(FP7 614100)

D4.1.2 Final application classification language and tool

05 January 2015 – 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: D4 1 2 Final application classification language and tool.docx
Document version: 1.0
Document owner: Ferry Pramudianto (Fraunhofer FIT)

Work package: WP4. Mixed Criticality Resource Management
Task: Task 4.1 Application classification language and tool
Deliverable type: P

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

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Ferry Pramudianto	1/11/2014	ToC and Mixed-Critical Application Scenario.
0.2	Jussi Kiljander	30/11/2014	Application Descriptions.
0.3	Nishananth Baskaran	20/12/2014	Implementation of the application description generator added.
1.0	Ferry Pramudianto	5/1/2015	Improved based on the review comments.

Internal review history:

Reviewed by	Date	Summary of comments
Enrico Ferrera (ISMB)	23/12/2014	Accepted with minor comments

Index:

Contents

1. Introduction

1.1 Background

In IMPReSS, we assume that sensors and actuators, which are referred as IoT resources in the remainder of this deliverable, could be developed separately by different developers or organizations than the software that access them. Moreover, in IoT, sensor and actuator resources are often accessed concurrently by different software. Without a proper resource management, the performance of the whole system maybe degrading since the sensor and actuator resources have a limited capacity to serve number of applications at the same time. This effect may not be acceptable for some highly critical applications that must be served near real-time. The mixed critical resource management in IMPReSS aims at providing a generic solution for deploying mixed critical applications so that they run as desired. The mixed criticality framework in IMPReSS should be able to balance the load to the IoT resources and serve the more critical applications first before the others.

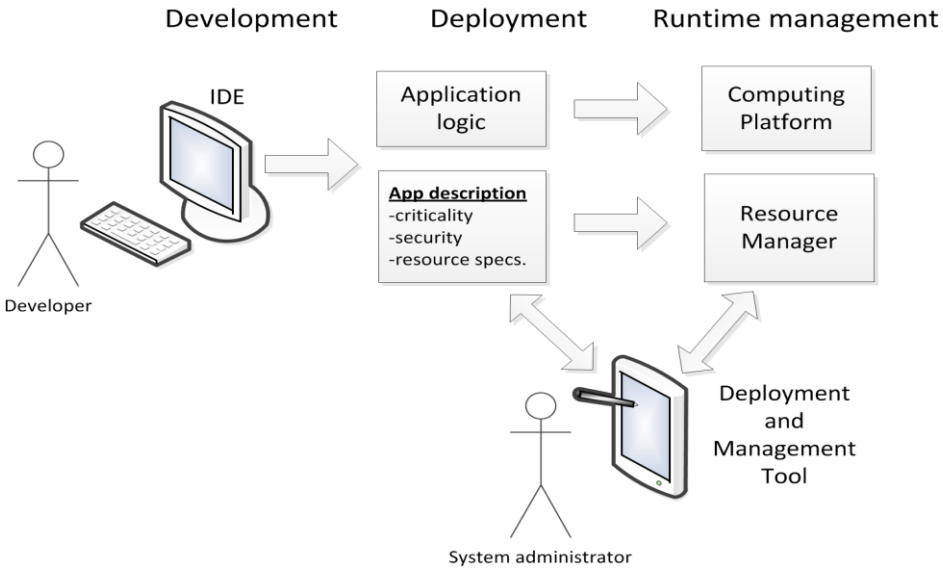

Figure 1. Development, deployment, and runtime management of mixed criticality applications.

As described in the D4.1.1, the mixed-critical application lifecycle within the IMPReSS platform consists of three phases including the development, deployment, and runtime. IMPReSS provides tools that help the development, deployment, and configuration of mixed criticality IoT applications.

At runtime, the resource management consists of two components, the global resource manager (GRM), and local resource manager (LRM). GRM is intended to solve conflicts between applications that either request exclusive access to a same resource or request access to different resources that might interfere with each other in the real word (e.g. lights, heating systems, etc.). Second, the GRM is also useful to keep the load between similar resources in balance in order to prevent bottlenecks and keep the performance of the whole IoT system is as optimal as possible.

The LRM functions as the final line of defense for the resources that are represented by software proxies. LRM guarantees that applications with higher criticality level are served first before the less critical applications.

To implement this framework, the applications and the available resources must be decoupled and dynamically coupled according to the decision made by the GRM and LRM. Secondly, the resources and the applications must be annotated with meta-information that allows the GRM and LRM to evaluate the critical levels of the applications as well as the appropriate resources that can be assigned to the applications.

1.2 Purpose, context and scope of this deliverable

This deliverable focuses on the development part of the resource management framework. In the D4.1.1, we presented the possible meta-information that can be used by the GRM to assign the resources to an application. In this deliverable, we present the further development of the application description concept and its implementation.

2. Mixed-Critical Application Scenario

Within the requirement elicitation process, we tried to identify use cases, in which mixed-critical IoT applications will likely be deployed. In this regards, we had to reimagine how building management systems work. In the requirement workshop, we envisioned that building management systems will be an open IoT systems, which decouple resources from the applications that access them. The system could be extended by adding apps that access resources, similar to what we could see on the computing and smartphone platforms. The main advantage of this approach is, the functionality of the system could be extended by third party developers. However, opening up the control to devices installed in the building requires a proper management in order to guarantee that the whole system could function well. The IMPReSS resource manager tries to provide a general solution to coordinate the access to the resources in the building to keep the system working in harmony. This includes prioritizing access to the IoT resources (sensors, actuators, and network) depending on the criticality of the applications. In addition, it is also useful to balance the load of the IoT resources.

2.1 Opera House Scenario

The opera house scenario consists of a building management system with different applications accessing the sensors and actuators for the lighting, air conditioner, ventilator, and displays including public displays as well as mobile devices such as tablet.

We identified the following use cases related to mixed-criticality:

1. Several applications are allowed to control the lighting and air conditioner in the dressing rooms. Currently, we foresee three applications. First, an application is used by the facility manager to schedule the usage of lighting and air conditioner (AC) to guarantee that they are switched off outside the operational hours of the building. Second, a mobile application is used by the employee of the opera house. The mobile app may be given a higher priority for controlling devices at his or her own workspace, e.g., they could override the schedule for the lighting, and AC based on their personal habits.

   Moreover, further applications to control the ventilation, and air conditioner should be deployed for increasing the air quality in the building, reducing the energy consumptions, preserving the historical painting, as well as user comfort level. These could be different applications, which could have contradicting policies for controlling the devices. For instance, in the normal condition, to save energy, the cooled air inside the building should be kept by closing the ventilation fans. However, another application could override this policy when the CO2 level in the building is too high.

2. The second use case is ambient notification. Ambient notification is implanted with Philips hue bulbs, which can change its color and intensity. The combination of the two could be used to notify the artist in the dressing rooms about important events at the main stage so that they could manager their timing better. For instance, if they have to go to the main stage in five minutes, the light intensity is increased and the color becomes warmer.

   In addition, the lights could also be used by other applications to notify the users about other events, e.g., if there are hazards such as a fire, too much CO2 in the building, or humidity level could damage the painting. Since several applications are changing the color and intensity of the lights, each applications must have different priority. For instance, the notification for hazardous condition should have a higher priority than the other apps. The fire App may use the Philips hue bulbs in the corridors to lead the users to the nearest exit.

3. The third use case is, environmental condition monitoring using wireless thermometer, humidity, CO2, and presence detectors. To save the battery of the wireless sensors, the public display application and the tablet application may only be allowed to poll the data every 15 minutes. However, in case of a fire, the sensors could be repurposed by another application from the fire department to create a heat map and detect if there are people in the rooms that need to be

saved by the firefighters. This application must have an exclusive right to poll the sensors in a very rapid interval of time to monitor the spread of the fire in real time.

