Target Outcome: b) Sustainable technologies for a Smarter Society

(FP7 614100)

D4.1.1 Initial application classification language

March 1, 2014 – 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 1 Initial application classification language_v10.docx
Document version:	1.0
Document owner:	Ferry Pramudianto (FIT)
Work package:	WP4 – Mixed Criticality Resource Management
Task:	T4.1 Application classification language and tool
Deliverable type:	R
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	4/01/2014	Table of content
0.2	Jussi Kiljander, Janne Takalo-Mattila	15/01/2014	First version for sections 2, 3 and 4
0.3	Ferry Pramudianto	26/01/2014	First version for sections 1, 5, 6 and 7
0.4	Jussi Kiljander, Janne Takalo-Mattila	3/02/2014	Contributions and modifications to sections 2, 4, 5, and 6.
1.0	Ferry Pramudianto	14/02/2014	Finalizing the document

Internal review history:

Reviewed by	Date	Summary of comments
Stenio Fernandes	11/02/2014	Accepted with minor corrections and comments
Enrico Ferrera (ISMB)	2014/02/07	Accepted with minor corrections and comments

Index:

1 Executive summary

This deliverable describes the initial proposal for mixed criticality resource management in the Internet of Things (IoT) scenario. The approach is based on a central resource manager that which controls and schedules the access to the devices, services and subsystems based on the criticality of the applications.

The resource manager stores the information of the resources in a knowledge base that contains representations of resources including functional capabilities and non-functional properties such as CPU and memory utilization and end-to-end network delay. This knowledge base is used by the global level resource manager to selects the most suitable resource for each application.

The local resource manager monitors the local resource utilization and reports this to the global resource manager. Moreover, the local resource manager administers the access to the resource in a way that only applications that have obtained a reservation from the global resource manager could access it. The local resource manager will need also to schedule the access to the resource based on the application criticality level.

Enabling the matching between application and requirements, the resources must have a metadata description. Currently we see SSN ontology as a promising solution since it provides a basic schema that we can extend to describe actuators and domain specific quality parameters.

The reminder of this deliverable is organized as the following. Firstly, in Section 2 we introduce the background of the problem in the existing mixed criticality systems, the current approaches and how these differs from the mixed criticality in the Internet of Things scenario. In Section 3, we present the domain model of the mixed criticality system for IoT scenario. Section 4 contains the IMPReSS requirements and the design of the resource management in detail, the communication among global & local resource manager, the resources, and the system knowledge base. In section 5, we elaborated the overview of ontology used to classify sensors and actuators, and we elaborated example of required parameters that must be matched to the device descriptions. Finally in Section 6 we provided our summary and conclusion.

2 Introduction

Mixed criticality (MC) system is a system that can execute several applications while guaranteeing their differing criticality requirements (e.g. real-time, performance, security, safety, etc.). Traditionally, MC is applied within single computing platform (single or multi-core) where processor (sometimes also memory) is the resource to be shared between different applications or task.

In traditional MC systems, encapsulation is important, since a failure of micro components of a non-safety-critical application subsystem must not cause the failure of application subsystems of higher criticality. The focus of a safety-critical applications lies on the simplicity and determinism in order to facilitate thorough verification and validation. On the other hand, non-safety-critical applications can provide more complex application services, for instance, they need to deal with insufficient prior knowledge about the environment.

Mixed criticality systems must ensure that upon incremental integration of subsystems, the prior services of already existing subsystems are not invalidated. Composable and incremental certification will require a means of identifying and assessing the "certifiability" of a component, even before it is implemented(Baruah, Li et al. 2010). This call for both formal and analytical methods that seek to define a design process that could produce certifiable systems. In practice, this is very challenging since, for instance, some avionic systems have the ability to reconfigure themselves under certain failures or contingencies, which yield the challenge of certifying reconfigurable systems from a cost, effort, and complexity standpoint. Modern mixed criticality systems need to exhibit a high degree of fault tolerance. However, achieving this in avionics system is particularly difficult.

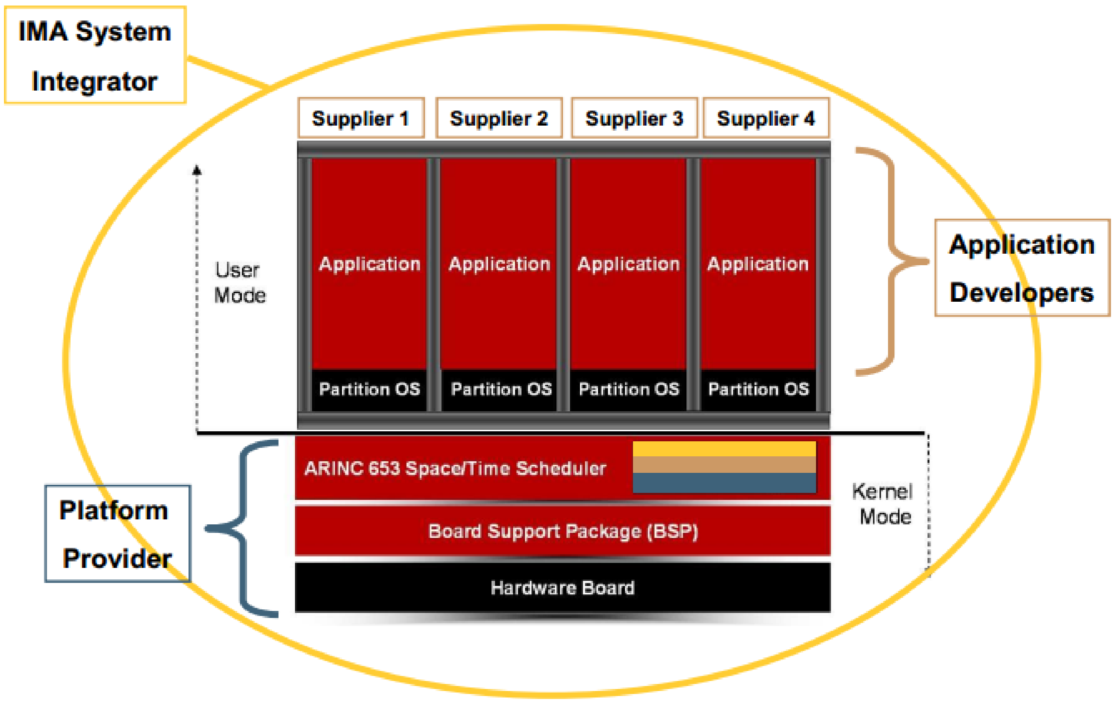


Figure 1. ARINC 653 RTOS Architecture

As depicted in Figure 1, a standard approach in avionics system is to use ARINC 653 (ARINC specification 653) time and space partitioning which partitions the processor based on isolated operating systems. The access to the hardware resources is administered by the ARINC 653 time/space scheduler. Inside each partition, a priority-preemptive scheduler is used to ensure that the tasks with highest priority are executed first.

In this project, we extend the MC idea from single device to Internet of Things (IoT) domain where the resources to be shared are real world objects (*e.g.* sensor, actuators, *etc.*) present in a given IoT environment. In addition to the resource type (i.e. sensors vs. processor) the main difference to the traditional MC system is that the IoT the systems consist of numerous different devices that are connected through best effort networks.

There are two main motivations for adopting MC for IoT applications. First, it is costly to deploy new physical resources for each new application deployed in an IoT environment and by adopting mixed criticality technologies IoT applications may share these resources without jeopardizing the required responsiveness. Second, in many use case scenarios it is not possible to deploy new actuator resources for each individual application because the functionality provided by the resources would cause interference between the applications. For example, it is not feasible to deploy new lights to the same physical space for each application that needs to control them because they would still interfere with each other in the real world.

The Internet of Things domain imposes many challenges to the resource sharing that are not typically present in traditional MC systems. These challenges are mainly caused by the open and heterogeneous nature of IoT. For instance, IoT environments are typically very dynamic which means that the system evolves and changes constantly. It is typical that new devices and applications emerge at runtime such as mobile devices, or devices that are added to upgrade the system. In IoT systems we do not typically want to restrict the applications and devices that can be part of the system in the future (i.e. we do not want to create closed systems). Because we cannot a priori define the type or functionality of the devices and applications that might emerge to the system in the future the approach for resource sharing needs to be also flexible and extendible for future needs. The heterogeneity of computing platforms makes the communication and especially security management a difficult task. Because of the evolving nature of the IoT systems and the security solutions also need to be dynamically adjustable to the given situation of the system. The fact that typical devices (and networks) in IoT are resource constrained makes this even more challenging.

In addition to the open and heterogeneous nature of IoT, the fact that IoT builds upon best effort networks and communication solutions (e.g. wireless networks, IP networks) makes the mixed criticality in IoT (depicted in Figure 2) a very different from the traditional MC systems whose components are connected in a closed network. Although there have been solutions for prioritizing network flows that are used for real-time traffics (e.g. multimedia traffic), we still do not have a full control of different network configurations along the path within the internet. Consequently, hard real-time requirements typically present in traditional MC systems cannot be guaranteed.

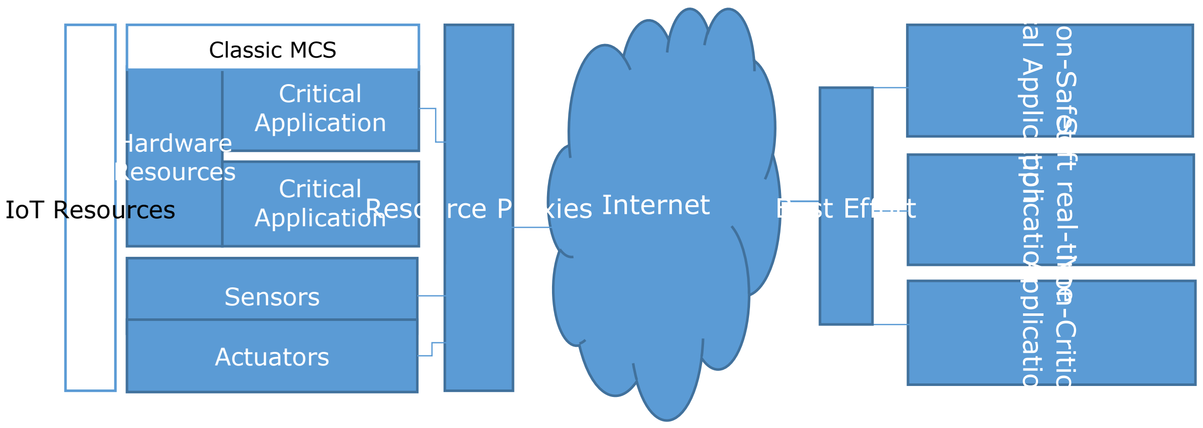


Figure 2. Mixed Criticality in an IoT Scenario

Providing a best effort mixed criticality for IoT, we propose an approach where access to the services provided by the IoT resources can be differentiated based on the criticality of the application. Enabling this approach the resources and applications are described with expressive and flexible representation format and the access to the resources is controlled and scheduled by a common Resource Manager (RM) component.

In this deliverable we will outline the overall approach for MC resource management in IoT. The main focus in this deliverable will be on the application description language that describes the applications in terms of criticality, security, and resources it needs to access.

