Architecture for mixed criticality resource management in Internet of Things

Janne Takalo-Mattila, Jussi Kiljander
VTT Technical Research Centre of Finland
Oulu, Finland
email:firstname.lastname@vtt.fi

Ferry Pramudianto
Fraunhofer Institute for Applied Information Technology
FIT
Sankt Augustin, Germany
email:ferry.pramudianto@fit.fraunhofer.de

Enrico Ferrera
Istituto Superiore Mario Boella
Torino, Italy
email: ferrera@ismb.it

Abstract— **We believe that the next big step in the field of Internet of Things (IoT) is to realize a virtual computing platform that provides access to heterogeneous group of device resources present in our living environments. By enabling 3rd party developers to access sensor and actuator resources present in a given environment in a same way they can access resources of a single mobile phone, the virtual computing platform would open a new market for the 3rd party IoT applications like the smart phones have done for mobile apps. To accomplish this vision, the virtual computing platform must be able to manage resource sharing between applications with differing criticality requirements for ensuring that the whole IoT system runs optimally. The main challenge is that the approach should be generic and extendable for future needs. To tackle this issue, we propose a two-level resource management architecture, where the necessary information about applications and resources are represented with machine-interpretable semantic descriptions based on the Semantic Web technologies. At the system level, these descriptions are used by the global resource manager for allocating resources to the applications based on their criticality and needs. At local level, each device is assigned with a local resource manager that schedules the access to resources provided by the device so that the performance of the more critical applications could be optimized at the expense of the less critical ones. To evaluate our approach in practice, we have implemented a reference implementation of the proposed architecture and demonstrated it through several applications with differing criticality levels. The results are very promising for managing mixed criticality applications in IoT.**

I. INTRODUCTION

Smart phones and related app stores have revolutionized the information and communication technology (ICT) market and changed the way people use mobile phones today. We believe that a similar revolution is possible in Internet of Things (IoT) and pervasive computing domains if we provide the application developers with means to create application to our everyday living environments. To accomplish this, there is a need for abstracting IoT as a virtual computing platform that enables developers to access the functionality and information provided by heterogeneous devices that inhabit our environments in a uniform way, analogous to the operating systems in mobile phones.

Many kinds of technologies and solutions are needed in order to make the virtual computing platform reality. The focus in the research and development towards IoT and pervasive computing middleware has been on the interoperability and discovery technologies. The role of the interoperability technologies is to abstract the heterogeneity of devices and provide uniform interface to the data and capabilities of devices. These interoperability technologies and solutions include Universal Plug and Play (UPnP) [1], Constrained Application Protocol (CoAP) [2], Network on Terminal Architecture (NoTA) [3], Message Queue Telemetry Transport (MQTT) [4] and Simple Object Access Protocol (SOAP) [5], to name a few. The discovery technologies in turn provide means to locate devices and information within the IoT system. Existing solutions for discovery include, for example, uID architecture [6] and EPCglobal [7]. In addition to these technology specific solutions, there are also numerous middleware platforms and frameworks that incorporate discovery and interoperability technologies to facilitate the development of IoT systems. These platforms and frameworks include HYDRA [8], AMIGO [9], Apple HomeKit [10], COBRA [11] and SPITFIRE [12], just to name a few.

In addition to the interoperability and discovery mechanism, an important functionality of the virtual computing platform is to manage how applications access and shares device resources provided by the IoT system. Without this type of functionality, the IoT systems would be unpredictable and far from optimal as 3rd party applications would interfere with each other in many ways. Systems that can manage and execute several applications while still guaranteeing their differing criticality requirements (*e.g.* real-time, performance, security, safety, *etc.*) are called mixed-criticality (MC)

ISBN 978-4-89362-316-4 (C)2014 T-Engine Forum

systems in the literature [13]. MC is traditionally applied within a single device, and the application domains are usually related to vehicles such as cars and airplanes (see AUTOSAR[1] and ARINC[2] for further information). Because of this, the research and development within MC systems has mainly focused on two aspects. The first aspect is approaches for scheduling processor time within a single or multicore platform. Examples of this type of approaches are presented in the papers [14][15]. The second aspect is approaches that provide protection for safety critical processes by system partitioning. This type of approach is presented in the paper [16]. In modern cars, both safety-critical and non-safety-critical applications coexist on the same platform and even share the same hardware resources. For instance, the same wheel sensor may be shared by the traction control system and a non-critical navigation application [17].

