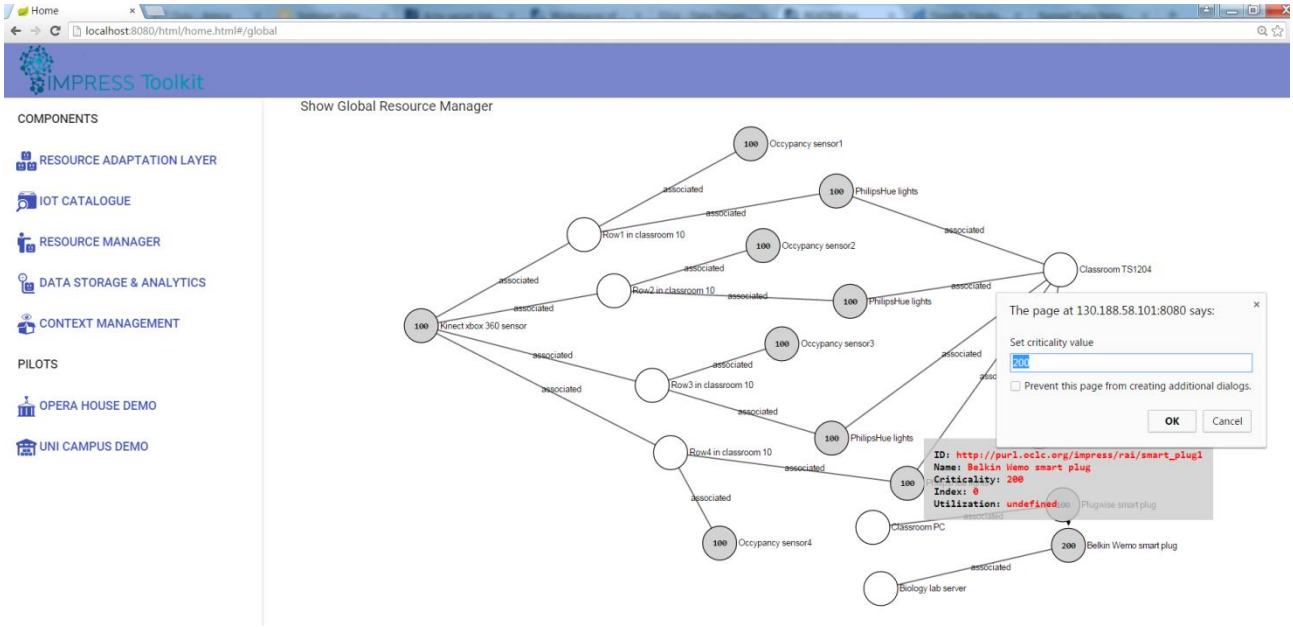


Figure 16. Setting criticality value for Belkin Wemo smart plug associated to Biology lab server.

## 5.2 Device management during power outages

When power outage occurs the following interaction happens in the mixed criticality resource management middleware. First, the IoT Resource dedicated for detecting power outages notifies the Global Resource Manager. After receiving the power outage event the Global Resource Manager starts to listen energy level notifications from the Energy level sensor. Whenever the energy level drops below a threshold limit (defined during Device-level Resource Manager initialization) all the devices whose criticality is lower than the threshold defined for the that energy level will be turned off. In practise this is done by sending the turn off request to the corresponding Local Resource Manager. When the power outage is over the Power outage sensor will notify the GRM, which starts to turn on devices using the schedule defined by the system administrator. Example interaction of the mixed criticality resource management components during power outage is illustrated in the Figure 17.