3 Domain Model for Mixed Criticality in Internet of Things

Mixed criticality in the domain of Internet of Things is a totally new concept and to understand what it actually means (and what are the challenges), we will first present our vision for a domain model of mixed critical IoT systems (depicted in the Figure 3).

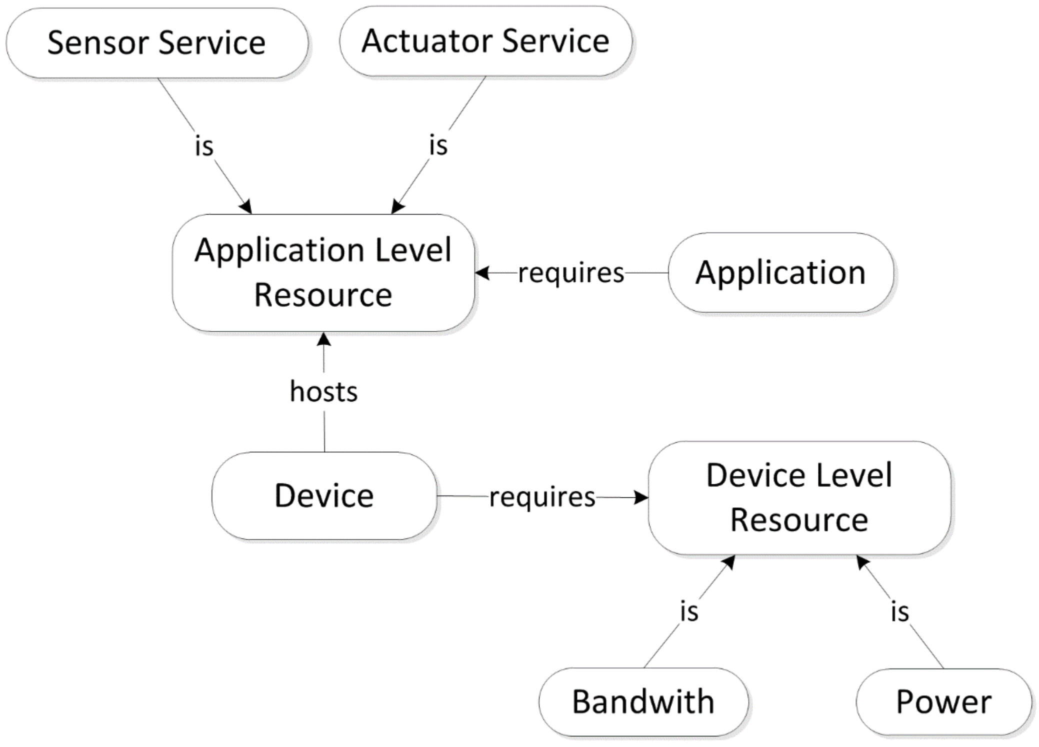


Figure 3. Domain model for mixed criticality in Internet of Things.

According the Internet of Things – Architecture (IoT-A) project (Carrez 2013) the fundamental goal of the IoT is to enable users (*i.e.* humans or software agents) to interact with the physical world. To make this interaction possible there are needs for means to monitor the current state of the physical world, and for methods to modify the state of the real world when needed. In practice, this interaction is achieved via devices that can be divided into two groups based on their role in the system:

- **Sensors:** Sensors are devices that sense and measure a real world phenomenon, make decisions based on the measurements and communicate with other parts of the IoT system using wired or wireless communication methods. Temperature sensors and smoke detectors are examples of sensor used in typical buildings.
- **Actuators:** Actuators are devices that modify the state of the physical world by taking commands from other parts of the system using different kinds of communication methods. Air condition systems, automated doors and lights are examples of typical actuators.

Similarly to IoT-A Architectural Reference Model (ARM) we refer the formal interface between the user and the device as *Service*. Thus, there exists two types of generic Services in our domain model. First a service that represent the functionality of actuators and Services that represent the functionality of sensors.

The business logic (or domain logic) defining how the user interacts with the environment in a specific use case is defined and implemented in applications (*i.e.* software programs) developed with the IMPReSS platform. In order to realize the use case specified in the application logic, the application needs to access the sensor and actuator services present in the particular IoT system. Therefore, the sensors and the actuators are referred to as *Application Level Resources* in the domain model. The capabilities of actuators and sensors to serve different applications in IoT systems are limited. Additionally, typical use cases require some kind of coordination between the applications to be useful. For instance, lighting can be controlled by many applications, but in order to provide a meaningful service for the end-user, we need to limit which applications can control the actuators. Therefore, we need to be able to define the level of criticality and priorities for different applications and provide means for managing and scheduling the resource access in order to make the performance of the most critical applications as good as possible.

In addition to the sensor and actuator services, traditional resources such as processor and memory provided by a computing platform can be also considered as *Application Level Resources* (*i.e.* these are resources required by the software implementing the application logic) in the domain model. However, since it is not typically relevant where the logic of the IoT application is deployed (*i.e.* the application logic of different application do not need to be deployed on same devices and the applications can even run in the cloud), we do not consider these traditional MC resources in this project.

Resource such as energy, power, current, and bandwidth not directly used by applications are also important for IoT systems. We refer to these resources as *Device Level Resources* because they are resources required by the computing platforms that either host the *Application Level Resource* or provide some other useful functionality for the user (*e.g.* fridge). Because the *Device Level Resources* are not directly used by applications they are not presented in the application descriptions and are thus not in the core focus of this deliverable.

In addition to managing and scheduling the resource access, security is also important part of mixed criticality systems. This is especially true in IoT, because the applications interact with the physical world and security attacks can have thus very serious consequences. The IMPReSS system needs to meet the following well-known security principles (Waltenegus and Poellabauer 2010):

- **Confidentiality:** only authorized parties are able to access to information. The level of confidentiality needed is not always totally self-explanatory. Patient data in hospital can be easily realized as confident information, but for example in building management momentary electricity consumption might be considered as non-confident information. However in some cases these types of information could be interesting for a potential attacker.
- **Integrity:** unauthorized parties cannot modify the information transmitted in the system. In IoT systems it is vital only the authorized parties are able to modify information and control the devices present in the system.
- **Availability:** critical services are available when needed. In the IoT system the most critical services need to be available at all the time. Preventing denial-of-service (DoS) attacks *e.g.* in the wireless sensor networks perfectly is impossible most of the times. Therefore, the most critical services need to work independently without network access.

In this deliverable the security will be covered as much as is feasible for the application descriptions. The overall approach and architecture for dynamic security management will be presented in the D4.4.

4 Approach to Mixed Criticality Resource Management

There exist several possibilities to implement resource management for the Internet of Things applications. For instance by defining static classifications and set priority based on these classifications. Or to use a reservation system which only allows the application to access to the resources after a reservation is approved.

As an initial proposal to address the challenges and requirements presented in the previous section we design a resource management approach for mixed criticality in IoT where the access to application level resources (*i.e.* sensor and actuator services) is managed by a functional component of the IMPReSS platform, called Resource Manager. In addition to the Resource Manager, the approach for mixed criticality management in IoT consists of tools that help the development, deployment, and configuration of mixed criticality IoT applications.

The whole process related to development, deployment, and runtime management of mixed criticality applications in IoT (depicted in Figure 4) is envisioned as follows. To enable the developer to focus to the development of the business logic for the application and not to let her/him spending time with technical details related, for example, to resource discovery, resource selection, resource access, *and the like*, we propose an approach where an application description is used to represent necessary information (*e.g.* criticality level, security level, resource specifications, *etc.*) about each application. In addition to simplifying the access to application domain resources the advantage of this approach is that, by taking the decision of which resources are used away from individual applications, the functionality of the whole IoT system can be improved (*i.e.* the Resource Manager has a holistic view of the system and it is thus able to optimize the resource usage for greater good of the whole IoT system).

When new application is developed, the developer defines the specification of the application level resources the application needs to access (*e.g.* the application needs to access a temperature sensor that provides the temperature in Celsius with +-0.5 C accuracy). He will also give recommendations for the criticality and security levels of each resource. These recommendations are based on the knowledge the developer has about the application (*i.e.* how important each resource is for the application, how private information the application needs to access, *etc.*). The application development tool will assist the developer in this process and generate an application description that will be used to manage and schedule the resource access. Additionally, the application developer needs to write a short description for the application in human readable format. This description is added to the application description and will be used by users to select which applications are deployed to their IoT environment.

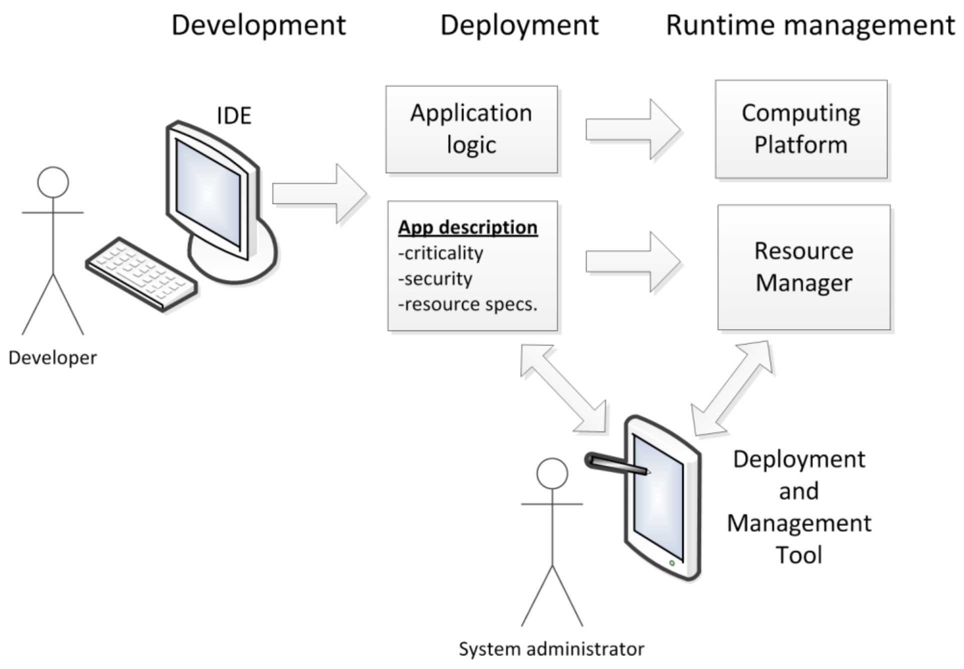


Figure 4. Development, deployment, and runtime management of mixed criticality applications.

The person responsible for deploying the application to the IoT environment (house, school, theatre, city, *etc.*) is called a system administrator. He is the one who selects which applications are deployed to her/his environment and she/he, of course, wants to make sure that the whole IoT system behaves as it is supposed to. Before the application is deployed to an IoT environment, the system administrator will need to set (or adjust) the criticality and security levels for the application (and for each resource the application accesses). This process is done "manually" via the Configuration and Management Tool and the decision will be based on:

- human readable description of the application,
- recommendations given by the application developer,
- the knowledge the system administrator has about the whole IoT environment.