Several important differences exist between the IoT domain and classical MC systems, which must be considered when utilizing the MC idea in the IoT. First, instead of computing platform resources such as processor and memory, the applications in IoT domain share data and actuation capabilities provided by different kind of devices. Second, the resources and applications in traditional MC systems (*e.g.* car, airplane, etc.) are typically integrated by a single manufacturer who is able to define the application criticalities from a holistic perspective. In IoT, the resources are heterogeneous. The third (and most notable) difference is that, in a traditional MC system, the resources and applications to be deployed into the system are known at design time. In contrast, IoT systems are constantly evolving (*i.e.*, the system can be upgraded with new devices and third party applications at runtime). The fourth difference is that, IoT systems are built on top of networks relying on a best-effort delivery that makes the resources of the system occasionally unavailable for applications. Because of these characteristics, the approach for the MC resource management in IoT must be flexible for the changes in the system, and extendible for the future needs.

In this paper, we propose a novel approach for MC management in IoT domain. The central idea in the approach is that, resources and applications are represented with Semantic Web knowledge representation technologies. The advantages of this design choice are twofold. First, it makes the approach flexible and easily extendable for the future needs. Second, it enables application developers to define their resource specifications at a high-abstraction level making their applications more easily portable to different systems. The proposed resource management architecture consists of two levels. At the system level, the resource management focuses on controlling which resources are allocated for which applications. At local level, the role of the resource manager is both to govern that only the authorized applications access a resource, and to schedule the requests performed by applications. In both levels, the decisions made by the resource manager are based on the criticality and needs of the applications. To evaluate the approach in practice, we have implemented a reference implementation of the architecture, as well as, a prototype of a mixed-critical IoT system in a smart home environment. To summarize, the main scientific contributions of the paper are following.

1) An approach for mixed criticality resource management in IoT.
2) The reference implementation of the architectural components.
3) Prototype IoT system in smart home domain.

II. System Model

The main goal of our approach is to create a generic and extendable resource management system for IoT environments. The proposed resource management system optimizes the function of the whole IoT system by managing resource sharing between applications with different levels of criticality. The system model for the resource management approach (presented in the fig. 1) consists of the following entities: Applications, Application Descriptions, Resources, Resource Descriptions and Resource Management components.

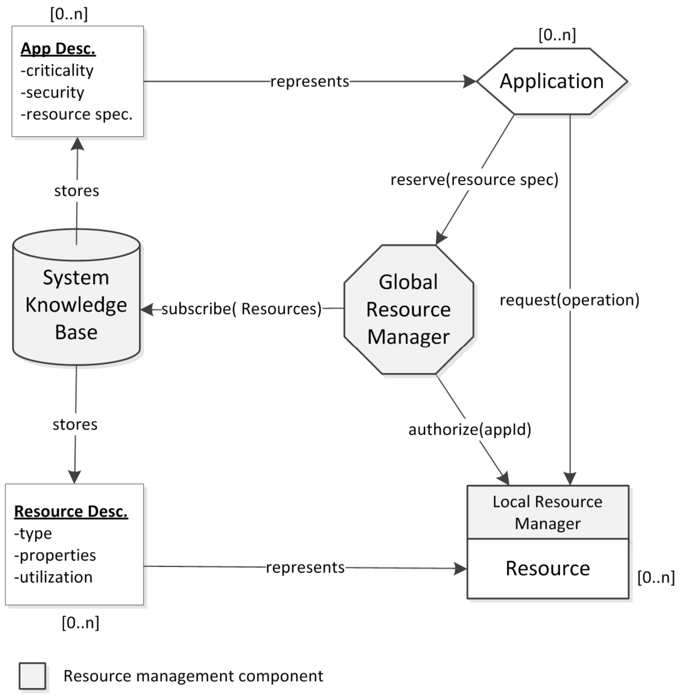


Figure 1. System model for resource management in IoT domain

Applications are software processes that provide different kinds of services for the end-users (e.g. turning on the lights in a room when the user enters the room). To do this, the application needs to access the resources provided by devices (*e.g.* motion detector and lights). These resources can be divided into two categories based on their role in the system:

[1] http://www.autosar.org/
[2] http://www.arinc.com/

- Sensors: peripheral devices that monitor the status of the environment and provide an interface for accessing this information.
- Actuators: peripheral devices that provide an interface for the applications to modify certain aspects of the physical world.

Resource and application descriptions represent the information about resources and applications that is relevant for the resource management. In order to make the approach extendable for future needs and make the applications as generic[3] as possible, Semantic Web technologies are utilized for representing the resource and application descriptions. Each resource description contains functional and non-functional properties of the resource. The attributes and classes used in the resource description depend on the domain specific ontology used for presenting the resource. The resource management approach is agnostic to the used ontology. However, the application developer must utilize the same ontology in the resource specifications.

Application descriptions consist of two main parameters: criticality of the applications and resource specifications. Application criticality defines how essential the application is for the IoT system. It can be defined either at the application design time by the developer or at run-time by the system administrator. Resource specifications describe what kinds of resources are needed by the application. The resource specifications are represented by *SPARQL* [18] SELECT queries, and typically contain both functional and non-functional specifications for the resources. For each resource specification, the application description also defines the access scheme (either shared or exclusive) and how critical the given resource is for the application.