2.2 University Campus Scenario

The university campus scenario consists of a building management system with a broader scope than the opera house scenario. UFPE operates several data centers and computing labs that have PCs turned on the whole day. Applying an energy management system in the campus will allow a great reduction of the energy consumptions as well as managing the devices centrally.

In addition to the similar scenario to the opera house that can be applied at UFPE, We identified the following use cases related to mixed-criticality:

1. UFPE campus now operates several uninterruptible power supply (UPS) for the server. Although, the capacity of the battery is increased regularly, the increase is not proportional with the increase of the server racks. Therefore, when there is an outage, the UPS cannot power the whole data center as planned anymore. As a solution, the servers must be prioritized and shut down when necessary in order to guarantee the more critical servers get fewer interruptions. Therefore, each server should host an agent software that request access to the battery through the resource manager. When the battery level become more critical, the access to the battery should be cancelled and the server should shut itself down.
2. Another use case is repurposing the lighting, displays (projectors, public screens), and speakers in the classrooms for different kind of notifications by different apps, similar to the entertainment system in the passenger flights. An application could use these devises to alert the users towards hazards situation such as fire, natural disasters. Using different modality will increase the awareness of the users in the building.

3. Application Descriptions

There are two representation formats for applications in the mixed-criticality resource management approach for Internet of Things. In development and deployment phase, JSON-serialised representation (presented in the section 3) for applications is used. This description is generated by the Application Description Generator described in more detail in the chapter 4. At runtime, applications representations are stored inside the System Knowledge Base as part of the overall resource management ontology. This ontology (presented in the section 3.1) is used for monitoring the system state during runtime resource management activities. A Turtle syntax representation for the ontology is presented in the Appendix A.

3.1 Development and deployment phase representation

The application descriptions consist of following parameters: application ID, human-readable description, criticality level, security clearance level and a list of resource specifications. Each resource specification in turn consists of resource specification ID, access scheme, significance, reliability and query parts. The type and content for these parameters are described in more detail in the following listing.

- **Application ID** : [string] Unique identifier for each application instance.

  Application developer defines a unique ID for the application, which is combined with the instance ID, generated by the Global Resource Manager in the deployment phase to form the actual application ID.

- **Description** : [string] Human-readable description of the application.

  Description of the application is created by the application developer in order to help system integrator and recipient understanding the purpose of the application.

- **Application criticality** : [number] Positive integer value that defines how important the application is the IoT system. Criticality of the application is initially defined by the application developer depending on the order how the access to the resources should be prioritized. During deployment phase, it can be modified by the system integrator if necessary. It is also possible to update the application criticality at runtime if needed.

- **Security clearance**: [string] Specifies how confidential the data should be delivered by the IoT resources to the applications, which could be used by the developers to decide on the data encryption level between the IoT resources and the applications. Possible values include: *Untrusted*, *Low*, *Medium*, *High*. Security clearance is defined in the deployment phase by the system integrator, and can be updated at runtime if necessary.

- **Resources** : [array] List of specifications for resources required by the app. The parameters for single resource specification are presented below. All the parameters are defined in the application development phase by the application developer. In deployment phase the query parameter is updated with property that associates the application to resources in specific environment.

  - **Resource specification ID** : [string] Unique identifier (inside one application description) for resource specification.

  - **Access scheme** : [string] Defines the access policy for the resource. Possible values are *Shared* or *Exclusive*.

  - **Significance** : [string] Defines the importance of the resource for the given application. Possible values are *Obligatory* and *Useful*.

  - **Reliability** : [string] Specifies the reliability requirement for the resource (the more critical actions are based on the resource the more reliable the resource needs to be). Possible values include: *Low*, *Medium*, and *High*.

  - **Query** : [string] SPARQL SELECT query that specifies the resource of interest. This query is used to find suitable resources for the app from the System Knowledge Base. Name "resource" should be used for the variable to be bound with resource URI. The resource management ontology represented in the section 3.2 provides the basic vocabulary for

defining the query part. The idea is that domain specific ontologies used to describe resources in higher detail can be used when necessary.

An example of the application description in presented in the Figure 2. The application depicted in the figure controls lights in a dressing room based on the room occupancy (or the time of the day). One instance of this application is deployed into each dressing room in the Teatro Amazonas Opera House. The application has two resource specifications: one for the lighting system and one for the occupancy sensor. The lighting system is a mandatory resource with exclusive access scheme. Since the application is able to control lighting based on the time of the day the significance of the occupancy sensor resource is only "Useful". The access to the occupancy sensor resource is shared.

```
{
  "application ID": "35fac920-2f4e-11e4-8c21-0800200c9a66",
  "description" : "Controls lights based on room occupancy. "
  "application criticality": 200,
  "security clearance": "Medium",
  "resources": [
    {
      "resource specification ID": "1",
      "access scheme": "Exclusive",
      "significance": "Oblicatory",
      "reliability": "Medium",
      "query": "PREFIX rm: <http://purl.oclc.org/IMPReSS/rm#>
PREFIX ex: <http://example.com/ns#> SELECT ?resource WHERE {
?resource rm:actsOn rm:Light ; rm:associatedTo ex:dressingRoom_
10 .}"
    },
    {
      "resource specification ID": "2",
      "access scheme": "Shared",
      "significance": "Useful",
      "reliability": "Medium",
      "query": "PREFIX rm: <http://purl.oclc.org/IMPReSS/rm#>
PREFIX ex: <http://example.com/ns#> SELECT ?resource WHERE {
?resource rm:monitors rm:Occupancy ; rm:associatedTo ex
:dressingRoom_10 . }"
    }
  ]
}
```

Figure 2. Application description example.

3.2 Runtime representation

At system runtime application descriptions are stored into the system knowledge base as part of the resource management ontology for mixed-criticality resource management. A simplified view (only classes and object properties are depicted) of this ontology is presented in the Figure 2.

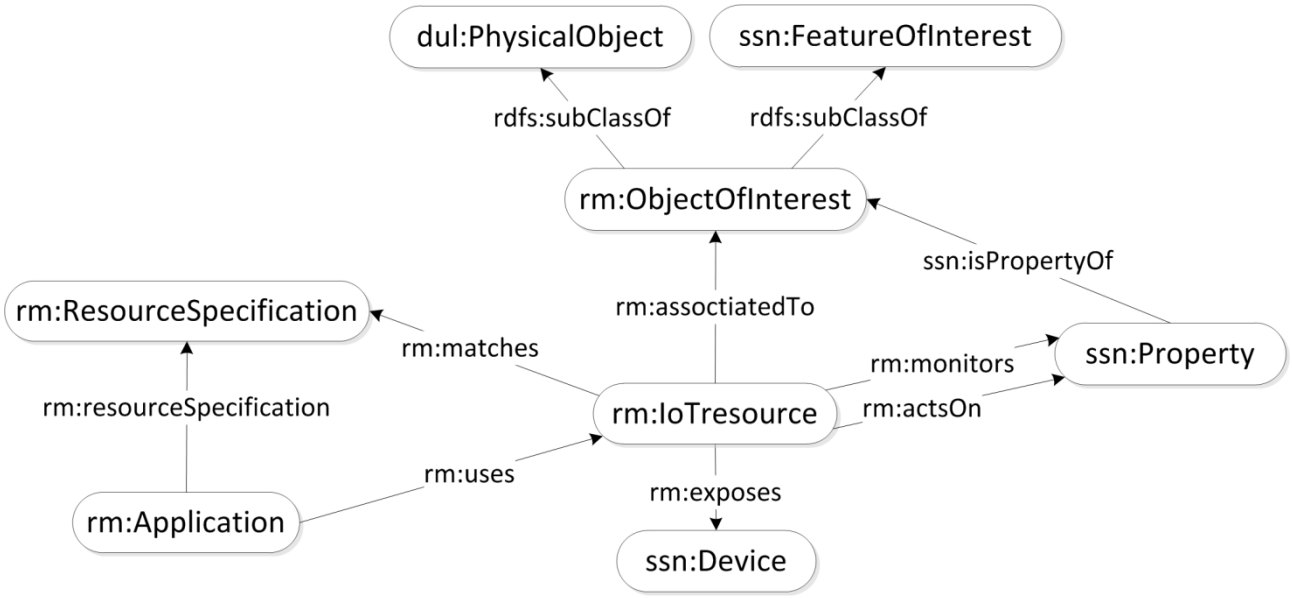

Figure 3. Simplified view of the resource management ontology.