We will also provide ways to modify the criticality and security levels of the application at runtime so that the application can adjust to the dynamically changing requirements of the system. These modifications will be made automatically by domain specific rules defined by the system administrator with the help of the Configuration and Management Tool (it is also possible that the rules are defined by experts and the system administrator only needs to select the most suitable rules for her/his system). This is useful to define application that has different criticalities in different states

In addition to adjusting the security and criticality levels the system administrator can also configure the resource specifications of the application. This is typically needed with general purpose application needs to be tailored to a specific IoT environment. For example, if a developer has implemented a general purpose application that controls fans in a room based on temperature and humidity values, the system administrator could customize the application by modifying the resource descriptions so that the sensors and actuators need to be located in a specific room (*e.g.* living room if she/he wants to control the fans there).

When the application is deployed to the system, the Configuration and Management Tool submits the application description to the Resource Manager component which will use the specification in three ways.

1. Search for suitable resources for each resource specification in the application description. If the SSM is unable to find suitable resources it will inform the system administrator that the application cannot be deployed to the given IoT environment.

2. Manage and schedule the resource access globally and within a single resource to achieve, at the same time, the most optimal resource utilization and to guarantee that the critical applications have access to resources over less-critical ones.
3. Select the security mechanisms and to adjust the security level used in application – resource interaction.

In the development process the software defining the business logic for the application needs also to be deployed in a computing platform and configured to the particular IoT environment. The computing platform where the application logic is deployed does not need to be present in the physical IoT environment (*i.e.* it only needs to able to access the sensors and actuators deployed to the given IoT environment) and it can be, for example, a mobile phone, a resource restricted device, a personal computer, or a server in the cloud. It is also possible that the application logic is already installed to a computing platform and the user just needs to configure the application so that it is able to join the IoT environment. The actual process related to deployment of the application logic (*i.e.* the software) is out of the scope of the work done in this work package. The final deployment of each application is achieved by configuring the application so that it is able to access the Resource Management components of the particular IoT environment. This configuration will be done via the Configuration and Management Tool.

In practice, the logical Resource Manager component is divided into two levels (*i.e.* global and local) and three components, namely *System Knowledge Base, Global Resource Manager,* and *Local Resource Manager*. The runtime architecture for the mixed criticality resource management is depicted in the Figure 5.

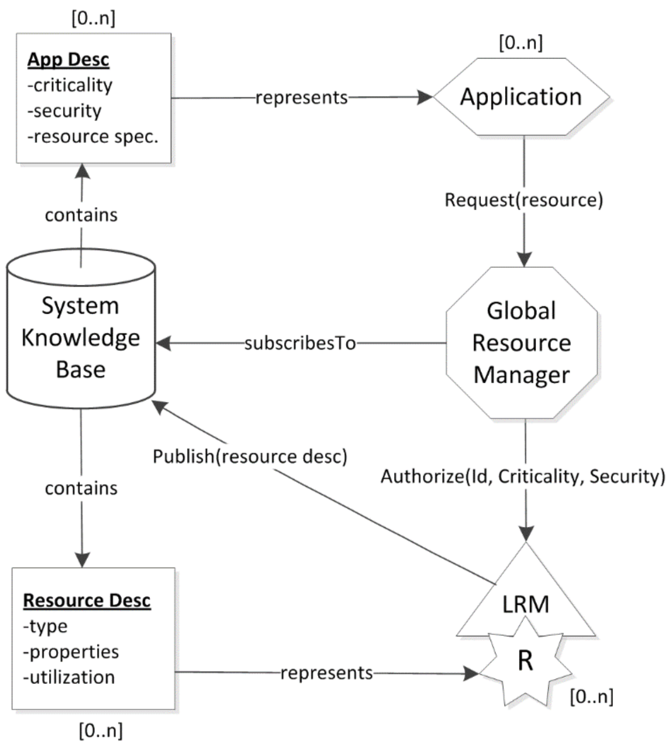


Figure 5. System model for runtime management and monitoring of application level resources.

The *System Knowledge Base* is a component responsible for representing the state of the system at the given point in time. To this end, it acts as a blackboard and provides publish/subscribe based interface for information about objects relevant for the IoT system. We use a Resource Description Framework (RDF) (Klyne, G. and Carroll, J. J. 2004) data model for information representation, because it provides flexible and extendable way to represent information about the system. This is needed because we are neither able to predict what kinds of applications, resources, and devices, for example, will be deployed to the system in the future, nor we are able to predict all the information that is needed to represent these entities. With this approach the developers can also start with small and simple descriptions and extend them when they have gained more understanding of the problem domain without a risk that they lose investments previously made (*i.e.* by using a blackboard pattern and semantic technologies we can avoid the need to redo schemas and interfaces). Another advantage of RDF and semantic technologies are that because they are standard knowledge representation formats there exists "standard" descriptions of sensors and actuators in these formats that can be used directly in the approach. The plan is also to apply techniques that enable subscriptions to events with SPARQL (Harris, S. and Seaborne, A. (eds) 2012) (Harris, S. and Seaborne, A. (eds) 2012) in a performance efficient way to make it possible to manage and monitor the environment in (quasi) real-time.

In the first phase, the *System Knowledge Base* will be used to store information about the applications and application level resources (*i.e.* sensor and actuator services) deployed to the system. Later it can be extended, for example, with descriptions of devices (*e.g.* supported security mechanism, power consumption, performance, *etc.*) and any other type of physical objects relevant for the management and monitoring of the IoT system. In addition to providing extendable and flexible information sharing solution for the resource management components, the *System Knowledge Base* provides interface for other components of the IMPReSS platform to access information about applications, resources, devices, and associations between them. This information can be used, for example, by the monitoring tools to provide a view for the user about the environment or to track various events occurring in the system. It can be also used by the context management components to enrich the context description of the system.

At global level the *Global Resource Manager* controls which resources an application can access. There is a single *Global Resource Manager* assigned for each IoT system. The idea in the global level resource management is to optimize resource usage by selecting most suitable resources for each application at runtime. The application description (presented in the section 5) defines the resource specifications for each application. The *Global Resource Manager* subscribes to the application descriptions published to the *System Knowledge Base* and will be notified when application descriptions are added, removed, or modified. When new application is added to the system, *Global Resource Manager* will subscribe to each resource specification defined in the application description. This way it is aware of the available resources for each application and will be notified when resources are added, removed, or modified. The Figure 6 presents the interaction between the resource management components in the above-described scenario.

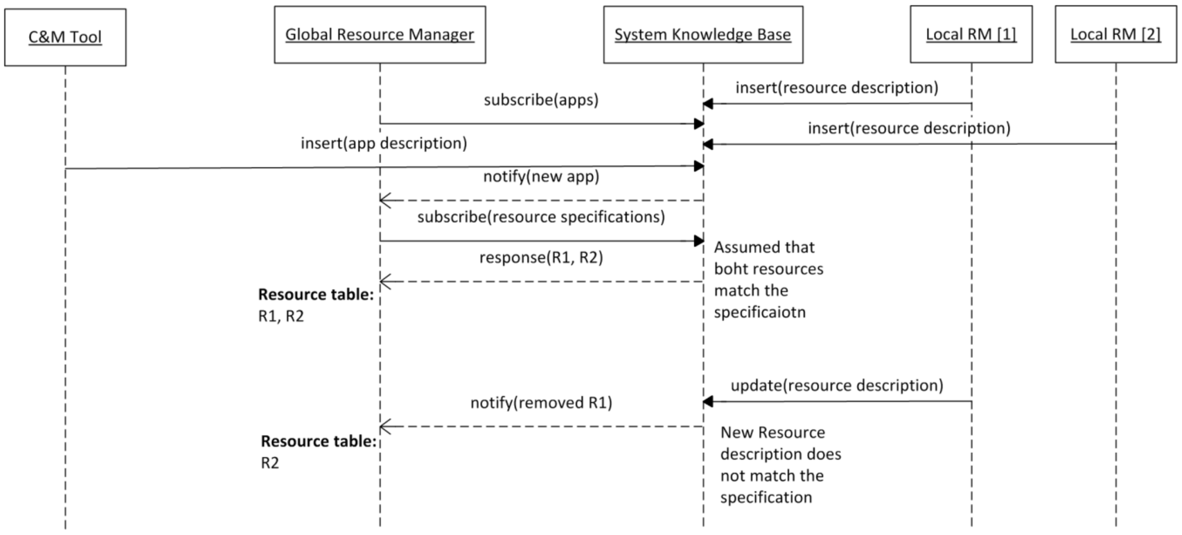

C&M Tool
Global Resource Manager
System Knowledge Base
Local RM [1]
Local RM [2]
insert(resource description)
subscribe(apps)
insert(resource description)
insert(app description)
notify(new app)
subscribe(resource specifications)
response(R1, R2)
Assumed that boht resources match the specificaiotn
Resource table: R1, R2
update(resource description)
notify(removed R1)
Resource table: R2
New Resource description does not match the specification

Figure 6. Interaction between Global and Local Resource Managers and the System Knowledge Base.

The applications get access to resources by making requests to the *Global Resource Manager*. When the *Global Resource Manager* receives a new request to a resource specified in the resource specification it selects the most suitable resource from resources matching the specification and notifies the *Local Resource Manager* and the application about the result of the matching process. In the matching process the *Global Resource Manager* will also define the security methods and levels used in application – resource interaction. The security related aspects will be described in detail in the D4.4. The resource selection process depends on the scheduling approach and algorithms and these will be described in the D4.3. When the *Global Resource Manager* associates an application with a resource it will also publish information about the association to the *System Knowledge Base*. By subscribing to this information, the monitoring tools and other IMPReSS component are able to dynamically follow the state of the IoT system (*i.e.* which resources are accessed by which applications).

The requests made by applications to resources are persistent and the applications need to inform the *Global Resource Manager* when they no longer need the given resource. It is also possible that sometimes the application needs to release a resource for more critical applications. This can happen, for example, when a more critical application with exclusive access scheme make request to the same resource. Additionally, an application may need to release a resource if the utilization rate of the resource rises so high that the more critical applications accessing the resource in a shared mode cannot be served in an appropriate manner. If an application needs to release a resource for more critical application the *Global Resource Manager* will notify the application and a new resource is assigned for the application if possible.

At local level there is a *Local Resource Manager* assigned for each resource. It schedules the resource access within single resources and is only needed for scheduling when multiple applications can access the same resource (*i.e.* when the applications use a shared resource access scheme). The basic principle in the local level resource management is to guarantee that more critical applications are served before less critical ones. In addition to scheduling the requests, the *Local Resource Manager* is responsible for publishing the resource description of the resource it manages into the *System Knowledge Base*. This way the resources are "registered" to the *Global Resource Manager*. The interaction between the applications, *Local Resource Manager*s and *Global Resource Manager* are illustrated with two example scenarios in the Figure 7 and the Figure 8. This interaction will also be described in more detail in the D4.2.