The Resource Manager is a distributed entity consisting of a System Knowledge Base (SKB), a Global Resource Manager (GRM) and an *n* number of Local Resource Manager (LRM) components (where *n* is the number of resources in the IoT system). The SKB stores the state of the IoT system and provides other components with means to monitor and update the state. The IoT system state represents the applications and resources deployed into the system, as well as, which resources are used by which applications in a particular point of time. Two design choices exist for the SKB that make the approach for resource management flexible and extendable. First, it utilizes a publish-subscribe interface, where the publishers are loosely coupled to the subscribers. This makes it possible to add new components (*e.g.* for monitoring the resources and applications at runtime) into the system without modifying the existing components. Second, a Resource Description Framework (RDF) data model is used for representing information related to the applications and resources. In the context of IoT, it is extremely difficult, even impossible to predict all possible applications, resources and devices that may be deployed into the system. Therefore, the flexibility and extendibility that RDF offers is a great benefit for the resource manager. The interface provided by the SKB is based on SPARQL 1.1 and presented in the table I.

TABLE I. OPERATIONS SUPPORTED BY THE SKB

Operation	Description
Update	Provides means to modify the system status. The exact operation to be performed is presented in SPARQL 1.1 Update syntax.
Query	Provides means to query the status of the system using SPARQL 1.1 SELECT queries.
Subscribe	Provides means to monitor the system status by performing persistent SPARQL 1.1 SELECT queries. The SKB will notify the client in the case the results of the operation change.
Unsubscribe	Terminates the subscription operation.

The GRM manages resources on the IoT system level and aims at optimizing the functionality of the whole IoT system. From the individual application point of view, it would be optimal to have access to any resource all the time. However, this could lead to a suboptimal behavior of the whole system, because applications that are more critical could be left out without the needed resources. Therefore, GRM optimizes the IoT system behavior in the expense of the less critical applications. The resource management operations provided by the GRM are listed in the table II.

TABLE II. OPERATIONS SUPPORTED BY THE GRM

Operation	Description
Register application	Provides means to registers an application. The parameters include application ID and the application description.
Reserve resource	Provides means to make a persistent reservation to a resource that matches the specification (defined by resource specification ID). The GRM responds with a resource ID that matches the specification. The GRM will notify the client whenever a different resource is allocated for it.
Release resource	Provides means to release a resource for other applications. Resource specification ID specifies the resource to be released.
Unregister application	Provides means to uninstall an application. The application ID parameter specifies the application to be unregistered.

In practice, the resource management at the system level works as follows: When a new application is deployed into the

[3] The idea is that the same application can be deployed to different IoT systems with little effort.

system, the application makes *register application* –request to the GRM. Once the application is registered, GRM starts searching for suitable resources for the application by subscribing to the resource descriptions stored into the SKB that match the resource specifications presented in the application description. This way the GRM is constantly aware of the most suitable resources for each application. Whenever an application needs to access a resource, it sends the *reserve resource* –message to the GRM. After receiving this message, GRM responds with the identifier of the resource that is most suitable for the application. It should be noted that since the GRM is all the time aware about the most suitable resources for each applications it is able to quickly respond to the reservations made by the applications (*i.e.*, no match making is needed at this phase). The *reserve resource* requests are persistent, and the application needs to release the resource in the case it is not needed anymore. The GRM will also notify applications, for example, if a resource that is more suitable is found or if a resource used by application has been assigned to a more critical application.

At local level, the resource management is performed by the LRM. The local level typically means within a single device. However, in some cases the local level can also mean a whole sensor network if it is modeled as a single resource. The idea in the local level resource management is that each resource is equipped with an LRM that authorizes the access to the resource and schedules the request made by the applications. The authorization of applications works so that when the GRM authorizes an application to utilize a certain resource, it will inform the LRM about the authorization. The LRM is then responsible for controlling that only the authorized application(s) can access its resources. The scheduling performed by the LRM is needed only if multiple applications are authorized to utilize the same resource at the same time. Basic principle in the scheduling is that the more critical applications are served first. In addition, to the authorization and request scheduling, the LRM provides interface for obtaining the resource description that is published into the SKB when the resource is deployed into the system. Operations supported by LRM are listed in table III.

TABLE III. OPERATIONS SUPPORTED BY THE LRM

Operation	Description
Authorize access	Informs the LRM about an application that is allowed to access the resource. The parameters are the application ID and the criticality level of the application.
Request resource	Applications use this operation to request resource specific services. The parameters of the operation include the application ID and the operation to be executed.
Deauthorize access	Informs the LRM that the application cannot access the resource anymore.
Get resource description	Used to obtain the resource description.

III. REFERENCE IMPLEMENTATION

In order to study the feasibility of our approach, we have developed a reference platform that follows the basic architecture presented in the previous section.