The ontology has three main concepts: *ObjectOfInterest*, *IoTresource* and *Application*. The *ObjectOfInterest* class is a subclass of *dul:PhysicalObject* (Gangemi 2007) and *ssn:FeatureOfInterest* Compton *et al.* 2012) classes. It represents all kinds of physical world objects (e.g. rooms, items, people, etc.) that are relevant for the given IoT system. The idea is that domain specific ontologies are utilised and created for modelling the properties of object of interest in more detail when necessary.

The *IoTresource* class represents a certain sensing or actuating capability of a device. The link between the *IoTresource* instance and the *ObjectOfInterest* instance it is attached to is represented 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. In addition to these object properties, the *IoTresource* has data properties for representing the reliability level it provides and the security level required to access the resource.

The *Application* class represents a business logic that utilises device capabilities in order to provide certain functionality for the system (e.g. turn on the lights when person enters a room). The resources an application needs to access are represented as instances of the *ResourceSpecification* class. Each resource specification has several data properties that define the functional specification, access scheme, significance and reliability requirement for the resource. The relation between suitable IoT resources (one that matches the functional specification) and the specification are represented with *matches* object property. The relation between an application and IoT resource that is currently reserved for the application is modelled with *uses* object property. In addition to these object properties, the Application class instances are modeled with rm:criticality and *rm:securityClearance* data properties.

To illustrate the ontology in practise, an example of runtime content of the System Knowledge Base (SKB) is depicted in the Figure 4. In this example the RDF graph inside the SKB consist of an object of interest, application and two IoT resources. The object of interest is a dressing room in the Opera House (Theatro Amazonas). The application is the same automatic light control application presented in the section 3.1. The IoT resources deployed into the room are Philips Hue lighting system and occupancy sensor. For simplicity reasons the devices domain specific properties of the object of interest or the devices exposed by the IoT resources are not illustrated in the figure.

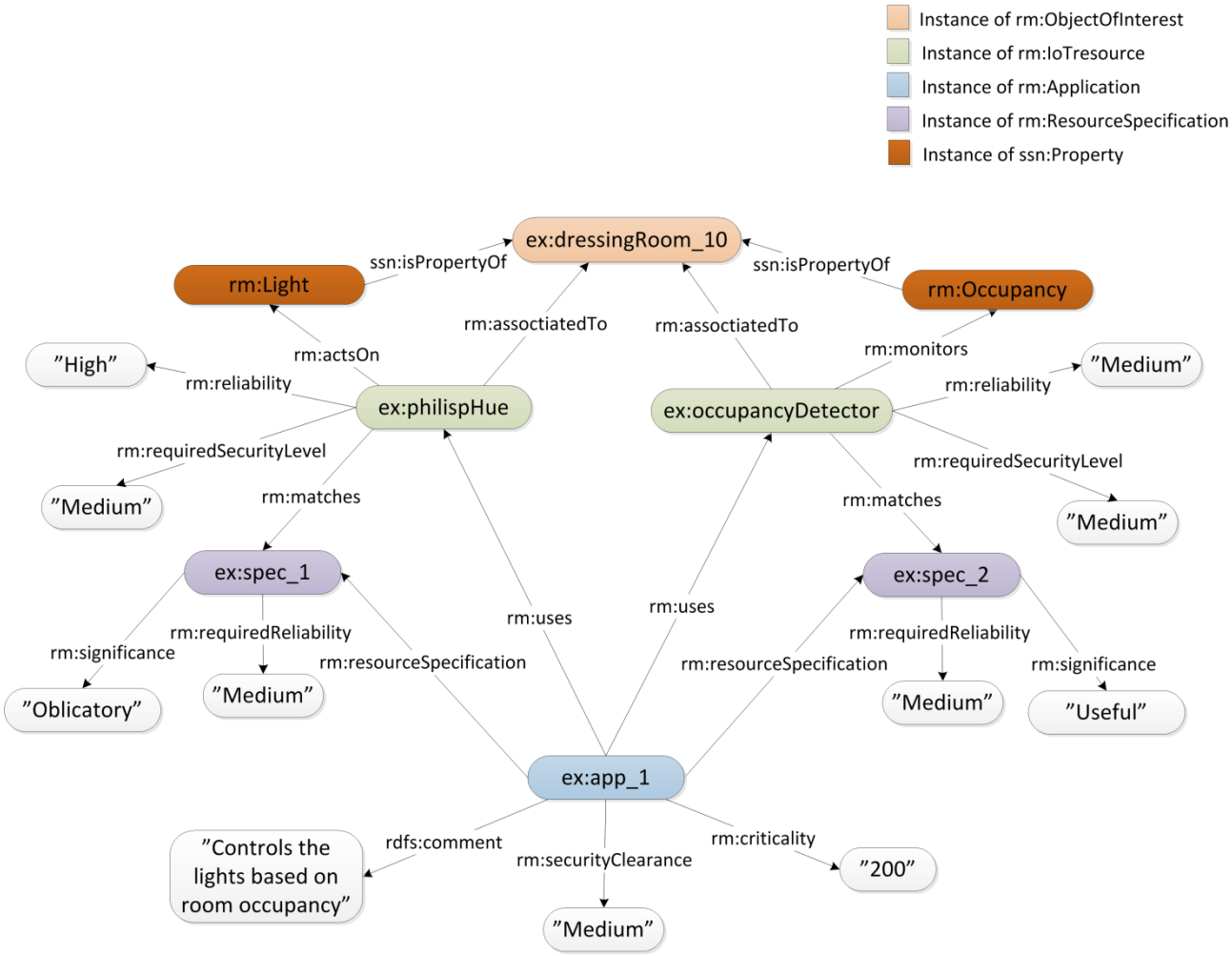

Figure 4. RDF graph illustrating object of interest, application and two IoT resources.

4. Application Description Generator

4.1 Use Cases

As elaborated in the introduction, to use the resource management capability for mixed critical applications, the developers have to create an application description containing the application resource requirements, which are used to register his application to the resource manager. The resource manager then stores the information about the application in the knowledge base, and finds the suitable resources that meet its needs. The application description must follow a format that can be understood by the resource manager. As a proof of concept, the resource manager uses a JSON formatted description, shown in Figure 2. JSON format was chosen since it can be compiled and processed by devices with limited computing resources. The application description contains the basic information about the application, required for the registration purpose. It also contains the SPARQL query, which will be executed by the Global Resource Manager and the suitable resources for the application is then found.