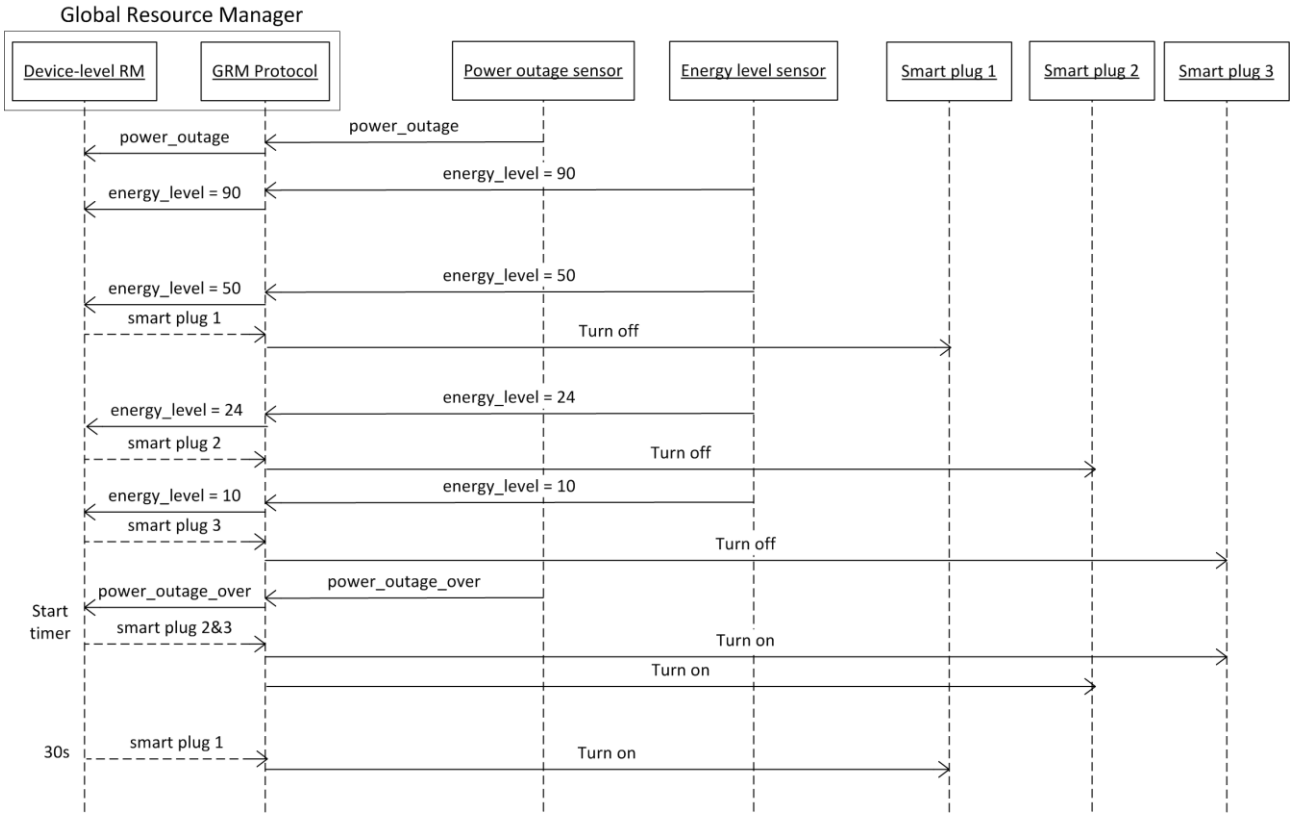


Figure 17. Example interaction of components during device-level mixed criticality management.

For simplicity sake only three smart plugs are used in the example scenario. The criticality values for smart plug 1, 2 and 3 are 50, 200 and 270 respectively.

The scenario begins when the power outage sensor detects a power outage and notifies the Global Resource Manager about it. The Global Resource Manager then starts to receive energy level notifications from the Energy level sensor. Whenever new notification is received the Device-level Resource Managers check if new devices need to be turned off. The list of devices and their criticalities are obtained from the SKB as presented in the Figure 15. This scenario uses the same configuration that is done in the Figure 15. So when the energy level drops to 50 percent all devices below criticality value 150 are turned off. When the energy level drops to 25 percent all devices below criticality value 200 are turned off. Finally when the energy level drops to 10 percent all devices below criticality value 300 are turned off.

When the power outage is over the Power outage sensor notifies the GRM about the event and the GRM starts to follow the wakeup plan defined by the system administrator. That is, all devices that have 200 or higher criticality are turned on immediately. Then the Device-level Resource Manager waits for 30 seconds and turns on rest of the devices.

# 6. Conclusion

In this deliverable the approach and architecture for mixed criticality resource management was described. The reference implementation of the mixed criticality middleware is available at the IMPReSS Git and can be accessed as specified in the Appendix I.

The IMPReSS mixed criticality middleware proposed in this deliverable consists of two levels: system and local. At the system level the Global Resource Manager discovers suitable resources for applications and selects which application can access which resource. The GRM also decided which devices are turned off in different phases of the power outage. At the device-level there is one Local Resource Manager for each device. The role of the LRM is to interact with the Global Resource Manager to make sure that only authorized applications access the resource. It also schedules the requests send by the applications if shared access scheme is used. Additionally, the Local Resource Manager is responsible for turning off/on devices whenever requested by the Global Resource Manager.

With the mixed criticality resource management approach described in this deliverable it is possible solve application and device resource management issues. The idea behind the application-level resource management is to manage how IoT applications can access sensor and actuator resources provided by the IMPReSS platform. That is, with the mixed criticality middleware it is possible for 3rd party application to share IoT resources without compromising the behavior of the most critical IoT applications. At the device-level the mixed criticality resource manager makes it possible to configure which devices are provided with power in the case of a power outage so that there would be enough energy for the most critical devices to stay functional for the duration of the power outage.

# 7. References


(Bassi et al. 2013) Bassi A, Bauer M, Fiedler M, Kramp T, van Kranenburg R, Lange S & Meissner S. (2013) Enabling Things to Talk - Designing IoT Solutions with the IoT Architectural Reference Model. Heidelberg, New York, Dordrecht, London: Springer

(Bauer et al. 2013) Bauer M, Boussard M, Bui N, Carrez F, Jardak C, De Loof J, Magerkurth C, Meissner S, Nettsträter A, Olivereau A, Thoma M, Walewski J, Stefa J & Salinas A. (2013) Internet of Things – Architecture (IoT-A): Deliverable D1.5 – Final architectural reference model for the IoT v3.0. D1.5: 1-482.