A. System Knowledge Base

The SKB reference implementation is based on Red-SIB (Semantic Information Broker), which is available as open-source software. Red-SIB [19] is an RDF-database that provides publish-subscribe mechanism for accessing RDF data. The communication between clients and the Red-SIB is based on an XML-based protocol called Smart Space Access Protocol (SSAP) [20]. In our reference implementation, SSAP protocol is directly used for communication between the SKB and the GRM. In the GRM side, we have used a Python-library called m3_kp_api to implement communication with the GRM.

We have extended the Red-SIB with a Websocket-interface [21] to provide a standard interface for applications to access the SKB services. Websocket provides a full-duplex communication channel over single TCP connection that allows implementing the subscription mechanisms. One of the main advantages of Websocket is that the protocol is supported by current web browsers. Because of this it is easy to develop browser applications that can monitor the information available inside the SKB.

The Websocket interface is implemented with the Python Autobahn websocket library. One of the core components of the Autobahn-framework is an event-driven networking engine, called Twisted. Event-driven programming paradigm is thus used in the interface design. The request messages are encoded using following JSON-based protocol:

```
[operationType, transactionID,
      sparql1.1Message]
```

The current version of the websocket interface enables applications to query information (operation type 0), subscribe to the information (operation type 1) and update information (operation type 2). The transaction ID (presented with Universally Unique Identifier (UUID)) identifies each request. SPARQL 1.1 message is the actual payload in the protocol that is forwarded to the Red-SIB RDF database. The Red-SIB will create events to the Twisted engine whenever the results of the subscription operation change. The results are encapsulated in SPARQL JSON format [22] and send to the client.

B. Global Resource Manager

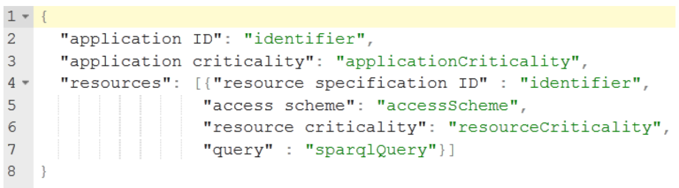
The GRM is implemented with Python and consist of two modules: Manager and Communication Handler. The Manager module is responsible for performing the resource management activities. Whenever a new application is registered, the Manager subscribes to the resource specifications and creates a thread for each resource specification to handle to notifications send by the SKB. Each resource that matches the specification is added to the list of suitable resources. If the resource is new (i.e., no other application is interested in the resources) the GRM also creates a thread for monitoring the status (e.g. utilization rate) of the resource. When applications make a reservation to a resource that matches a certain resource specification, the GRM first searches suitable resource from free resources (*i.e.*, resource that is not used by other application in exclusive mode). If free resources are not available, a resource used by least critical application is selected. If the selected resource is used by other application in exclusive mode, the application will be notified and new resource (if possible) will be assigned to the application. The applications will be also notified (and new resource is assigned) in the case a resource disappears or resource utilization rate becomes too high (90 % utilization is the threshold in the current implementation).

The Communication Handler module provides applications (and other clients) with interface to access the GRM services. Similarly to the SKB implementation, the communication handler module in GRM is based on the Websocket protocol, and it's Autobahn-implementation. The Communication Handler reacts to two types of events. First, messages sent by the application need to be forwarded to the Manager module. Second, the clients need to be notified about the events created by the Manager-module whenever status of the reservations changes.

Using the Websocket protocol, we can provide full-duplex communication channel between the applications and the GRM. Following JSON-based message format is used:

```
[operationType, messageType,
        {dataObject}]
```

The *OperationType* defines the type of the operation (i.e., reserve resource, release resource, register application, unregister application). The *messageType* defines the type of the message (*i.e.*, request, response or notification). The content of the dataObject field depends on the operation type. In *reserve resource* and *release resource* operations the dataObject specifies the *resource specification ID* to be reserved/released. In unregister application, the dataObject field contains the *application ID* of the application to be unregistered. In *register application* operation, the dataObject field is more complex and contains the following JSON object with multiple key-value pairs:



```
{
  "application ID": "identifier",
  "application criticality": "applicationCriticality",
  "resources": [{"resource specification ID" : "identifier",
                 "access scheme": "accessScheme",
                 "resource criticality": "resourceCriticality",
                 "query" : "sparqlQuery"}]
}
```

Figure 2. DataObject field for register application request.

The application ID and the criticality of the application are defined in the first two parameters. The third parameter consists of a list of resource specifications. Each resource specification starts with the ID of the resource specification. The application uses this ID to refer to the resource when it needs to be accessed. The access scheme (either shared or exclusive) defines whether the same resource can be used by other applications. Resource criticality defines the importance of the resource for the application. Possible values for resource criticality are 3 (obligatory), 2 (essential) and 1 (useful). The actual resource specification is the last parameter, and it is represented with SPARQL SELECT query.