As we interviewed several developers, we found out that many developers are not familiar with RDF and SPARQL. This makes crafting the application description manually by hand maybe error prone. Consequently, we designed and created a graphical tool to support the developers creating the application description. Using the tool, they only need to enter the required information through a graphical user interface. The tool then generates the application description in a JSON formatted file based on the input. The tool is implemented as an HTML5 based application which can be embedded in the more professional IDE such as Eclipse or Visual Studio.

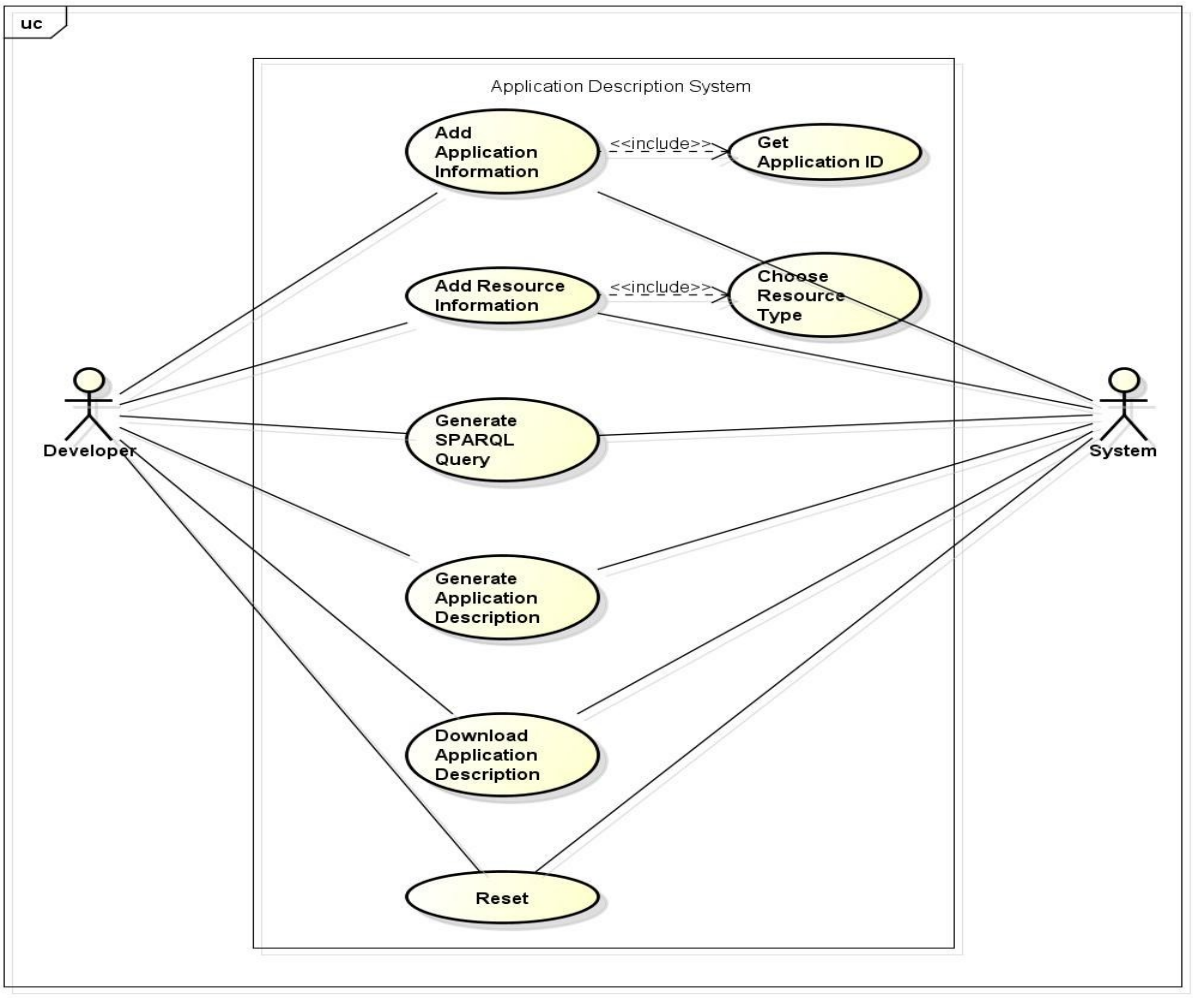

Figure 5. Use case diagram for the application description generator.

We identified more detail use cases on how the developers interact with the application description generator as the following:

1. The developer of the application has to enter the information of the application and resources in a form provided by the tool.
2. The information of the application is converted from form data into JSON format.
3. If the application can only be bound to resources with specific IDs, the developers may enter the unique ID of the resources or choose from a list of the available resources.
4. If the application requires resources with specific requirements, the developers may enter the requirements as key-value pairs in the form. The list of the requirement is then converted from into a SPARQL query, which is embedded in the JSON-formatted application description.
5. The developer is able to download the application description as a JSON file, which can be deployed together with his application.
6. The developer is able to reset the form, remove specific resources, and remove the key-value requirements.

4.2 User Interface Design

Based on the use cases, we created the user interface mockup that reflects the required features. As shown in Figure 6, the developers could enter the criticality level of the application, add resource requirements, which consist of more detail critical level, access scheme, and different properties of the resources, e.g., latency, accuracy, etc. In the future, these properties will be filled automatically depending on the ontology used by the resource manager, and the resource list will be connected to the discovery manager which is done in WP3.