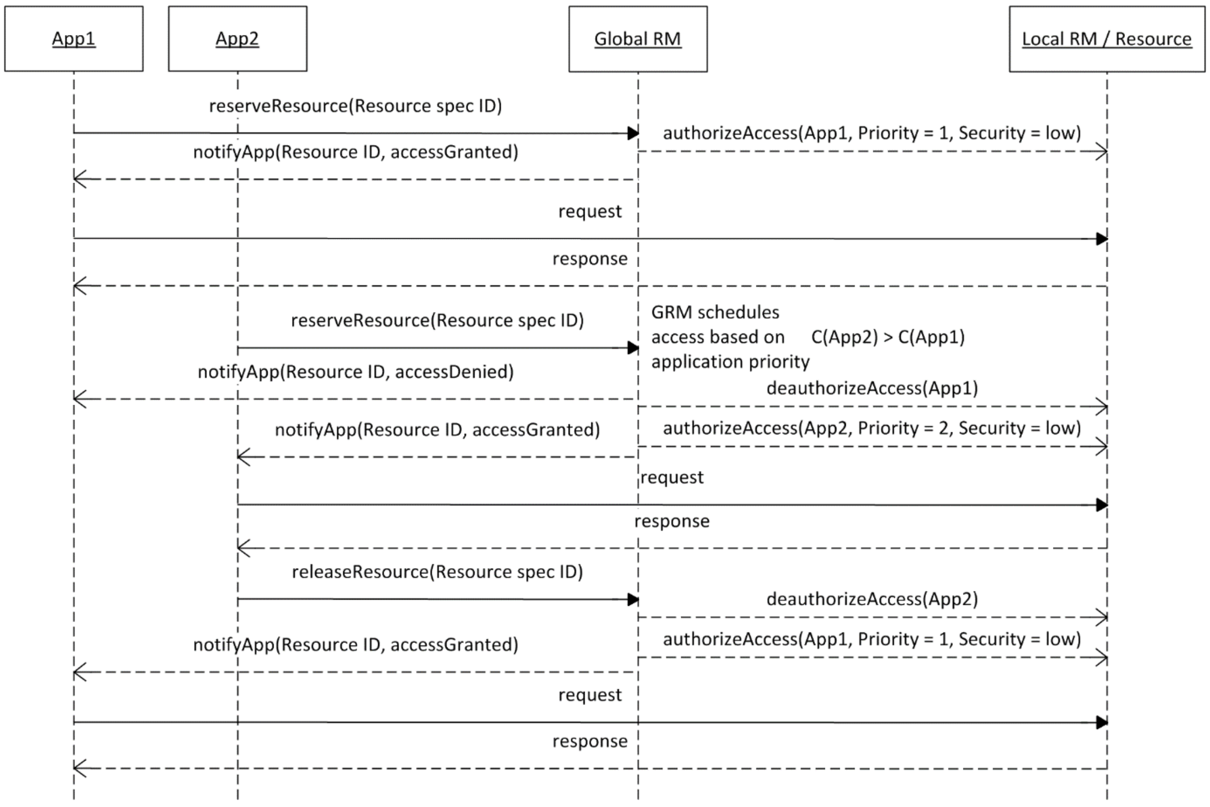

App1
App2
Global RM
Local RM / Resource
reserveResource(Resource spec ID)
authorizeAccess(App1, Priority = 1, Security = low)
notifyApp(Resource ID, accessGranted)
request
response
reserveResource(Resource spec ID)
GRM schedules access based on application priority
C(App2) > C(App1)
notifyApp(Resource ID, accessDenied)
deauthorizeAccess(App1)
notifyApp(Resource ID, accessGranted)
authorizeAccess(App2, Priority = 2, Security = low)
request
response
releaseResource(Resource spec ID)
deauthorizeAccess(App2)
notifyApp(Resource ID, accessGranted)
authorizeAccess(App1, Priority = 1, Security = low)
request
response

Figure 7. Example of resource management with exclusive resource access scheme.

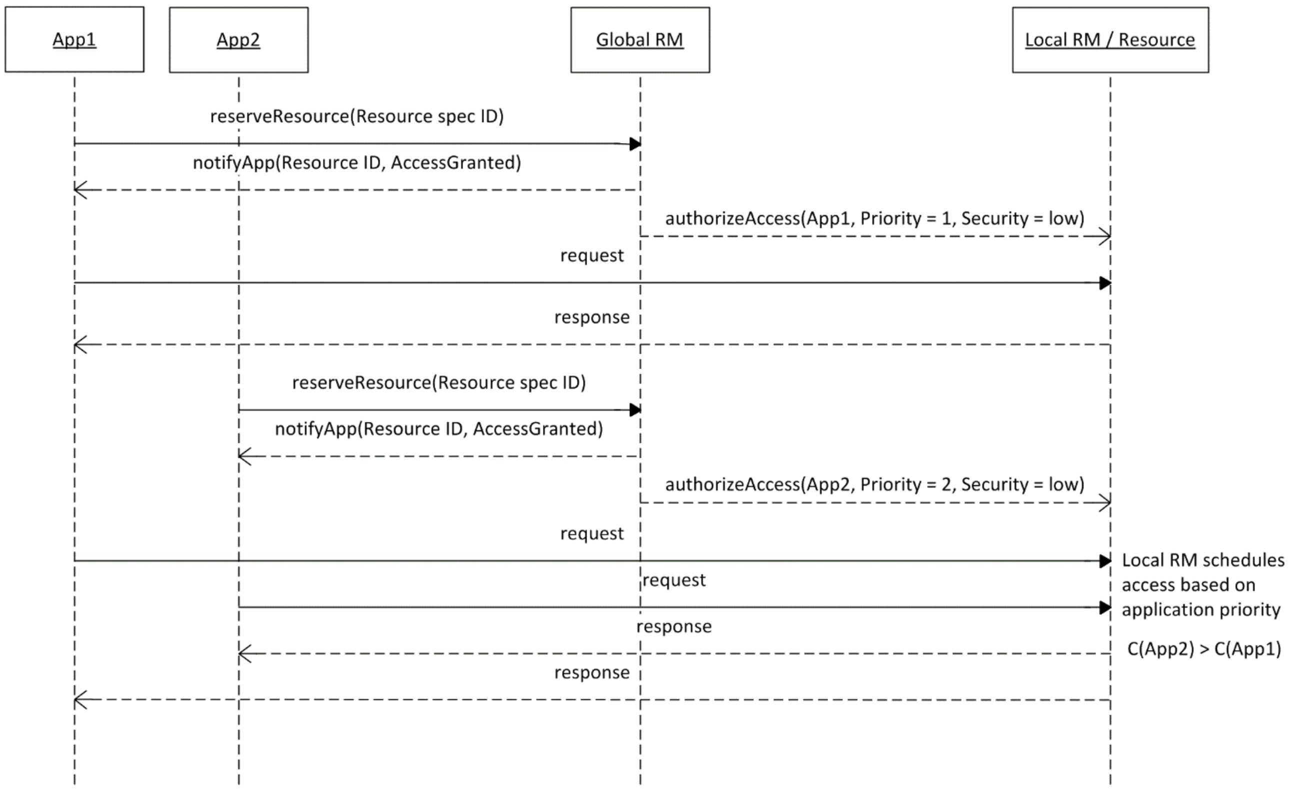

App1
App2
Global RM
Local RM / Resource
reserveResource(Resource spec ID)
notifyApp(Resource ID, AccessGranted)
authorizeAccess(App1, Priority = 1, Security = low)
request
response
reserveResource(Resource spec ID)
notifyApp(Resource ID, AccessGranted)
authorizeAccess(App2, Priority = 2, Security = low)
request
request
response
response
Local RM schedules access based on application priority
C(App2) > C(App1)

Figure 8. Example of resource management with shared resource access scheme.

5 Application Descriptions

As introduced in the beginning of this deliverable, in IMPReSS, we focus on the Internet of Things scenario in which resources consists of distributed sensor and actuator devices that can be shared by different applications. In this context, applications that access shared resources might not know in advance about the resources that would use since binding between the application and the resources could happen dynamically at runtime.

For this purpose, we use an application description to express the required resources and how critical the application is, so that the global resource manager could make a decision on which resources to be assigned to the application. This will allow distributed resources to be bound "on demand" to different applications that require them on different time frames.

The application description includes information such as:

- Criticality level of the application relative to required resources
- Functional and non-functional specification of the required resources the application needs to access.
- Required security level to be fulfilled by resources
- Resource access scheme for each resource that denotes whether resources can be shared or to be exclusively owned by an application.

5.1 Criticality level

Criticality level is used to decide which application is authorized to use the available resources for a periodic of time. The criticality level of the applications must be maintained dynamically by a centralized entity to ensure the fairness of the criticality level. In some cases, the application criticality is depending on the state of the application and it could also be relative to the required resources. For instance, the criticality of an application displaying temperature for the end-user might vary depending on the location of the temperature sensor resource (*e.g.* the temperature of the server room is more critical than the temperature of the living room).

Secondly, the need for accessing different resources to perform a task might have different priorities. For instance an application which is responsible to detect a fire and log the events in a central storage might have the highest priority to the smoke detector and heat sensors while logging the event task to a storage device could have a lower priority. However the execution of the latter task must not interfere with the first one. Therefore a separation of these two tasks must be provided by the system.

Because of these reasons, IMPReSS should allow developers to define criticality level for every resource that is going to be requested by their application as well as changing these criticality levels depending on the state of the applications. For the sake of simplicity we group the criticality level into three simple levels (low, normal, and high). When a more complex ordering is required in the future, each level can be assigned with a more detail numeric levels e.g.:

- Low 100 - 299
- Normal 300 – 599
- High 600-899
- 1000 for the owner of the resource which must be prioritized all the time

Defining the application's criticality for every single resource manually is not a scalable solution and therefore in the future work of this deliverable, developers should be allowed to express these requirements in a simple rule language pattern. However, for the sake of simplicity of our initial implementation, we will not address this requirement first.

```
<Application id="App123" desc="fire alarm">
```

```
        <State appState= "No Fire">
                <Resources>
                        <Query>
                                SELECT ?sensor ?precision ?resolution ?refObject
                                WHERE{
                                   ?sensor ssn:Observation domain:Temperature;
                                    ?sensor ssn:Precision ?precision .
                                    ?sensor ssn:Resolution ?resolution .
                                    ?sensor domain:RefObject ?refObject .
                                 }
                        </Query>
                </Resources>
        </State>
</Sensing>
```

5.2 Resource Requirement

Allowing dynamic allocation of resources to applications, the required resources to perform intended tasks must be expressed in the application description which then will be matched with the resource description in order to find the most suitable resources for the applications. This means that the language used to express the resource requirements depends on the metadata format used to describe resources.

These resource requirements could be categorized into functional and non-functional requirements. The functional requirements are related to the functions that the application need to perform while the non-functional requirements are related to the performance that resources must deliver in order to guarantee that the application could perform as intended. The functional requirements are much related to what the application is designed to do. For instance, a weather monitoring application requires different weather sensors such as thermometer, barometer, and possibly satellite imaging to predict the cloud movements. The requirements for these sensors could be expressed in an abstract way such as "a device that delivers the outside-temperature in Paris in Celsius with a precision of 0.001 degrees".

As we cannot foresee every possible function that IoT application needs to do in the future, the application description must be extendable and allows application developers to describe new kind of resource and performance requirements.

A promising solution to describe resources is by using RDF which is designed as an information model for metadata. RDF has been used in semantic web to provide a machine readable metadata of the web content. Similarly, every resource such as sensors and actuators could be described using RDF. Using RDF in this project allow us to take advantage of the components that have been well developed such as using SPARQL and the query engine to match resource description with resource requirements of the applications. Secondly, the project could re-use one of the sensor ontology approaches that are described in section 5.2.1.1. Reusing this open ontology ensure the future compatibility of IMPReSS components to external systems.