C. Local Resource Manager

The current reference implementation of the LRM component is implemented as a web service that provides RESTful implementation of the operations presented in the table III. The REST interface of the LRM reference implementation is depicted in the table IV.

TABLE IV. REST INTERFACE FOR LRM

Services	**Parameters**	**Returns**
GET lrm/request/:op	op: *required.* Operation to be executed on the resource.	Resource/operation specific value.
POST lrm/authorize	appID: *required.* Identifier for the application. criticality: *required.* Criticality level of the application.	
POST lrm/deauthorize	appID: *required.* Identifier for the application.	
GET lrm/description		RDF representation of the resource

The LRM is implemented with Java programming language using Jersey[4] RESTful Web Services framework. Pre-emptive

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

scheduling is used for scheduling the request made by applications.

IV. Case study in smart home environment

The smart home environment used to evaluate the approach for MC resource management in IoT domain is depicted in the figure 3. The current version of the demonstration contains one resource – a smart lightning system that consists of three Philips Hue LED-bulbs and a Hue gateway. The Hue gateway communicates with smart bulbs using Zigbee and provides HTTP/JSON based interface for external applications. To make the lights controllable from the Web, we have connected the gateway to Ethernet router with static public IP address. The gateway has static IP address in the local network behind the router that forwards messages coming from the Internet to the desired resources in the local network. With this configuration, we can easily add new resources to the system and make the resources available for applications via the Internet.

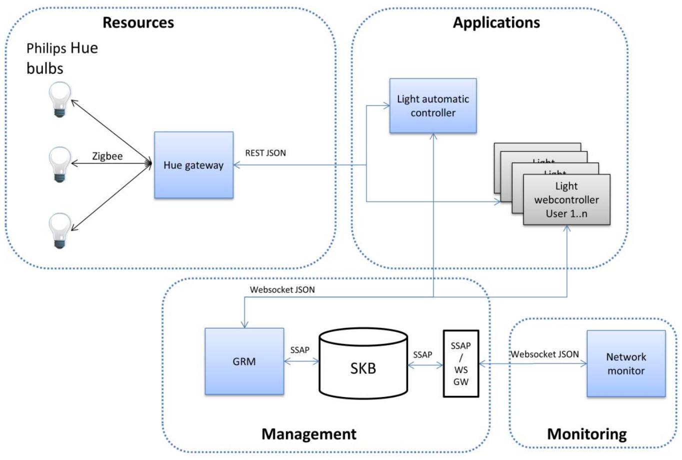
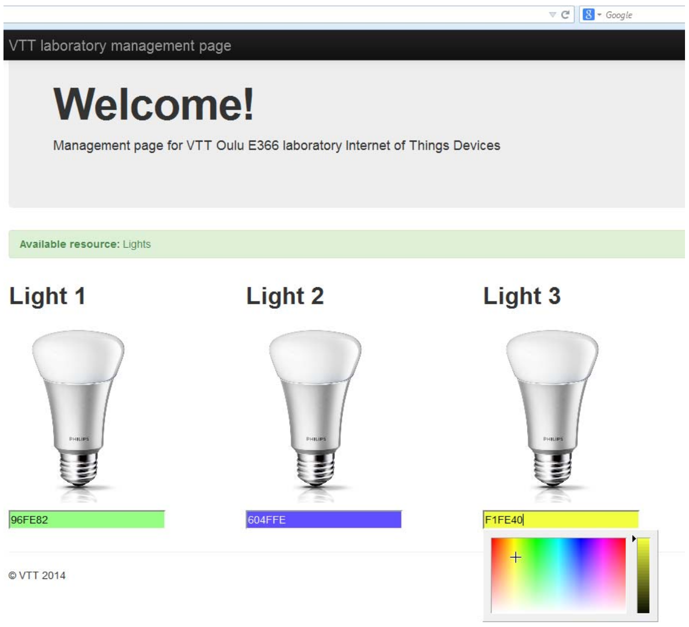


Figure 3. Proof-of-concept use-case in the smart home environment

The demonstration contains two kinds of applications that utilize the lighting system resource: automatic controller and Web user interface (UI). The automatic controller application is a Python script that controls the room lighting automatically, based on the people presence, and the time of the day. It has the lowest criticality (200) of the two application types. In addition to the smart lighting system resource, the automatic controller utilizes a motion detector resource to control the lights based on user presence. However, the motion detector resource is not part of the current implementation, and the control logic is thus based solely to the time of the day information.

The Web UI application provides user with Web page UI to manually control the lights in the room. Utilizing our approach, we can guarantee that needs of the most critical users are served first, and their activities and preferences are not disturbed by less critical users.