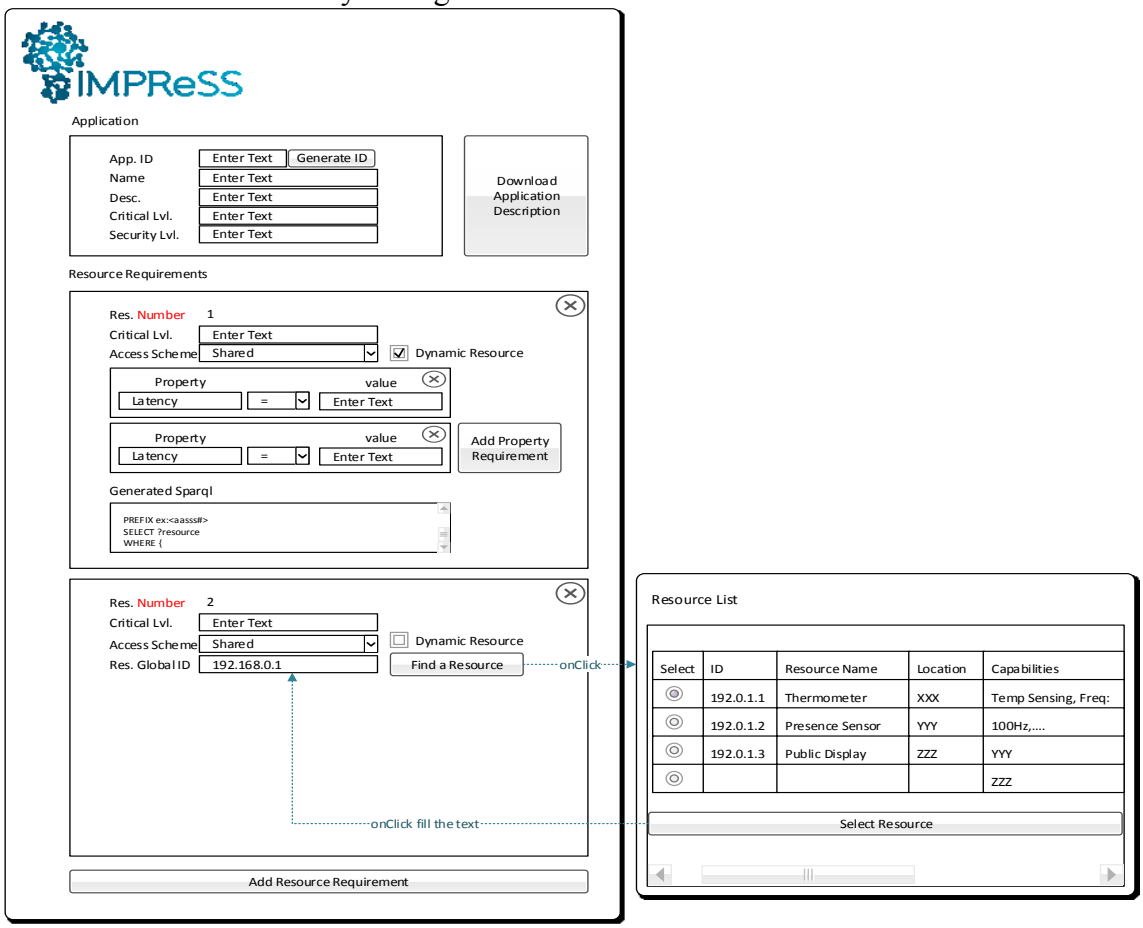

Figure 6. Application description generator mockup.

4.3 Architecture & Implementation

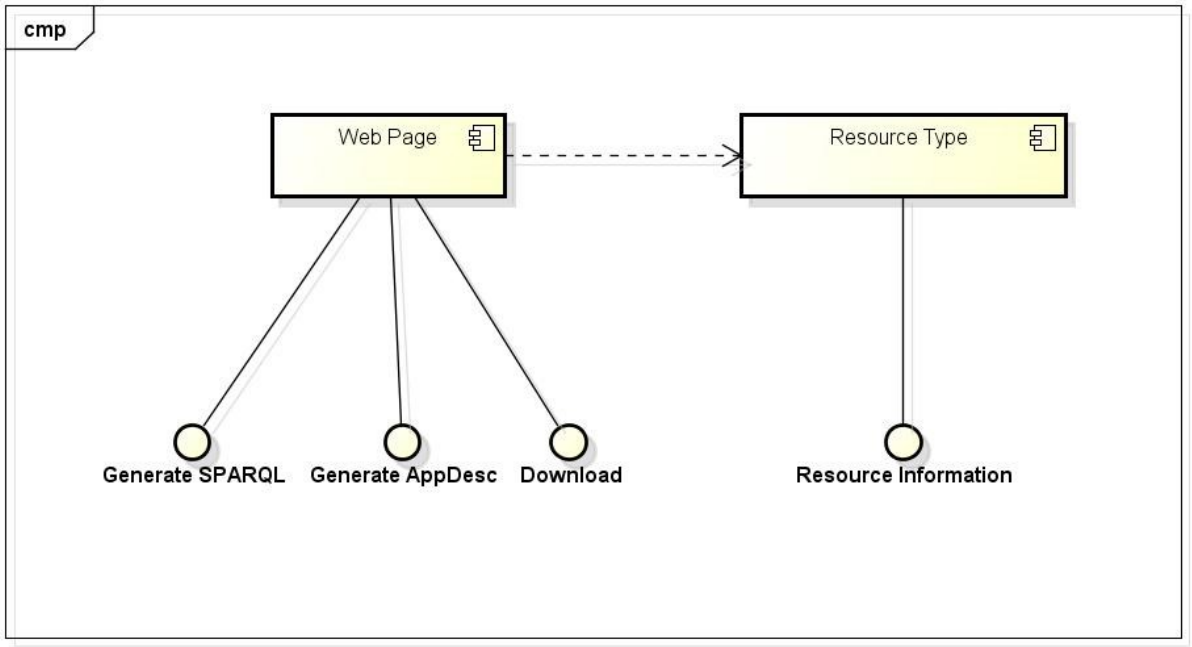

Figure 7. Component Diagram

The architecture of the application description generator tool has six main modules including:

- Application ID Generator
- Generate SPARQL Query
- Generate Application Description
- Enabling/Disabling Module
- Download Application Description
- Reset the form

Application ID Generator Module:

Application ID for each new application is generated by this module. The application ID of an application is alphanumeric and unique. The length of the ID is 32 bits. When the form is loaded for the first time, the application ID is generated automatically and it is randomized. This is done by a for loop and each time it chooses a random character from the list and the same is done for 32 time in order to generate a 32 bit length application ID. The list contains numbers and alphabets. After the developer finish generating the application description, the developer will press the reset button to reset all the values of the form in order to generate the application description for the next application. During this time, the application ID also will be reset. To get a new application ID, the developer has to press get app id button. Now again, a new unique and random application ID with 32 bits will be generated. By the end of the project, this module will be connected to the Global Resource Manager to avoid duplicate IDs being generated.

Generate SPARQL Query Module:

One of the main objectives of this tool is to generate a SPARQL query from the form data. This is achieved by the Generate SPARQL Query module. The developer is asked to fill a form which has details about the resources. After entering the details, the developer clicks the SPARQL Generator button to generate SPARQL query from the form data. This is done by analyzing the details which is being entered by the developer. Before going to the analysis part, it is good to know about the

SPARQL structure. SPARQL consists of four parts. They are prefix declarations, query type, projection and graph pattern. In this, prefix declarations are the short forms of the URIs we are going to use. Query type indicates always SELECT, because we are going to fetch the result. Projection will be always the result what we need and in this case it is the resource unique ID always. So, these three parts will not change at all. The only part that will change is the graph pattern. Graph pattern in simple words are the conditions given in a query to fetch the exact resource what the application needs.

Here again, the developer is given a choice in determining the resource type. There are two types of resource types. They are static and dynamic. Resource type is said to be static, when the developer himself knows the resource unique ID which can be its IP address or its MAC address. Resource type is said to be dynamic, when the developer does not know the resource's exact unique ID and instead he knows the properties of the resources. The properties such what is the purpose of the resource, its location, and the like. When the developer selects the resource type as static, he is just asked to enter the unique ID of the resource. This unique ID is then inserted into the graph pattern when SPARQL Generator button is clicked. If the developer selects dynamic as resource type, then the developer has to enter the properties of the resources with its values. The properties are entered in a table which has two columns. One column indicates the property name and the other indicates the value of the properties. The developer can add more properties by clicking the add property button. When doing this, a new row is appended to the table with two columns. After entering the property name and its values, the developer will press the SPARQL Generator button. During this, the total number of rows in the table is calculated and it will be converted into the graph pattern.

There are two strings maintained. One string contains the first three parts of the SPARQL query which does not change. The second string contains the graph pattern which is changed every time according to the developer. Finally, both the strings are merged to get the resultant SPARQL query. The final SPARQL query is displayed in a texarea provided for it. Later, this will be used for the generation of application description in JSON format.

Generate Application Description Module:

The next main objective of the tool is to generate the application description in JSON format from the form data. Now for this the whole form data has to be used for generating data in JSON format. The form data is divided into three sections. The first section contains the information about the application. The second section contains the information about the first resource including the SPARQL query. The third section contains the information of the second resource including the SPARQL query. These three sections has to be grouped together for the JSON format. The JSON format is divided into two parts. They are the application part and the resource part. Since there is only one application, the application part contains one object which holds all the direct information about the application. Since there are two resources, there will be two objects that hold information about two resources. Then these two resource objects are pushed into an array. Now these data is converted into JSON and the resultant JSON format of the form is displayed in a textarea when the developer clicks Generate App Desc button. This will help the developer to verify the data he entered and the JSON format using any JSON validator.