5.2.1 Functional Requirements of Applications

The IoT applications require devices that have capabilities such as sensing physical qualities of the environment or performing actions which may influence the physical quality of the environment. To enable a dynamic binding of devices, the requirement must be expressed in some degree of abstraction.

The definition of device is very broad, therefore this deliverable does not intend to restrict it. We discuss some types of devices and their possible classifications, which represent a good starting point to analyze what kind of abstraction could be used for the project. However, the implementation should not be limited to work only with devices mentioned in this section.

There exist several approaches for abstracting device capabilities for instance by introducing device classifications. As mentioned above, in general devices can be divided as sensors and actuators that are discussed more detailed in the following sections.

5.2.1.1 Sensors and their classifications

Sensors are devices capable of detecting physical qualities, such as electric current, mechanical torque, temperature, among others (Kaltenbacher 2007). "Common commercially available sensors include temperature sensors, pressure sensors, flow sensors, stress/strain sensors, accelerometers, dielectric sensors, conductivity sensors, shock sensors, and vibration sensors (Fink 2012). There are different ways of classifying them in types. For example, in (Fink 2012), they subdivide them into optical and electrical sensors, depending on whether the signal is eventually monitored in an electrical or in an optical way". However they go on to explain that there are ambiguities in this classification and that it is a difficult problem to try to strictly classify sensors in definite non overlapping types. Later on they broaden the classification to include humidity sensors, biosensors, mechanical sensors, electrochemical sensors, piezoelectric sensors, acoustic wave sensors, among others. A different type is mentioned in (Brauer 2006), where the author refers to magnetic sensors, which use magnetic fields that obtain an electrical signal in order to sense motion. Some typical magnetic sensors mentioned in this source are:

Proximity sensors to determine presence and location of conducting objects for factory automation, bomb or weapon detection, and petroleum exploration.

- Microphones that sense air motion (sound waves).
- Linear variable-differential transformers to determine object position.
- Velocity sensors for antilock(Compton, Henson et al. 2009) brakes and stability control in automobiles.
- Hall effect position or velocity sensors (Brauer 2006).

On the other hand, (Bishop 2007) separates sensors in: linear and rotational sensors, acceleration sensors, force measurement sensors, torque and power measurement, flow measurement, temperature measurement, distance measuring and proximity sensors, light detection, image and vision systems, integrated micro sensors and vision sensors. The variety of classifications available shows that there is no standard way of categorizing sensors that applies for every domain of interest. This is important to consider when designing an information model to describe sensors, because it should not restrict them to any specific categorization, given that it might not apply or be useful for all contexts.

There exists several effort to provide an ontology for classifying sensors such as the Semantic Sensor Network (SSN) Ontology, as well as others that preceded it, such as CSIRO Sensor Ontology, OntoSensor, MMI Device Ontology and CESN(Lefort, Henson et al. 2011). We provide a brief overview of these approaches in the following sections.

5.2.1.1.1 The Semantic Sensor Network Ontology (SSN Ontology)

The W3C Semantic Sensor Network Incubator Group (SSN XG) (Compton, Barnaghi et al. 2012) developed an ontology to describe sensors and sensor networks. They also studied and recommended ways to use their ontology in systems based on the Open Geospatial Consortium's (OGC) Sensor Web Enablement (SWE) standards. The SWE standards focus on Web-connected sensors and sensor systems in a framework called Sensor Web. These standards "provide description and access to data and metadata for sensors", but "they do not provide facilities for abstraction, categorization, and reasoning offered by semantic technologies".

The Semantic Sensor Network Ontology is a "formal OWL DL ontology for modeling sensor devices (and their capabilities), systems and processes" (W3C 2011). It includes the process of sensing and how sensors are deployed or attached to platforms. It describes as well systems of sensors and sensing methods. The ontology "leaves the observed domain unspecified" (Lefort, Henson et al. 2011), but when it is instantiated it allows domain semantics, units of measurement, time and time series, location ontologies and mobility ontologies to become attached to it.

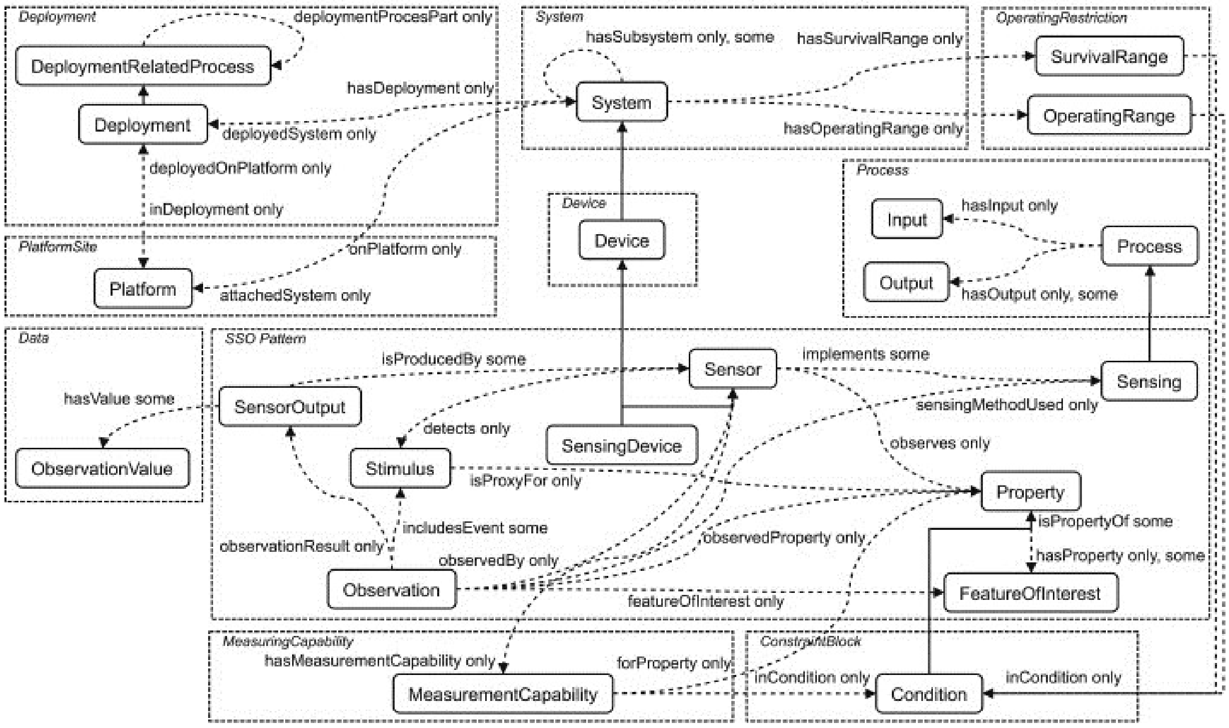


Figure 9. Overview of the Semantic Sensor Network Ontology classes and properties (W3C 2011)

Figure 9 shows an overview of this ontology, which extends further beyond the classes shown in this figure. Each dotted rectangle represents a different module: Deployment, System, Operating Restriction, Process, Device, Platform Site, Data, Skeleton, Measuring Capability and Constraint Block. Each module has properties and classes "that can be used to represent particular aspects of a sensor or its observations: for example, sensors, observations, features of interest (...), the measuring capabilities of sensors, as well as their environmental, and survival properties of sensors in particular environments" (W3C 2011).

The sensor ontology can be combined with the measurement unit ontology when the application require specific unit of measurement from a sensor. Existing approaches that already define the quantities of sensors and their units using ontologies or controlled vocabularies can be summarized as the following points:

• MyMobileWeb Measurement Units Ontology (MUO):

This ontology is divided in two blocks. The first one contains definitions of classes and properties, which "provide the essential vocabulary to define the semantics of measurements in domain ontologies"(Berrueta, Polo et al. 2008), and is further divided in three parts: units of measurements, physical qualities that can be measured, and common prefixes for units of measurements.

The second block contains several instances for the previously mentioned classes. In order to "correctly formalize the different kind of measurement units and the relationships between them", a hierarchy of measurement units was created in MUO. The classification considers whether a unit is base or derived.

• Quantities, Units, Dimensions, Values (QUDV):

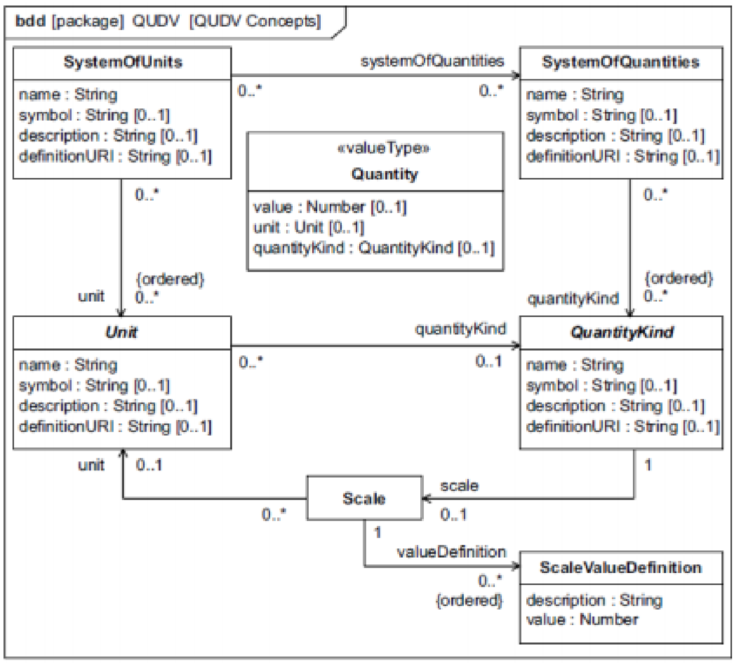


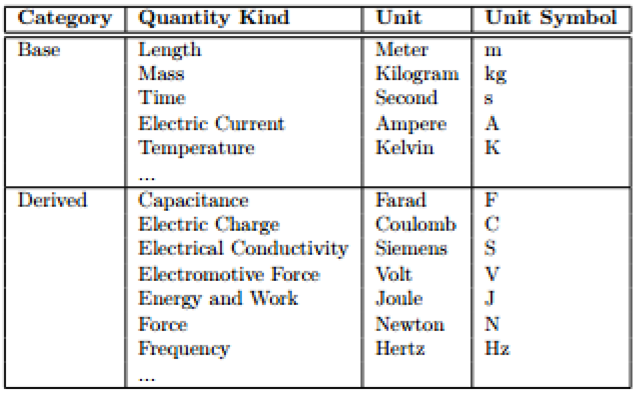
Figure 10. QUDV Concepts diagram (OMG 2009)

The Object Management Group (OMG) has developed this conceptual model of systems of units and quantities to be used in system models (OMG 2009). Figure 10 shows the concepts diagram of this model presented in SysML, an alternative to UML developed by the OMG. It contains the central concepts of System of Units, Unit, System Of Quantities, and Quantity Kind.