Each user of the Web UI is a separate application (with unique application ID) from the resource management perspective. Normal users are assigned with application criticality value of 300. Admin user has the highest criticality (301) in the current system. The Web UI is implemented using both server-side and client-side application development technologies. On the server-side, Twisted-framework is used for communication with the GRM. Flask web framework is used for creating dynamic websites for the user. On the client-side, HTML5 and JavaScript are used to implement the UI (depicted in the figure 4) for controlling the lights. Websocket connection to the GRM has also been created to the client-side in order to provide the user with real-time information about the available resources. Web UI is designed to behave correctly both in desktop browsers mobile browsers. That is how we have avoided making native light control applications for multiple platforms.


VTT laboratory management page

Welcome!

Management page for VTT Oulu E366 laboratory Internet of Things Devices

Available resource: Lights

Light 1 | Light 2 | Light 3

96FE82 | 604FFE | F1FE40

© VTT 2014

Figure 4. Light webcontroller user interface

In addition to the resources, the applications, and resource management components, the prototype implementation contains a monitoring component, called the network monitor. This component is implemented as a web application that uses the SKB websocket interface to monitor the connections between applications and resources. Connections are visualized as a graph with D3.js JavaScript library. The nodes in the graph represent applications and resources. The vertices illustrate which applications use which resources in a given point in time.

To illustrate how the resource management approach works in practice, we will present two scenarios in the smart home environment. The first scenario illustrates the deployment phase. The second scenario presents how the global resource manager allocates the resource for the applications in an example situation.

A. *Deployment phase*

Whenever a new resource or application is deployed into the system, it needs first to be registered to the resource management components. The Philips Hue smart lighting system resource is registered by inserting its resource description into the SKB. This can be done by using the SKB Websocket interface and SPARQL INSERT DATA as presented in the figure 5.

```
[2, 0, "PREFIX ex: <http://example.org#>
        INSERT DATA
        {
          <uuid:913b4850-2f53-11e4-8c21-0800200c9a66> a ex:LightActuator;
                                                      ex:location “E366”.
        }"
]
```

Figure 5. Insert resource description message.

Applications in turn register themselves by sending a register application message to the GRM. The register application messages for the automatic controller and the admin Web UI applications are presented in figures 6 and 7, respectively[5].

```
[2, 0, {
  "application ID": "35fac920-2f4e-11e4-8c21-0800200c9a66",
  "application criticality": "200",
  "resources": [
    {
        "resource specification ID" : "1",
        "access scheme": "Exclusive",
        "resource criticality": "3",
        "query" : "PREFIX ex: <http://example.org#>
                   SELECT ?resource
                   WHERE
                   { ?resource a ex:LightActuator ;
                               ex:location “E366” .
                   }"
    },
    {
        "resource specification ID" : "2",
        "access scheme": "Shared",
        "resource criticality": "1",
        "query" : "PREFIX ex: <http://example.org#>
                   SELECT ?resource
                   WHERE
                   { ?resource a ex:MotionDetector ;
                               ex:location “E366” .
                   }"
    } ]
  }
]
```

Figure 6. Register application message for automatic controller.

[5] The SPARQL query is divided into multiple lines for simplicity sake.

```
[2, 0, {
  "application ID": "bf099bb0-2f4e-11e4-8c21-0800200c9a66",
  "application criticality": "301",
  "resources": [
    {
        "resource specification ID" : "1",
        "access scheme": "Exclusive",
        "resource criticality": "3",
        "query" : "PREFIX ex: <http://example.org#>
                   SELECT ?resource
                   WHERE
                   { ?resource a ex:LightActuator ;
                               ex:location “E366” .
                   }"
    }]
  }
]
```

Figure 7. Register application message for admin Web UI.

When the aforementioned applications are registered to the GRM, it starts searching appropriate resources needed by the applications by initiating the SPARQL SELECT subscriptions defined in the application descriptions. This way the GRM can respond immediately when applications make reservations to the resources. The Philips Hue lights match the resource specifications of both applications. There is no matching resource for the motion detector specification (needed by the automatic controller) available in the system however.

B. *Resource allocation phase*

When the automatic controller and the Web UI need to utilize resources provided by the system, they send the *reserve resource* –message to the GRM. An example scenario of resource reservation process is illustrated in the figure 8.

