Target Outcome: b) Sustainable technologies for a Smarter Society

(FP7 614100)

D7.2.1. Integrated First Proof of Concept IMPRESS platform

18 September 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:	D7.2.1. Integrated First Proof of Concept IMPRESS platform.docx
Document version:	1.0
Document owner:	Ferry Pramudianto (FIT)
Work package:	WP7. IDE Framework for Model-driven development
Task:	Task 7.2 Components Integration
Deliverable type:	P
Document status:	☒ approved by the document owner for internal review ☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Ferry Pramudianto	August 1, 2014	ToC defined
0.2	Ferry Pramudianto	Sept 16 , 2014	Content defined
1.0	Ferry Pramudianto	Sept 18 , 2014	Document finalized

Internal review history:

Reviewed by	Date	Summary of comments
Davide Conzon	Sept 18 , 2014	Accepted with minor comments

Index:

1 Executive summary

The IMPRESS development platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, and performing monitoring and control operations on complex systems. This is achieved through the definition of a number of tools and pre-defined modules that can be managed and combined in order to define a specific logic flow.

In the first iteration we focus on developing the initial middleware solution consisting of the following components:

- Resource adaptation interface aims to provide uniform abstraction of the IoT resources that are exposed by the IoT Service Proxies.
- *Resource Management* operations for solving conflicts, for scheduling and management of mixed-criticality, through the implementation of priority policies. In this phase,
- The *Data, Policy and Knowledge* Storage is responsible for managing the persistence of various data and information. We adopted graph database that is able to provide the necessary expressiveness to store data and information as well as knowledge extracted from for instance historical sensor data, analysed information, learned knowledge, policies, configurations etc.
- The *Data Analysis & Support System Module* provides the necessary algorithms to extract information and pattern among data and increases the effectiveness of the support system operations. The analytics module also enables applications to perform interpolation and data forecasting based on the historical data in order to support the decision-making processes effectively.
- Prototype of a model driven development tool. The tool comprise a visual language that allows developers to define the component of context aware applications visually. Moreover, it allows the developers to define *Sensor and Data Fusion* algorithms to process inputs from the available Application-domain Resources by aggregating and filtering raw data and events. (e.g. calculating the average temperature in a room using temperature measures from sensors deployed in the room or the variable resistor values from voltage and current measures, etc.). The generated java code enables applications to associate the acquired sensor values to the context of the domain objects based on the context model defined by the developers.

The impress platform are integrated through REST based services that can be accessed through different programming languages and computing platform.

The current state of the IMPReSS middleware has provided a framework for the future development of the middleware. Moreover, the initial IDE is able to generate java codes that take an advantage of the available components allowing developer to experiment with context aware applications rapidly. However, a more integrated solution is required in the second iteration e.g. using the data analytics module from the IDE's user interface, generate not only the access to the device proxy but also allows developers to define device proxies from the IDE.

2 Introduction

2.1 Purpose, context and scope of this deliverable

The IMPRESS development platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, and performing monitoring and control operations on complex systems. This is achieved through the definition of a number of tools and pre-defined modules that can be managed and combined in order to define a specific logic flow.

The purpose of this deliverable is to report the progress of the IMPReSS platform in each technical work packages (WP3, WP4, WP5, WP6) and the integration between the components developed within these work packages. Therefore, in this deliverable, the technical components are summarized and the APIs are presented.

The integration activities were mainly driven by WP2 where the architecture was discussed and agreed. To ensure that these components can be integrated easily the partners have agreed to provide REST based API that is supported by different programming languages. That may be used by the partners to create the prototype of the components. Moreover, the partners have agreed to use JSON data format, which still can be used by resource-constrained devices such as Raspberry Pi.