Enabling/Disabling Module:

As already mentioned, the developer is given a choice in choosing the resource type. The developer must be given space for entering appropriate details based on his selection. Other details should not be shown or enabled to him which may create confusion for the developer. For static resource type, the developer can enter the resource unique ID directly in a text box. For the dynamic resource type, the developer can enter the properties of the resources in a table and he can increase the properties with the help of add property button. If the developer gets both the text box and the table enabled at the same time irrespective of his selection, then it may lead him to confusion. In order

to avoid this, one of the above mentioned has to be disabled according to the developer selection. If the developer selects static resource type, then the table with the add property button will be disabled and only the text boy for entering resource unique ID will be enabled. If the developer selects dynamic resource type, then the text box for entering the resource unique ID will be disabled and table with the add property button will be enabled.

Download Application Description Module:

Once the developer has generated the application description, he can download the same as a JSON file. This JSON file will be helpful for the application to register in the knowledge base during deployment and to find its suitable resource. The final JSON format is displayed for developer's reference in a textarea. When the developer clicks the Download button, the value of this textarea which is the JSON format is taken and a window is opened where we can select save option. Then the developer is asked select the file path and to enter the file name along with .JSON extension.

Reset the form Module:

This module resets the form data by erasing the values that is being entered. This will help the developer to enter the information for the next application for which the application description has to be generated. While resetting, the data which is displayed in the textareas are also removed. In the resources section, if the developer has selected resource type as dynamic, then all the rows expect the first row are deleted. And the selection for the resource type is changed to default which is empty selection. Also, the text box for entering static resource ID and the table for entering dynamic resource propertied will be disabled.

4.4 The initial implementation

The initial implementation is depicted by Figure 8. Currently, the tool is able to convert the resource requirements into SPARQL queries and combine them with the other input into a JSON file that can be downloaded and embedded into the application. The application currently is responsible to send this description to the resource manager. However, in the future, this will be done automatically similar to the Meta information of an OSGi bundle.

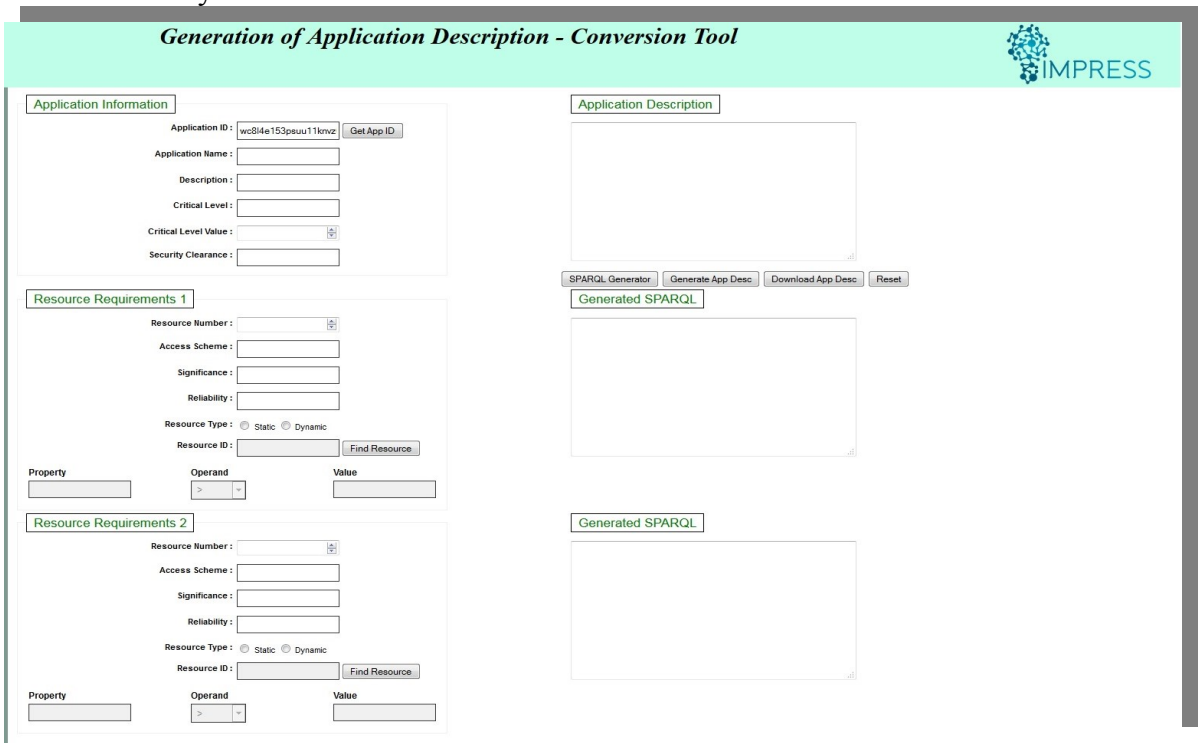

Figure 8. Screenshot of the initial implementation

4.5 Examples of using the generator

In this section we discuss case study of applying the application description generator for the scenarios discussed in the Figure 3.

In the first scenario, a mobile app is used to control devices in the office of an employee. The mobile app use IoT resources that are installed in that particular room. These resources are shared with centralized application that schedule when the resources are on and off. To save energy, we plan to override the static schedule with a schedule that takes into account the personal habit of the users when he is in the office.

Let assume that in the office, the mobile app is able to override the schedule of the lighting, heating, and a personal laser printer which can be switched off from the network. In order to gain access to these devices, the developer needs to fill the information about the application (name, critical level, and description) and the resources that are required as depicted in Figure 9.

On the right side of the tool, the resulting SPARQL queries for each resource requirement are shown. Moreover, the whole application description is also shown and can be edited when required.