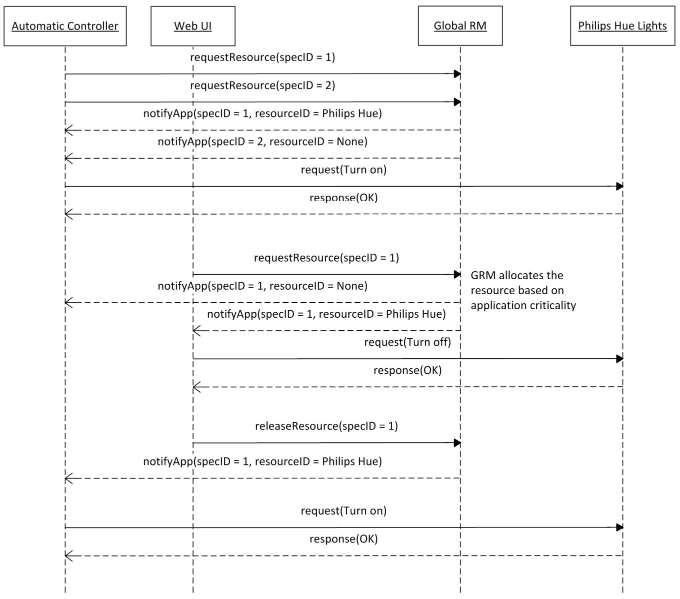

Automatic Controller
Web UI
Global RM
Philips Hue Lights
requestResource(specID = 1)
requestResource(specID = 2)
notifyApp(specID = 1, resourceID = Philips Hue)
notifyApp(specID = 2, resourceID = None)
request(Turn on)
response(OK)
requestResource(specID = 1)
GRM allocates the resource based on application criticality
notifyApp(specID = 1, resourceID = None)
notifyApp(specID = 1, resourceID = Philips Hue)
request(Turn off)
response(OK)
releaseResource(specID = 1)
notifyApp(specID = 1, resourceID = Philips Hue)
request(Turn on)
response(OK)

Figure 8. Message interchange between the resource management components in the example scenario.

The automatic controller reserves the light and motion detector resources immediately after startup. At this point, it is the only application that has active reservation to the Philips Hue resource and the GRM thus grants access to the resource. The GRM also notifies the application that no resources match the second resource specification. The Web UI reserves the Light resource when user opens the website presented in the fig. 4. Because the Web UI has a higher criticality than the automatic controller (and because applications use *exclusive* access scheme), the GRM gives the resource matching the query for the Web UI and denies the access from the automatic controller. When the Web UI is closed the Philips Hue smart lighting system resource is released, and the GRM will notify the automatic controller that the resource is again free to be used.

It should be noted that when GRM grants access to a resource, it should also notify the LRM about the application that has rights to use the resource. The LRM component is not yet integrated into the system, however, and thus, this interaction is not presented in the sequence chart in the figure 8. Instead of dynamically informing the LRM about the applications that are authorized to access the resource, the authorization in the current demonstration is based on static authorization provided by the Philips Hue gateway. In the static authorization process, the application identifier is registered to the Hue gateway by pressing a button on the top of the Hue gateway on each application startup. With the static authorization of all the applications that have registered into the system, the access to the resource is based on a trust that applications do not try to access resources the GRM has not authorized them to access.

V. CONCLUSIONS AND FUTURE WORK

A virtual computing platform that enables 3rd party application developers to access resources provided by heterogeneous group of devices is a key enabler for IoT. This kind of virtual computing platform could open the market for IoT applications in a similar way the smart phones have done for mobile apps. In this paper, we have presented an approach for mixed criticality resource management that is a central building block for the virtual computing platform.

The key idea in the approach is that resources and application are represented with Semantic Web knowledge representation technologies. This makes the approach flexible and extendable for future needs. The resource management is performed at two levels: system and local. At the system level, the role of the resource manager is to manage which resources the applications can access so that the functionality of the whole IoT system is optimal. In turn, the local level resource managers make sure that only the authorized application access their resources, and schedule the access to them based on the criticality of applications. In addition to the approach, central contributions of the paper include reference implementations of the architectural components and a prototype implementation of IoT system demonstration that were used to evaluate the approach in practice. The early experiments obtained from the demonstration indicate that the proposed approach is suitable for MC resource management in IoT domain. Our approach can adapt to the changes in the evolving IoT system and serve the resources to the 3rd party applications based on their criticality.

Although the early experiments obtained from the demonstrations are promising, there are still many limitations and areas for the future work. At the moment, the architecture can only solve conflicts in situations where applications access exactly the same resource. However, there are many situations where different resources can interfere with each other. Therefore, the approach should be extended to cover these situations as well. Typical examples of actuators that can cause this kind of interferences are audio devices (e.g. phones, TV, stereo system, etc.) and heating systems (e.g. fans, radiator, central heating system, etc.). Another future work area is to extend the approach so that it also covers a resource management at the device level. For example, each device in IoT environment typically needs power, and it would be useful if the virtual computing platform could manage how to power is allocated to different devices when power shortage occurs.

In addition to the improvements related to the resource management approach, there is also a room for improvements in the proof-of-concept demonstration. For instance, the current implementation has only one resource and two application types. The plan is to add more resources and applications to evaluate the performance and scalability of the approach better. Additionally, the LRM component needs to be integrated to the demonstration in the future and also evaluate new access scheduling methods. In order to provide necessary tools for application developers and system administrators, we are also planning to improve the network monitoring tool so that it is able to provide more information about the status of the system. A detailed performance and scalability analysis are also needed to evaluate the feasibility of the approach for large scale IoT systems in the future.