2.2 Background

IMPReSS is a EU-Brazil cooperation project aiming at providing a Systems Development Platform (SDP), which enables rapid and cost effective development of mixed criticality complex systems involving Internet of Things and Services (IoTS) and at the same time facilitates the interplay with users and external systems. The IMPReSS development platform will be usable for any system intended to embrace a smarter society. The demonstration and evaluation of the IMPReSS platform will focus on energy efficiency systems addressing the reduction of energy usage and CO2 footprint in public buildings, enhancing the intelligence of monitoring and control systems as well as stimulating user energy awareness.

The IMPReSS Platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, and performing monitoring and control operations on complex system. The IMPReSS project aims at solving the complexity of system development platform (SDP) by providing a holistic approach that includes an Integrated Development Environment (IDE), middleware components, and a deployment tool.

The architecture presented here is used as the reference for building IMPReSS applications and as such, it provides views on different design aspects and concerns of stakeholders of the IMPReSS platform. A unique software architecture plays a key role in maintaining partners aware of the IMPReSS platform capabilities so that they can always refer to when designing and implementing particular modules. The architecture establishes fundamental concepts and properties of the system contextualized within its environment and expressed by their elements and relationships and evolution guidelines.

3 IMPReSS Development Platform

As envisioned at the beginning of the project, the IMPRESS development platform consists of a set of technologies that help to build general-purpose applications accessing to a plethora of sources, such as information from the physical world, analyzing and fusing relevant data, perform monitoring, and control operations on complex system. This is achieved through the definition of a number of tools and pre-defined modules that can be managed and combined in order to define a specific logic flow.

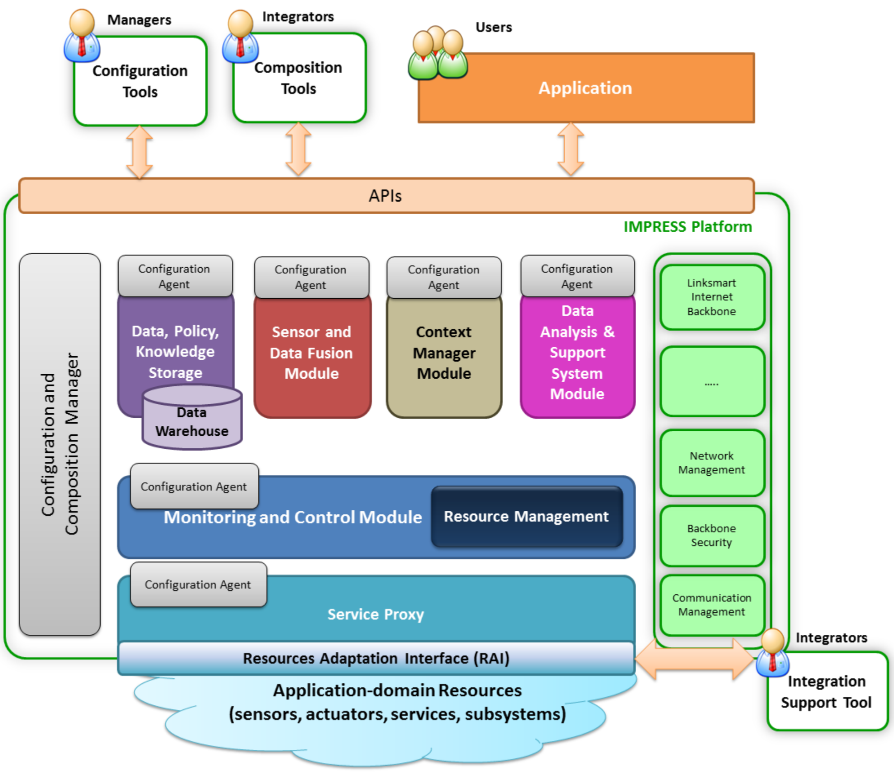


Figure 1. IMPRESS platform.

The *Application-domain Resources* cloud represents all the entities that IMPRESS platform can interoperate with. These entities are:

- Physical world devices (e.g. appliances, sensors and actuators, smart meters, etc.) including personal devices (e.g. smartphones and tablets).
- External and third-parties systems (e.g. pre-existent management systems and networks, business-domain and public authorities systems, third-parties systems, pre-existent systems and networks to avoid extensive retrofitting, third-parties platforms for seamless federation with other similar platforms, subsystems etc.).
- Open and proprietary services (e.g. cloud-based services such as, for instance, weather services, sensor streams, RSS feeds, etc.).