(Berners-Lee et al. 2001) Berners-Lee T, Hendler J & Lassila O. (2001) The Semantic Web. Scientific American : 29-37.

(Burns and Davis 2015) Alan Burns and Robert I. Davis (2015) Mixed Criticality Systems - A Review. Sixth edition,1/8/2015. [Online]. URL: https://www.cs.york.ac.uk/media/computerscience/documents/public/researchprojects/review2015b.pdf

(Compton et al. 2012) Compton M, Barnaghi P, Bermudez L, García-Castro R, Corcho O, Cox S, Graybeal J, Hauswirth M, Henson C, Herzog A, Huang V, Janowicz K, Kelsey WD, Le Phuoc D, Lefort L, Leggieri M, Neuhaus H, Nikolov A, Page K, Passant A, Sheth A & Taylor K. (2012) The SSN ontology of the W3C semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web 17(0): 25-32.

(Fette& Melnikov 2011) I. Fette, A. & Melnikov (2011), The WebSocket protocol, HyBi Working Group. [Online] URL: http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17.

(Gangemi 2007) Gangemi A. (2007) DOLCE UltraLite OWL Ontology. URI: http://www.loa.istc.cnr.it/ontologies/DUL.owl. 2014(10/11).

(Hill and Lake 2010) Hill MG & Lake TW. (2000) Non-Interference Analysis for Mixed Criticality Code in Avionics Systems. Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on. : 257-260.

(Huber et al 2008) Huber B, El Salloum C & Obermaisser R. (2008) A Resource Management Framework for Mixed-Criticality Embedded Systems. Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE. : 2425-2431.

(Vestal 2007) Vestal S. (2007) Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance. Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International. : 239-243.

(W3C 2014) W3C RDF Working Group. (2014a) RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation. URI http://www.w3.org/TR/rdf11-concepts/.

(W3C 2013a) W3C SPARQL Working Group. (2013) SPARQL 1.1 Query Language, W3C Recommendation. URI: http://www.w3.org/TR/sparql11-query/.

(W3C 2013b) W3C SPARQL Working Group. (2013) SPARQL 1.1 Update, W3C Recommendation. URI: http://www.w3.org/TR/sparql11-update/.

(W3C 2013c) W3C SPARQL Working Group. (2013) SPARQL 1.1 Query Results JSON Format, W3C Recommendation. URI: http://www.w3.org/TR/sparql11-results-json/.

(Weiser 1999) Weiser M. (1999) The computer for the 21st century. SIGMOBILE Mob.Comput.Commun.Rev. 3(3): 3-11.

# Appendix I - Installation and example apps

**Pre-requisites**

- Linux OS (tested on Ubuntu derivatives)
- Python 2.7, python-dev and python-setuptools packages
- Red-SIB is needed in Impress resource management architecture. It can be found from Sourceforge Red-SIB.
    - After downloading the package, extract it and run *install.sh*

**Installation**

- Download the whole Impress repository from Fraunhofer git-server : *git clone https://[username]@scm.fit.fraunhofer.de:8181/scm/git/impress*
- Install the grmClientLib: sudo python /impress/grmServer/grmClientLib/setup.py install

**Running GRM server**

- GRM uses SIB as database, so it needs to be started first
- Start SIB:
    - start redsibd: redsibd&
    - start sib-tcp: sib-tcp&
- Run grmServer: python /grmServer/grmServer_dist/grmServer/grmServer.py
    - Change GRMSERVERADDRESS, GRMSERVERPORT, GRMSERVERIP, SKB_ADRESS, SKB_PORT according your configuration in /grmServer/grmServer_dist/grmServer/config.py
- Run resource catalog interface (responsible for registering the resources in the SIB):
    - python /grmServer/grmServer_dist/grmServer/resource_catalog_interface.py

**Example applications:**

Energy saver application (in /impress/grmServer/examples )can be used as starting point to develop new applications. There are three versions of the application available. The versions 1 and 2 are simple examples that use only two resource specifications. The version 1 creates the application description using the grmClientLib API. In the version 2 the application description is represented in string format. The idea here is that the output provided by the application description generator tool can be directly utilized.

In addition to the application the example folder includes scripts that register IoT resources in the global resource manager. Resources matching the specifications of the energy saver app 1 and 2 can be inserted by running the /impress/grmServer/examples/occupancy_and_light_resource_registerer.py script. The resources matching the specifications of the version 3 can be added by running the /impress/grmServer/examples/python resource_registerer.py (i.e. this script insets the IoT resources, VEs and associations to be used in the final review).

To run the application:

- python /impress/grmServer/examples/energy_saver_app.py OR python /impress/grmServer/examples/energy_saver_app_v2.py

To run the resource register script:

- python /impress/grmServer/examples/occupancy_and_light_resource_registerer.py

The version 3 of the energy saver app simulates the energy saver app to be implemented with context manager module. It contains 8 resource specifications.

To run the application:

- python /impress/grmServer/examples/energy_saver_app_v3.py

To run the resource register script:

- python /impress/grmServer/examples/python resource_registerer.py