ACKNOWLEDGMENT

This work has been done in Intelligent System Development Platform for Intelligent and Sustainable Society (IMPReSS) – project funded by the European Commission under the 7th Framework Programme in the area of EU-Brazil Research and Development cooperation.

REFERENCES

[1] Contributing members of UPnP forum. (2008, October 15). *UPnP Device Architecture 1.1*. 136 p. [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf. Accessed 8/14 2014.

[2] Z. Schelby, K. Hartke, and C. Bormann (2013, August 28) *Constrained Application Protocol (CoAP)*. CoRE Working Group Internet-Draft [Online]. Available: http://datatracker.ietf.org/doc/draft-ietf-core-coap/.

[3] A. Lappeteläinen, J. Tuupola, A. Palin and T. Eriksson, "Networked systems, services and information – the ultimate digital convergence," in *First International NoTA Conference,* Helsinki, Finland, 2008.

[4] IBM and Eurotech. *MQTT V3.1 Protocol Specification* [Online] Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf, Accessed 8/14 2014.

[5] W3C XML Protocol Working Group, W3C Recommendation, In W3C.org http://www.w3.org/TR/soap12-part1/, Accessed 8/14 2014.

[6] N.Koshizuka, K. Sakamura, "Ubiquitous ID:Standards for ubiquitous computing and the internet of things", Pervasive Computing, IEEE 9(4), pp. 98-101

[7] EPCglobal, http://www.gs1.org/epcglobal. Accessed 8/14 2014

[8] Sarnovsky, M., Butka, P., Kostelnik, P., Lackova, D.: HYDRA - Network Embedded System Middleware for Ambient Intelligent Devices. In: ICCC'2007: International Carpathian Control conf. (2007)

[9] Georgantas, N., Issarny, V., Mokhtar, S., Bromberg, Y.D., Bianco, S., Thomson, G.,Raverdy, P.G., Urbieta, A., Cardoso, R.: Middleware Architecture for Ambient Intelligence in the Networked Home. In: Handbook of Ambient Intelligence and Smart Environments, pp. 1139–1169. Springer US (2010)

[10] HomeKit framework, In developer.apple.com https://developer.apple.com/homekit/, Accessed 8/14 2014

[11] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich and D. Chakraborty, "'Intelligent agents meet the semantic Web in smart spaces," *Internet Computing*, IEEE, vol. 8, no. 6, pp. 69-79.

[12] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, Cuong Truong, H. Hasemann, A. Kro□ller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant and R. Richardson, "'SPITFIRE: toward a semantic web of things," Communications Magazine, IEEE, vol. 49, no. 11, pp. 40-48.

[13] A. Burns and R. I. Davis, "Mixed criticality systems: A review", Department of Computer Science, University of York, Tech. Rep. MCC-1(b), 2013.

[14] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In Proc. of the IEEE Real-Time Systems Symposium (RTSS), pages 239–243, 2007.

[15] B. Huber, C. El Salloum, and R. Obermaisser. A resource management framework for mixed-criticality embedded systems. In 34th IEEE IECON, pages 2425–2431, 2008.

[16] M.G. Hill and T.W. Lake. Non-interference analysis for mixed criticality code in avionics systems. In Proceedings of the 15th IEEE international conference on Automated software engineering, pages 257–260. IEEE Computer Society, 2000.

[17] Information Society and Media Directorate-General Unit G3/Computing Systems Research Objective, "Mixed Criticality Systems", report from the Workshop on Mixed Criticality Systems 2012. http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/sra-mixed-criticality-systems.pdf, Accessed 10/14 2014.

[18] S. Harris and A. Seaborne (eds.) (2013, March 21). *SPARQL 1.1 Query Language, W3C Recommendation*. Available: http://www.w3.org/TR/sparql11-query/. Accessed 8/14 2014.

[19] F. Morandi, L. Roffia, A. D'Elia, F. Vergari, and T. S. Cinotti, "RedSib: a Smart-M3 semantic information broker implementation," in Proc. 12th Conf. of Open Innovations Association FRUCT and Seminar on e-Tourism . SUAI, Nov. 2012, pp. 86–98

[20] J. Honkola, H. Laine, R. Brown and O. Tyrkko, "'Smart-M3 information sharing platform," *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pp. 1041-1046.

[21] I. Fette, A. Melnikov (2011), *The WebSocket proto*col, HyBi Working Group, Available: http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17, Accessed 8/14 2014.

[22] K.G. Clark, L. Feigenbaum, E. Torres, (2013, March 21), *SPARQL 1.1 Query Results JSON Format*, Available: http://www.w3.org/TR/sparql11-results-json Accessed 9/01 2014.