• Quantities, Units, Dimensions and Data Types in OWL and XML (QUDT):

The QUDT ontologies and related XML Vocabularies were being developed by TopQuadrant and NASA (Hodgson and Keller 2011), with the purpose of defining a unified model for quantities, units, dimensions and data types which were needed for NASA's now canceled Constellation Program for deep space exploration(Hodgson and Keller 2011). They collect a very large number of terms and take into consideration both base and derived quantity kinds.

QUDT's main class structure has similar concepts as QUDV, such as System Of Units, Unit, System Of Quantities, Quantity Kind and Quantity. Table 3.1 is an extract of the Quantity Kind and Unit Systems that are currently defined in the QUDT ontology.



Category	Quantity Kind	Unit	Unit Symbol
Base	Length	Meter	m
	Mass	Kilogram	kg
	Time	Second	s
	Electric Current	Ampere	A
	Temperature	Kelvin	K
	…		
Derived	Capacitance	Farad	F
	Electric Charge	Coulomb	C
	Electrical Conductivity	Siemens	S
	Electromotive Force	Volt	V
	Energy and Work	Joule	J
	Force	Newton	N
	Frequency	Hertz	Hz
	…		

Figure 11. Extract of the Quantity Kind and Unit Systems from the QUDT ontology (Hodgson and Keller 2011)

5.2.1.1.2 CSIRO Sensor Ontology

This is a generic ontology to describe and reason about sensors, deployments, observations and scientific models. "It is intended to be used for data integration, search, classification and workflows"(Compton, Henson et al. 2009) It "was designed not to be 'complete' in the sense that it should provide a language to specify sensors, but is agnostic about domain concerns"(Lefort, Henson et al. 2011).

It was considered to be a good basis for the SSN Ontology given several beneficial features. One of them was the "Plug and Play" quality, i.e. "removing the ontology from domain concerns, issues of how to represent units of measurements, locations, etc. The SSN Ontology should not include these, but rather allow any such ontology to be plugged in" (Lefort, Henson et al. 2011).

5.2.1.1.3 OntoSensor

It was created to be a general knowledge base of sensors for query and inference, which consists of a taxonomy of sensors and various properties. "The CSIRO and OntoSensor ontologies are each being able to describe most of the spectrum of sensor concepts and thus cover a wider range of concepts than the other ontologies" (Compton, Henson et al. 2009). OntoSensor contains more data and sensor types than CSIRO, but CSIRO can describe composition and structure, therefore they have differences in expressiveness.

According to the W3C, the OntoSensor ontology is incomplete and not updated since 2008. After being reviewed by the SSN XG, it was not considered to be a good basis for the SSN Ontology because its organization was not easily extendable or customizable (Lefort, Henson et al. 2011).

5.2.1.1.4 MMI Device Ontology

The Marine Metadata Interoperability Device Ontology describes oceanographic devices, sensors (devices that measure things) and samplers (devices that pick up things). It includes descriptions of measurements, systems, their components and their organization. By the time of the survey in(Compton, Henson et al. 2009), it was a work in progress and intended to expand its scope, e.g. adding concepts to describe physical properties. This ontology is also able to describe the platform to which a sensor is attached, as well as the components of that platform.

It is important for this ontology to broadly categorize the devices, which helps users to discover sensors of interest and to solve their set of use cases, some of which are similar to the application queries listed in the problem statement in section 1.2:

1. Discover and plot data from common sensor types.
2. Classify devices according to functionality.
3. Find out devices that can be deployed from given platform.
4. Find devices associated with certain real-world properties.
5. Find all sensors that perform a particular measurement.
6. Find devices that can obtain certain physical samples.
7. Find all the devices that have a certain characteristic or meet certain criteria.
8. Find device or component that can measure a given real-world property or produce a given output parameter [Int08].

According to the SSN XG, the best feature of MMI Device Ontology was that it has "system" and "capabilities" as concepts and includes hierarchies (Lefort, Henson et al. 2011).

5.2.1.1.5 The Coastal Environmental Sensor Networks (CESN)

CESN ontology describes sensor networks for coastal observing, which includes "sensor types and a DL and logic programming rules reasoner for making inferences about data and anomalies in measurements. The CESN ontology has ten concept definitions for sensor instances and six individuals" (Compton, Henson et al. 2009). It is very oriented at describing sensor types and organizing sensors in hierarchies of sensing concepts(Compton, Henson et al. 2009).

According to (Lefort, Henson et al. 2011), the CESN ontology restricts "sensor" to measure only one "physical property", it defines "instrument" as a set of "sensors", and "deployment" refers to relating "instruments" readings to "time and place of a real-world event". The SSN XG states that this is a narrow scope which is only application specific, and that "the explicit mention of sensor types will always be incomplete" (Lefort, Henson et al. 2011).

Both the SSN Ontology and CSIRO show that it is advantageous to design the ontology in such a way that it can be used in different contexts. Two examples that apply the opposite of this are OntoSensor and CESN, which offer expressive taxonomies of sensors types, but fail at being extendable or customizable. In this regard, the design of the Resource Manager’s base ontology should not restrict the application domain by providing a set of instances that most likely can never be complete and will prevent it from being easily extended.

5.2.1.2 Actuators and their classifications

Actuators translate electrical current or voltage into other forms of energy, such as force and pressure, speed and acceleration, temperature, gas composition, electromagnetic fields, light, etc(Steyaert, Van Roermund et al. 2009). The author of (Bishop 2007) presents diﬀerent types of actuators. First, there are electrical actuators, such as diodes, thyristor, bipolar transistor, triacs, diacs, power MOSFET, solid state relay, etc. Second, there are electromechanical actuators, which subdivide in direct current motor, alternate current motor and stepper motor. Third, there are electromagnetic actuators, such as solenoid-type devices, electromagnets, relay, hydraulic and pneumatic, cylinder, hydraulic motor, airmotor, valves,etc. Forth, there are smart material actuators, such as piezo electric, electrostric- tive, magnetostrictive, shape memory alloy, electrorheological fluids, ultrasonic piezo motor, etc. And finally, there are micro- and nanoactuators, such as micromotors, MEMS thin film optical switches, MEMS mirror deflectors, MEMS fluidic pumps and valves, NEMS drug dispenses, etc. According to (Brauer 2006), magnetic actuators use magnetic fields to produce motion of small or large objects using an electrical signal. Some typical magnetic actuators mentioned by this author are:

- Electrohydraulic valves in airplanes, tractors, automobiles, and other mobile or stationary equipment.
- Fuel injectors in engines of automobiles, trucks, and locomotives.
- Biomedical prosthesis devices for artificial hearts, limbs, ears, and other organs.
- Head positioners for computer disk drivers.
- Loudspeakers.
- Contactors, circuit breakers, and relays to control electric motors and other equipment.
- Switchgear and relays for electric power transmission and distribution [Bra06, pp. 3-4].

From a diﬀerent perspective, the author of (Pawlak 2006) presents two further types of actuators: linear and rotary. "Rotary actuators, also called torque motors or torque actuators, are electromechanical devices that develop torque with limited-angular travel. Linear actuators are force motors that develop force with limited linear travel". These types can be further subdivided into more specific types. Therefore, the previous consideration about not restricting the types of sensors also applies to types of actuators, because there no standard way of classifying them that is useful in every context.

5.2.2 Non-Functional Requirements of Applications

Applications are designed to fulfill certain performance parameters in order to perform the anticipated functions properly and to ensure the expected user experience. The system performance is highly depending on the combination of the hardware and software. This become more extreme on embedded systems where hardware resources is very scarce, thus the software components usually must be optimized for specific hardware to gain a maximum performance that is required for safety critical applications, such as avionic system.

In the Internet of Things context, a dynamic environment is often foreseen. Less critical applications may not be directly coupled with resources such as sensors and actuators since they may borrow resources from more critical applications when they are not utilized. Therefore the less critical applications are not able to know in advance which resources are available. IMPReSS aims to tackle this problem by linking the best matching application to the available resources to provide an optimal system performance. For this purpose, applications are required to specify resource requirements including the capabilities of devices that are expected, the performance and precession of devices or even more specific requirements such as the vendor or model number. In addition, devices are tagged with device description using the same vocabularies which allows the resource manager to match application requirements to the device description. Application requirements may include execution time constrain as depicted in the following xml:

```
<Criticality level = "high">
  <level value = "899" />
  <!-- describe more detail time constrain -->
  <ExecutionTime>
    <Computing>
      <CPUSpeed Clock="400Mhz" core="2" averageUtilization<”80%”>
      <BestCase min="" max="" />
      <WorstCase min="" max="" />
    </Computing>
  </ExecutionTime>
</Criticality>
```

A simpler approach could only express the average time required to from invoking the service until the application receives the reply without having to define the CPU requirement:

```
<Criticality level = "high">
  <level value = "899" />
  <!-- describe less detail time constrain -->
  <ExecutionTime>
      <Average min="" max="" />
      <WorstCase min="" max="" />
  </ExecutionTime>
</Criticality>
```

5.2.2.1 Network QoS

In this section we review, the relevant Network QoS approaches as consideration when designing algorithms for prioritizing services which will be discussed in the D4.3 Resource management and access scheduler.

The IoT scenarios add more complexity into the classical mixed criticality problems by introducing open-shared network medium into the system. The highly distributed nature of IoT systems and the “Best Effort” nature of the Internet’s network layer with fist come first serve scheduling leads to unpredictable response delays under heavy load periods. This may not even suitable for safety critical systems and it requires soft real-time systems to anticipate communication delays and failures.

There exist many approaches to provide a quality of service (QoS) guarantees on different layers. These approaches enable service providers to monitor bandwidth, the bandwidth availability, detect congestion, and prioritize or throttle network traffic. For the sake of simplicity of this deliverable, we will only review briefly QoS approaches that are relevant for IMPReSS.

On a packet switched network, two main techniques are available to implement QoS: Constraint-Based Routing (CBR) and Traffic Engineering (TE) (Toguyeni and Korbaa 2007). CBR considers QoS requirements such as delay, jitter, and bandwidth when making routing decisions. TE monitors and controls the flows of traffic inside the network to achieve an even network utilization. An example of TE is to utilize an admission control to grant or reject traffic flow based on the network load.

IETF proposes two different approaches. First, Integrated Service (IntServ) utilizes reservations to guarantee a congestion free network traffic. However this approach requires a huge overhead to maintain the state of network flows. Secondly, a more lightweight approach, Differentiated Service (DiffServ) classifies packet flows based on three kinds of services:

- EF (Expedited Forwarding) : premium service with reliable, low delay and low jitter delivery,
- AF (Assured Forwarding) : assured service with reliable and timely delivery,
- Default (Best-effort): it is the normal service offered by IP networks

Alternatively, many internet service providers choose to provide QoS by generously over-provisioning a network bandwidth that could handle the highest peaks based on capacity estimation. Although this method is considered simple and more cost efficient by many ISP, it does not provide a real QoS solution when the network flows grow beyond the anticipated peaks.

For a mobile ad-hoc networks there exist several approaches such as FQMM (Flexible QoS (Quality of Service) Model for MANET (Mobile Ad Hoc Network)) which combines IntServ & DiffServ. SWAN (Supporting Service Differentiation for Real-Time and Best Effort Traffic in Stateless Wireless Ad Hoc Networks) uses a rate control for UDP and TCP best-effort traffic, and sender-based admission control for UDP real-time traffic (Khoukhi and Cherkaoui 2010). Despite of these approaches, guarantying hard deadlines in wireless networks are impossible due to unpredictable wireless interference in the environment.