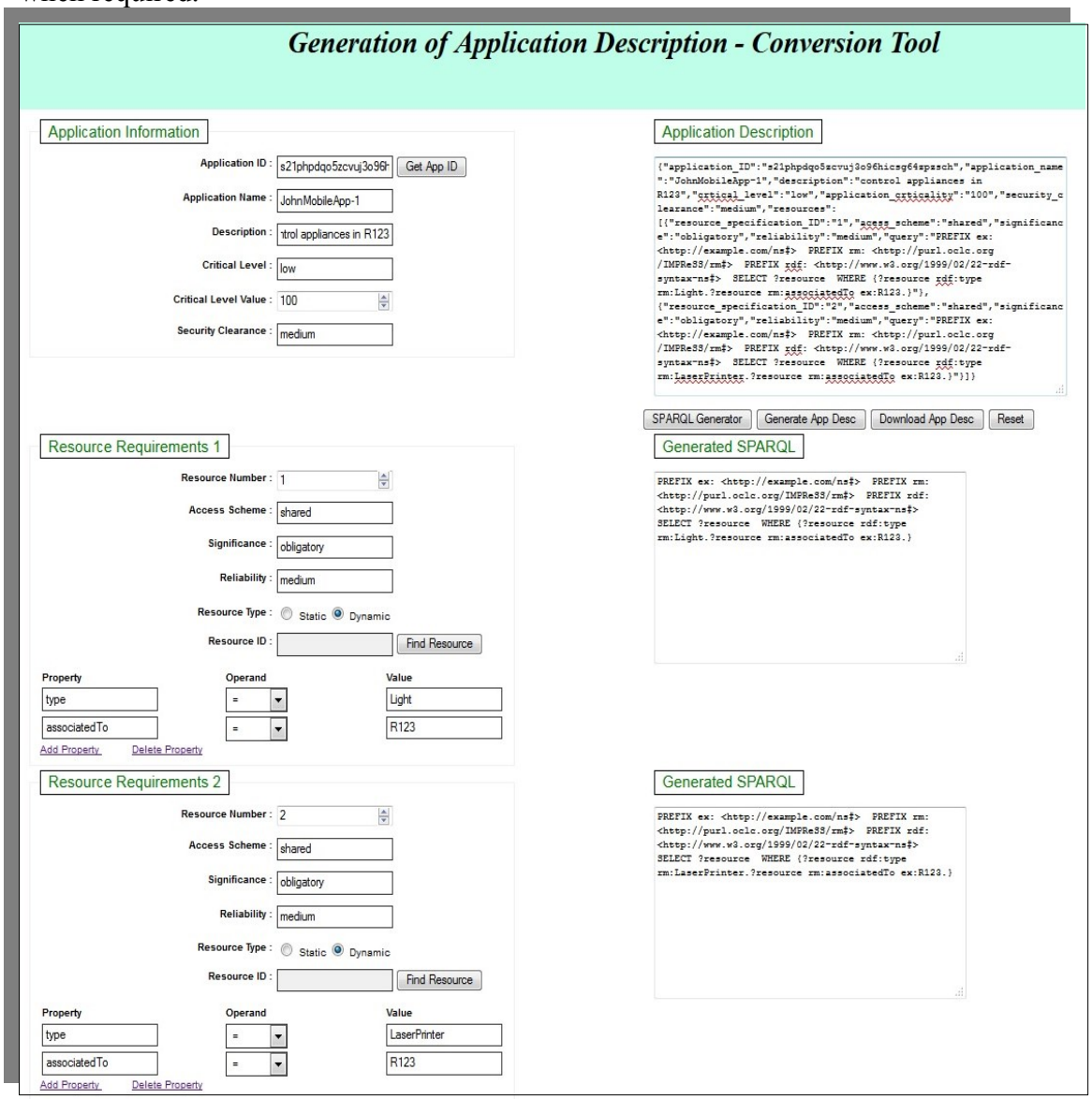

Figure 9. Example of Using Application Description Generator.

```
{
        "application_ID": "s21phpdqo5zcvuj3o96hicsg64zpzsch",
        "application_name": "JohnMobileApp-1",
        "description": "control appliances in R123",
        "crtical_level": "low",
        "application_crticality": "100",
        "security_clearance": "medium",
        "resources":[
        {
                        "resource specification ID": "1"
                        "access scheme": "shared"
                        "significance": "Oblicatory"
                        "reliability": "Medium"
                        "query": "PREFIX rm:<http://purl.oclc.org/IMPReSS/rm#>
                                  PREFIX ex:<http://exampel.com/ns#>
                                  SELECT ?resource WHERE{
                                         ?resource rdf:type rm:Light.
                                         ?resource rm:associatedTo ex:R123
                                  }"

        },
        {
                        "resource specification ID": "2"
                        "access scheme": "shared"
                        "significance": "Oblicatory"
                        "reliability": "Medium"
                        "query": "PREFIX rm:<http://purl.oclc.org/IMPReSS/rm#>
                                  PREFIX ex:<http://exampel.com/ns#>
                                  SELECT ?resource WHERE{
                                 ?resource rdf:type rm:Heater.
                                 ?resource rm:associatedTo ex:R123
                        }"
        },
        {
                        "resource specification ID": "3"
                        "access scheme": "shared"
                        "significance": "Oblicatory"
                        "reliability": "Medium"
                        "query": "PREFIX rm:<http://purl.oclc.org/IMPReSS/rm#>
                                  PREFIX ex:<http://exampel.com/ns#>
                                  SELECT ?resource WHERE{
                                 ?resource rdf:type rm:LaserPrinter.
```

Figure 10. Application Description

During the development time, the developers may not know how the IoT resources and his applications will be deployed. Consequently, the association between the applications and the resources could be done after the deployment. To allow the mobile app to find the relevant resources, the three devices (lighting, heater, and printer) are annotated with device description as shown in Figure 11. These device description are sent to the resource manager, which then stored in the knowledge base. The most basic way to bind applications with IoT resources is through the device type and its association as depicted in Figure 10.

Given that devices may be sensors, actuators, displays, communication devices or many others, this can be a possible way of classifying them in types. But it is also possible that the devices used in a specific situation are all sensors, or all displays, thus it is not useful to enforce a classification by type that does not contribute with information to differentiate devices in all cases. Furthermore, there can be many classifications of sensors according to

different categories that may be relevant for certain cases and not others. Similarly, there will be different classifications of actuators, displays and communication devices that are interesting in different situations. Since it is not known in advance which ones will be relevant in each case, the granularity of the type classification should not be imposed by the base ontology. For example, the granularity could be very coarse, e.g. "sensor", "actuator", "display" and others alike. It could be slightly finer, e.g. "stepper motor", "light sensor", "computer monitor", "land line" and others alike. It could be even finer, e.g., if there were only actuators connected, the types could be "electrical", "electromechanical", "electromagnetic", "smart material", "micro", "nano" and others alike. Finally, it could be as fine as it is convenient, e.g., if there were only stepper motors connected, the types could just be "unipolar" and "bipolar". Depending on the case, a different type classification could be chosen. Thus, the granularity of device types should be left open and unrestricted.

In any case, the base ontology should contain a resource named Type, and its instances should contain the classification types that are assigned by the developers. This definition of device type is broad and refers to any classification of devices that is meaningful and useful in the application domain.

```
{
        "resource ID" : "Light-X111"
        "type" : "Light"
        "associatedTo" : "R123"
        "description" : "Ceiling lamp in R123"
        "capability" : "illumination", "dim"
}
{
        "resource ID" : "Heater-X122"
        "type" : "Heater"
        "associatedTo" : "R123"
        "description" : "Heating Unit in X122"
        "capability" : "heating"
}
{
        "resource ID" : "Printer-X123"
        "type" : "LaserPrinter"
        "associatedTo" : "R123"
        "description" : "Laser Printer in R123"
        "capability" : "B/W printing"
}
```

Figure 11. Device description.

4.5.1 Deployment

In the deployment, sometimes the system administrator will need to associate the devices with the application or change the associated devices to the application. The "associatedTo" attributes at this point could be added or updated to bind the IoT resources to some location or object of interest that is to be measured or affected as shown in Figure 12. It shows that a smart plug is bound to the laser printer. The printer is also bound with the room R123.

Having the information about how IoT resources are associated with the domain objects, allow application to find the required sensors and actuators to run the application logic. Binding the application and the required resources is done through the SPARQL query that refer to the required devices. Alternatively, the SPARQL query could also simply be replaced by any specific ID of the device.

Assuming that the available devices are unknown during the application development, the system integrator must assign the association between the application and the available devices by redefining the application description to accommodate the available IoT resources. Therefore, the

application description generator could be used by the system integrator to generate the necessary SPARQL query to find the devices.

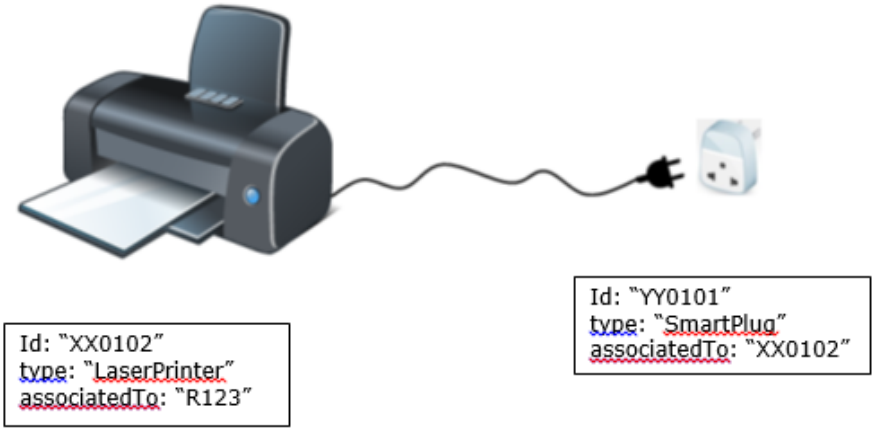

Figure 12. Binding IoTResources to the domain object.