Another approach to provide QoS is to differentiate the access to internet servers that host web services. This approach provides better quality of service to more critical tasks on the application layer. Service Differentiating Internet Server (SDIS)[1] provides an approach to prioritize service requests by utilizing admission control for rejecting requests when the queue is almost full. It also exploits a priority scheduling algorithm to serve the highest priority tasks first.

As an initial approach, IMPReSS focuses on providing end-to-end network related QoS parameters including delay, jitter, available bandwidth, and packet loss which are acceptable for the applications. Alternatively, the application could define simply the preferred end-to-end delay. Expressing these requirements, the applications need to define the QoS parameters with minimum and maximum values that are acceptable to execute the requests. In order to guarantee these requirements, the resource manager will monitor these quality parameters and use this information to assign the suitable resources to applications. An example how the network requirements could be expressed in the application description:

```
<Network end-to-end-delay preferred = "" max = "" / >

<Network>
      <Latency preferred = "" max = "" />
      <Jitter preferred = "" max = "" />
      <Throughput preferred = "" min = "" />
      <PacketLoss preferred = "" max = "" />
</Network>
```

5.2.2.2 Device time to failure requirement

Since the IoT scenarios often involve battery powered sensor nodes which have a limited lifetime, the applications might be interested in using devices which will be active for a predictable time frame. Enabling this feature, the application may describe their expectation of the device lifetime. For instance an application may define that it requires a temperature sensor installed at a location that will last for a year.

```
<TimeToFailure min = "100" max = "365" unit = "days" />
```

[1] http://spirit.cs.ucdavis.edu/pubs/journal/sdis.pdf

5.2.2.3 Trust and Security requirements

In IoT trust and security requirements have many confluences and similarities with performance and reliability requirements. Section 3 presented security objectives in three main categories: *confidentiality, integrity, and availability*. Availability of the resource can be affected by internal causes in the system such as poor reliability of network or external causes such as denial of service attacks. The former cause relates more to reliability requirement and the latter to security requirement, but still both causes affects to same objective, availability.

The minimum security level needed in the resources such as sensors and actuators depends on the phenomenon they are measuring or what they are controlling. If sensor measurement is considered as confident information, application using the information is not allowed to change the level of the security below the minimum level required. Therefore in the cases where multiple applications are using the same resource, none of these applications are allowed to use lighter security mechanisms than the minimum level required by resources.

Since IMPReSS aims at providing a development tools for novice developers which do not have an extensive knowledge of security, we will provide developers with a simple abstraction of the security levels that could be understood by non-expert developers such as "No Security", "Low", "Medium", "High".

These levels will be mapped onto different specific encryption algorithms and trust frameworks. Application using IMPReSS platform needs to support the same security mechanism than the resources are using. Therefore supported security mechanisms need to be expressed both in the application description and resource description in order to bind appropriate resource with application.

From the application point of view, most important categories for the security and trust are integrity and availability. For example factory automation has strong requirements for information integrity and availability. Therefore when designing the application, required integrity level for the information needs to be based on the severity of the consequences. Application description will express the required integrity level of the resource. Levels need be easy to understand such as high, medium and low and then mapped to real integrity verification solutions such as SHA and MD5.

Some of the resources are necessary for the application in order to work properly and some are useful but not totally necessary. Therefore availability level needs to be expressed using the similar levels that integrity levels.

Security related definitions in the application description can be utilized in two different cases. First, the most appropriate resource for application can be selected partly based on the security definitions. For example, applications requiring high integrity level for temperature measurement will only be offered the resources supporting e.g. SHA1 verification or better. Secondly, security level of the resource can be adjusted on run-time based on the current risk level. Trust and security requirements may influence the performance of the system and increase end-to-end delays. For instance, there are many trust frameworks that require an exchange of certificates and validation from an independent authority before data transmission is started. Therefore it is useful to use as low security level as possible. For example, if the given resource is used in secure location, security level can be adjusted to minimum level.

There are several security ontologies available for describing security both for resources and applications. Some of the security related ontologies are aimed to use when designing the applications and some ontologies can be used for resource and service discovery. Following sections present some of the current ontologies related to security.

5.2.2.3.1 NRL Security Ontology

NRL Security ontology (A.Kim, J.Luo, et al. 2005) focuses on annotation of functional aspects of resources using OWL. This ontology is capable of representing security statements like mechanisms, protocols, algorithms and credentials can be represented and it can be utilized to any electronic resource. NRL Security ontology is composed of seven sub-ontologies, which create the overall ontology. According to (A.Kim, J.Luo, M. Kang, 2005) the subontologies of NRL Security ontology are following:

1. Main Security ontology: an ontology to describe security concepts
2. Credentials ontology: an ontology to specify authentication credentials
3. Security Algorithms ontology: an ontology to describe various security algorithms
4. Security Assurance ontology: an ontology to specify different assurance standards
5. Service Security ontology: an ontology to facilitate security annotation of semantic Web services
6. Agent Security ontology: an ontology to enable querying of security information
7. Information Object ontology: an ontology to describe security of input and output parameters of Web services

5.2.2.3.2 SOA security ontology

SOA security ontology (P.Savolainen, et al. 2007) introduces the taxonomy of information security, focusing on service oriented architectures. Service taxonomy presented in this paper embodies five different aspects: *Security assets, security attributes, security threats, security solutions and security metrics*.

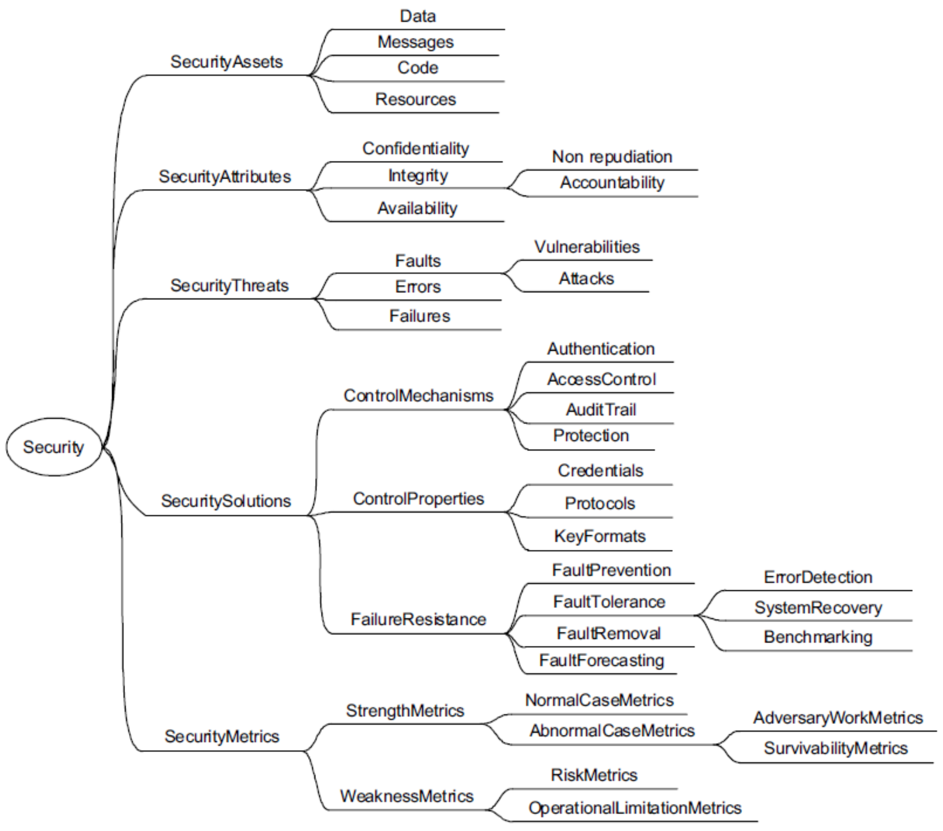


Figure 12. SOA security ontology

5.2.2.4 Quality of device capabilities

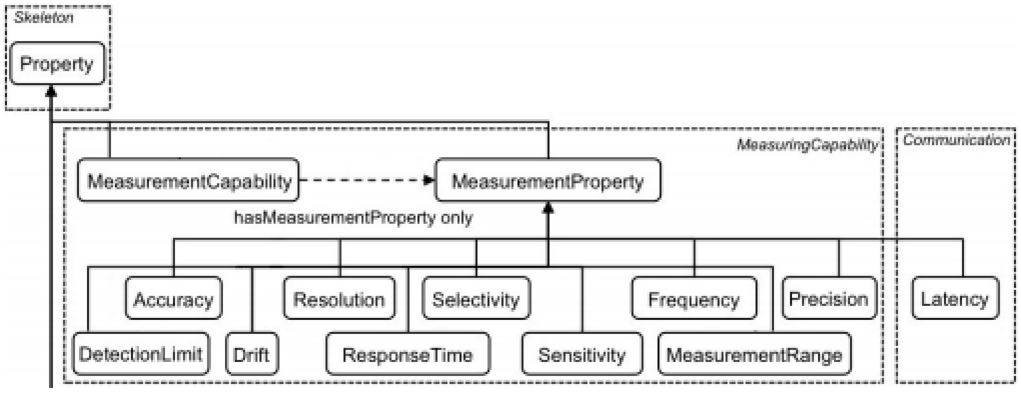


Figure 13. Enumeration of Measurement Properties in the SSN Ontology (W3C 2011)

Each device may have quality that can be associated with its capability. For instance, a camera sensor has resolution, color depth, and image sensor size, which is relevant to determine the quality of the picture that the camera could produce. These quality parameters vary greatly from device to device and therefore must be defined by the device manufacturer.