5. Conclusion

Through the resource management in IMPReSS and the application descriptions, we present an approach to address the dynamic binding between IoT resources and the applications. This decouples application logic with the IoT resources and allows the application to share IoT resources.

Our approach considers the following scenarios:

1. The IoT resources are unknown during the development, but known during the deployment. In this case, the system integrator may define a fixed IDs of the required devices and use the application description generator to generate the application description with that information.

The resource manager will bind the application with the specific devices as defined in the application description.

2. The IoT resources are unknown during the development, but also unknown during the deployment since they dynamically enter and leave the environment. In this case, the application developer or system integrator must define the application requirements in terms of the IoT resources. The application description generator provides an easy to use user interface that allows developers without SPARQL knowledge to define the necessary queries.

The resource manager will that use these queries to find the best possible IoT resources for the application.

To ease the developers' task in describing the applications' requirement, we created a web based application. The developers only need to fill the form on the application without having to care about the file format which is required by the resource manager. The application description generator can then generate a file consisting the application requirements expressed in JSON and SPARQL queries. The Global Resource Manager uses these queries to find the most suitable resources for the applications by considering their critical levels, as well as balancing the load among IoT resources.

6. References

Use the following style for references.

(EC, 2007) European Commission (2007). A lead market initiative for Europe. Brussels. COM(2007) 860 final.

(Milagro et al 2008) Milagro, F., Antolin, P., Kool, P., Rosengren, P., Ahlsén M. (2008). SOAP tunnel through a P2P network of physical devices, Internet of Things Workshop, Sophia Antopolis.

(Chen et al 2007) Chen, Y.C., Liu, C.H., Wang, C.C., Hsieh, M.F. (2007). "RFID and IPv6-enabled Ubiquitous Medication Error and Compliance Monitoring System", 9th International Conference on e-Health Networking, Application and Services, 2007, 19-22 June 2007 Page(s):105 - 108.

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

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

Appendix A: Resource Management ontology

```
@prefix rm: <http://purl.oclc.org/IMPReSS/rm#> . # This is not a valid URI (namespace) yet.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix    xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix dul: <http://www.loa-cnr.it/ontologies/DUL.owl#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .


# Classes
rm:ObjectOfInterest rdfs:subClassOf dul:PhysicalObject, ssn:FeatureOfInterest ;
                rdfs:comment "Physical object that is relevant for the IoT system (e.g. person, room, product)" .
rm:IoTresource a rdfs:Class ;
            rdfs:comment "IoT resource that exposes data or cabapility of a device. " .
rm:Application a rdfs:Class ;
             rdfs:comment "A program that runs in the IMPReSS system and realises certain domain logic by invoking services
provided by the IoT resources" .
rm:ResourceSpecification a rdfs:Class ;
            rdfs:comment "A specification of IoT resource." .

# Datatypes
rm:securityLevel a rdfs:Datatype;
              owl:oneOf ("Untrusted" "Low" "Medium" "High" "System").

rm:reliabiliyLevel a rdfs:Datatype;
              owl:oneOf ("Low" "Medium" "High").


# Properties
rm:resourceSpecification a owl:ObjectProperty ;
             rdfs:domain  rm:Application ;
             rdfs:range   rm:ResourceSpecification ;
             rdfs:comment "A relation between application and IoT resource specification." .

rm:uses a owl:ObjectProperty ;
             rdfs:domain  rm:Application ;
             rdfs:range   rm:IoTresource ;
             rdfs:comment "A relation between application and IoT resource and it uses." .

rm:criticality a owl:DatatypeProperty ;
            rdfs:domain  rm:Application ;
            rdfs:range xsd:integer.

rm:securityClearance a owl:DatatypeProperty ;
                rdfs:domain  rm:Application ;
                rdfs:range rm:securityLevel .

rm:requiredSecurityLevel a owl:DatatypeProperty ;
         rdfs:domain  rm:IoTresource ;
         rdfs:range  rm:securityLevel .

rm:associatedTo a owl:ObjectProperty ;
             rdfs:domain  rm:IoTresource ;
             rdfs:range   rm:ObjectOfInterest ;
             rdfs:comment "A relation between IoT resource and object of interest it is associated with." .

rm:exposes a owl:ObjectProperty ;
         rdfs:domain  rm:IoTresource ;
         rdfs:range  ssn:Device ;
         rdfs:comment "A relation between IoT resource and device whose capabilities it exposes for applications." .

rm:monitors a owl:ObjectProperty ;
             rdfs:domain  rm:IoTresource ;
             rdfs:range   ssn:Property ;
             rdfs:comment "A relation between IoT resource and property it provides data about." .

rm:actsOn a owl:ObjectProperty ;
```

```
                rdfs:domain  rm:IoTresource ;
                rdfs:range   ssn:Property ;
                rdfs:comment "A relation between IoT resource and property it provides means to act on." .

rm:matches a owl:ObjectProperty ;
                rdfs:domain  rm:IoTresource ;
                rdfs:range   rm:ResourceSpecification ;
                rdfs:comment "A relation between IoT resource and resource specification it matches." .

rm:reliability a owl:DatatypeProperty ;
          rdfs:domain  rm:IoTresource ;
          rdfs:range rm:reliabiliyLevel ;
          rdfs:comment "A relation between IoT resource and reliability level it provides." .

rm:requiredReliability a owl:DatatypeProperty ;
                rdfs:domain  rm:ResourceSpecification  ;
                rdfs:range   rm:reliabilityLevel ;
                rdfs:comment "A relation between resource specification and reliability level it requires." .

rm:significance a owl:DatatypeProperty ;
                rdfs:domain  rm:ResourceSpecification  ;
                rdfs:range   xsd:string . # TODO: specific in more detail

rm:accessScheme a owl:DatatypeProperty ;
                rdfs:domain  rm:ResourceSpecification  ;
                rdfs:range   xsd:string . # TODO: specific in more detail

rm:functionalSpecification a owl:DatatypeProperty ;
                rdfs:domain  rm:ResourceSpecification  ;
                rdfs:range   xsd:string ;
                rdfs:comment "SPARQL format specification for the resource functional capabilities".

# Individuals
rm:Sound a ssn:Property ; #check if any sound ontologies are available and make a referene to those
        rdfs:comment "A vibration that propagates as a typically audible mechanical wave of pressure and displacement, through
a medium such as air or water." .
rm:Light a ssn:Property ; #check if any light ontologies are available and make a referene to those
         rdfs:comment "A radiant energy, usually referring to electromagnetic radiation that is visible to the human eye, and is
responsible for the sense of sight. " .
rm:Temperature a ssn:Property ; #check if any temperature ontologies are available and make a referene to those
        rdfs:comment "A a comparative objective measure of hot and cold. " .
rm:Humidity a ssn:Property ; #check if any temperature ontologies are available and make a referene to those
        rdfs:comment "The amount of water vapor in the air " .
rm:Occupancy a ssn:Property;
        rdfs:comment "The state of being an occupant or tenant " .
```