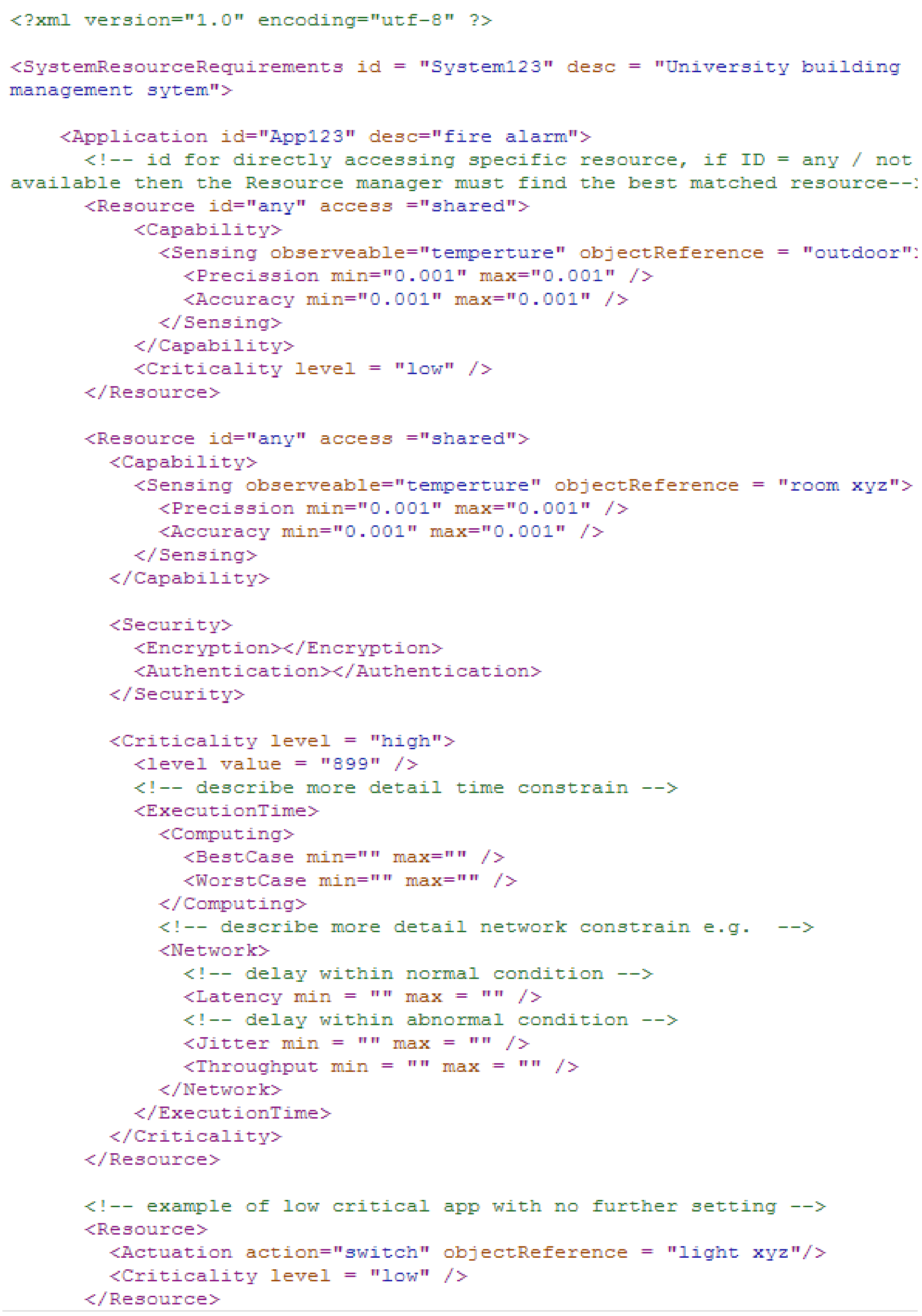
However there are some qualities that are common for different type of devices such as for sensors sampling rate, sensing distance, operating temperature, etc. There exist also similar qualities with different unit of measurements such as precision and accuracy.

Figure 13 zooms in the SSN ontology. On some of these modules and shows that Measurement Capabilities are related to a collection of Measurement Properties of a sensor in specific conditions. All the possible Measurement Properties of a sensor are shown represented as subclasses, some of which are: drift, sensitivity, selectivity, accuracy, precision and latency. There are further relationships in the SSN Ontology between Measurement Capability, Measurement Property and other predicates like "forProperty" and "inCondition" (W3C 2011).

We define an abstraction of the functional and non-functional requirements into the following attributes. For every required resources, application developers must define the required capability for instance a sensor has a "sensing" capability that can be further elaborated by using different attributes such as what kind of physical world events (observable) that the sensor is able to sense, to which object the sensor is measuring, what kind of precision, accuracy, and range. An example how the requirements could be expressed as the following:

```
<Capability>
  <Sensing observable="temperature" unit="Celcius" objectReference = "room xyz">
    <Precision min="0.01" max="0.09" />
    <Accuracy min="0.01" max="0.09" />
    <Range min="0.01" max="0.09" />
  </Sensing>
</Capability>
```

5.3 Serialization and Format of the application descriptions



```
<?xml version="1.0" encoding="utf-8" ?>

<SystemResourceRequirements id = "System123" desc = "University building
management sytem">

    <Application id="App123" desc="fire alarm">
      <!-- id for directly accessing specific resource, if ID = any / not
available then the Resource manager must find the best matched resource--
      <Resource id="any" access ="shared">
          <Capability>
            <Sensing observeable="temperture" objectReference = "outdoor"
              <Precission min="0.001" max="0.001" />
              <Accuracy min="0.001" max="0.001" />
            </Sensing>
          </Capability>
          <Criticality level = "low" />
      </Resource>

      <Resource id="any" access ="shared">
        <Capability>
          <Sensing observeable="temperture" objectReference = "room xyz">
            <Precission min="0.001" max="0.001" />
            <Accuracy min="0.001" max="0.001" />
          </Sensing>
        </Capability>

        <Security>
          <Encryption></Encryption>
          <Authentication></Authentication>
        </Security>

        <Criticality level = "high">
          <level value = "899" />
          <!-- describe more detail time constrain -->
          <ExecutionTime>
            <Computing>
              <BestCase min="" max="" />
              <WorstCase min="" max="" />
            </Computing>
            <!-- describe more detail network constrain e.g.  -->
            <Network>
              <!-- delay within normal condition -->
              <Latency min = "" max = "" />
              <!-- delay within abnormal condition -->
              <Jitter min = "" max = "" />
              <Throughput min = "" max = "" />
            </Network>
          </ExecutionTime>
        </Criticality>
      </Resource>

      <!-- example of low critical app with no further setting -->
      <Resource>
        <Actuation action="switch" objectReference = "light xyz"/>
        <Criticality level = "low" />
      </Resource>
```

Figure 14. Application description serialized in XML.

So far for this deliverable we have presented an example of parameters that could be requested from the application developers' perspective. The format and serialization chosen for this deliverable is in XML for the sake of readability. However, we foresee that IMPReSS would offer different type of serialization that the developers could choose from. For instance, the serialization of the application and device description could be done depending on the available computing power to parse and process the data format. When limited computing power is not a primary concern but the readability of the descriptions is more important, the developers may serialize the descriptions in XML or JSON format. However if the devices and applications have a very limited computing power, the application and device description should be serialized more efficiently such as using a binary format.

Since we are planning to use RDF to maintain the actual state of the resources, the application requirements could be translated into SPARQL which could be directly use by the resource manager to find the matching requirements and available resources. This will simplify the implementation of the resource manager, however during our requirement engineering we found that the majority of developers are not familiar with semantic web technology including SPARQL. Therefore if we decide to use SPARQL for expressing the application requirement, we need to provide a user friendly interface able to generate SPARQL. However the final decision will be made during the implementation phase.

6 Summary and Conclusions

In this deliverable we presented the overview of current approaches for handling MC systems and the envisioned system to enable MC in the Internet of Things scenario. We elaborated the additional problems that IoT introduces into the classical MC problems such as the current internet infrastructure based on best effort assumption. This makes IoT systems cannot guarantee the hard deadlines as required by safety critical systems. However mixed criticality is still a valid problem for IoT system that shares distributed resources to reduce the overall cost of the system.

IMPReSS aims at adopting mixed criticality approaches for IoT to provide a best effort solution in maximizing the system’s performance while lowering the overall system cost. To achieve this vision we reviewed similar approaches that have been investigated for providing internet QoS. Many of these approaches differentiate services on different network layers based on certain parameters such as the content of the traffic, the consumer, or by utilizing a reservation system.

We propose to manage the distributed resources such as sensors and actuators from a centralized resource manager supported by local resource manager. The centralized resource manager controls the resource access in a global level aiming both at optimal use of resources in the system and solving conflicts between mixed criticality applications by using prioritization algorithms that will be define in the “D4.3 Resource management and access scheduler”. A central component in the global level resource management is a knowledge base that contains representations of resources including functional capabilities and non-functional properties such as CPU and memory utilization and end-to-end network delay. This knowledge base is used by the global level resource manager to selects the most suitable resource for each application.

The local resource manager monitors the local resource utilization and reports this to the global resource manager. Moreover, the local resource manager administers the access to the resource in a way that only applications that have obtained a reservation from the global resource manager could access it. The local resource manager will need also to schedule the access to the resource based on the application criticality level.

Enabling the matching between application and requirements, the resources must have a metadata description containing information such as device capabilities, quality parameters such as device precision, accuracy, network end-to-end delay, etc. We discussed the current approaches proposed as ontology that can be extended for this project. Currently we see SSN ontology as a promising solution since it provides a basic schema that we can extend to describe actuators and domain specific quality parameters. Moreover, we have discussed different relevant parameters which were elicited during requirement workshop. These parameters will be used as starting point that can be extended or slimed down depending on the developer’s need when building IoT systems using IMPReSS.

We also discussed the possibility of having different serialization format for the resource and application description depending on whether readability or system performance is the focus of the development. In this deliverable we presented an example of application description in XML format for the sake of readability. The XML description could be translated into SPARQL query by the resource manager for querying the most suitable devices for the applications. Another solution that we would like to investigate is providing users with a simple form-based user interface that generates SPARQL query to keep a clear separation of concern between the description language processing and the resource management.

In conclusion, we believe that the quality parameters used to differentiate services depends on the application domain therefore IMPReSS should not try to provide every possible parameters. Thus, in this deliverable we provide several quality parameters which are mostly used cross domains such as end-to-end delay, priority. IMPReSS should also provide a tool that allows developers to extend these parameters without having the knowledge about RDF and SPARQL.

7 Bibliography

Baruah, S., et al. (2010). Towards the design of certifiable mixed-criticality systems. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, IEEE.

Berrueta, D., et al. (2008). "Measurement Units Ontology." URL: http://idi. fundacionctic. org/muo/muo-vocab. html.

Bishop, R. H. (2007). Mechatronic systems, sensors, and actuators: fundamentals and modeling, CRC press.

Brauer, J. R. (2006). Magnetic actuators and sensors, John Wiley & Sons.

Compton, M., et al. (2012). "The SSN ontology of the W3C semantic sensor network incubator group." Web Semantics: Science, Services and Agents on the World Wide Web **17**: 25-32.

Compton, M., et al. (2009). A Survey of the Semantic Specification of Sensors. SSN.

Fink, J. K. (2012). Polymeric Sensors and Actuators.

Hodgson, R. and P. J. Keller (2011). "QUDT-quantities, units, dimensions and data types in OWL and XML." Online (September 2011) http://www. qudt. org.

Kaltenbacher, M. (2007). Numerical simulation of mechatronic sensors and actuators, Springer.

Khoukhi, L. and S. Cherkaoui (2010). "Intelligent QoS management for multimedia services support in wireless mobile ad hoc networks." Computer Networks **54**(10): 1692-1706.

Lefort, L., et al. (2011). "Semantic sensor network xg final report." W3C Incubator Group Report **28**.

OMG (2009). "Quantities, Units, Dimensions, Values (QUDV)." Retrieved january 24, 2014, from http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-qudv:quantities_units_dimensions_values_qudv.

Pawlak, A. M. (2006). Sensors and Actuators in Mechatronics: Design and Applications, Taylor & Francis.

Steyaert, M., et al. (2009). Analog circuit design: high-speed clock and data recovery, high-performance amplifiers, power management, Springer.

Toguyeni, A. and O. Korbaa (2007). Quality of service of internet service provider networks: State of the art and new trends. ICTON Mediterranean Winter Conference, 2007. ICTON-MW 2007.

W3C (2011). Report work on the ssn ontology, W3C.