The various resources are seamlessly connected to the IMPRESS platform through *Service Proxies* that expose their functionalities as Internet of Things services, irrespective of their underlying communication protocols. Service Proxies uses a *Resources Adaptation Interface* (RAI) that allows the IMPRESS platform to connect the Application-domain Resources and expose their measurements and capabilities through a common interface and data model.

Monitoring and Control Module aims to optimize complex system operations acting on available Application-domain Resources exposed by the IoT Service Proxies. This module performs also *Resource Management* operations for solving conflicts, for scheduling and management of mixed-criticality, through the implementation of priority policies.

The control algorithms implemented in this module are fed with data collected from Application-domain Resources and with additional information inferred by the processing operations performed by *Sensor and Data Fusion Module* and *Data Analysis & Support System Module*. Context information from *Context Manager Module* can also be useful for monitoring and control tasks. Moreover, the control process can also consider user commands from external applications (e.g. turn-on remotely an appliance from smartphones, etc.) as input.

The *Data, Policy and Knowledge* Storage is responsible for managing the persistence of various data and information. The Data, Policy and Knowledge Storage makes the upper layers and modules independent of where the data is stored, whether locally or in the cloud. It supports relational as well as noSQL storage technologies. Data and information to be maintained include for instance historical sensor data, analyzed information, learned knowledge, policies, configurations etc. It is available for all the components of the IMPRESS platform, storing both raw data and enhanced information. Within the *Data, Policy and Knowledge Storage*, the *Data Warehouse* stores raw data from Application-domain Resources and enhanced data and information inferred by sensor and data fusion modules.

The *Sensor and Data Fusion Module* processes inputs from available Application-domain Resources by aggregating and filtering raw data and events (e.g. to ease scalability storing data with a granularity suitable for the application, to perform high-data-rate applications, etc.) and combining data to synthesize new and enhanced application-domain information (e.g. calculating the average temperature in a room using temperature measures from sensors deployed in the room or the variable resistor values from voltage and current measures, etc.).

The *Context Manager Module* keeps and manages context information. It manages the context information using data extracted from available Application-domain Resources. It associates context information to raw and enhanced values (e.g. stating that temperature sensor, which its unique identifier is '1234', is deployed in the room identified as 'bedroom' on the '3rd floor' of the building 'xyz' sited at '50th Avenue', belonging to 'abed' company, etc.). It defines general data models, possibly based on open standards, to describe the context in a suitable way.

The *Data Analysis & Support System Module* identifies and extracts information such as relations among data and increases the effectiveness of the support system operations; the main function of this module is to extract in a short time the information coming from large amounts of data, in order to effectively use this information in the decision-making processes. It provides support to the control algorithms performed in the *Monitoring and Control Module* and generates suggestions and alarms to user-side application. This module is in charge of performing runtime analysis, allowing the system to be aware of its current status and adapting its operation depending on the context information.

The *Configuration Tool* sets the policies of the whole platform. It shows to the platform *Manager* all the devices and modules belonging to the system, allowing to configure the parameters of the modules of the overall platform.

The *Composition Tool* allows interconnecting the various modules belonging to the platform. This module is a commissioning tool used by the platform *Integrator* that allows defining the connections among the different modules needed to implement specific application logic.

These tools are strictly related to a configuration framework defined within the IMPRESS Platform. This framework is inspired by SNMP architecture and aims at performing the configuration and integration of hardware and software resources. It is composed by two components:

- A Configuration and Composition Manager
- A Configuration Agent

The Configuration and Composition Manager is the module in charge of managing the configuration and composition processes of the other modules into the platform; it works as an interface between the Configuration and Composition Tools and the various modules within the platform.

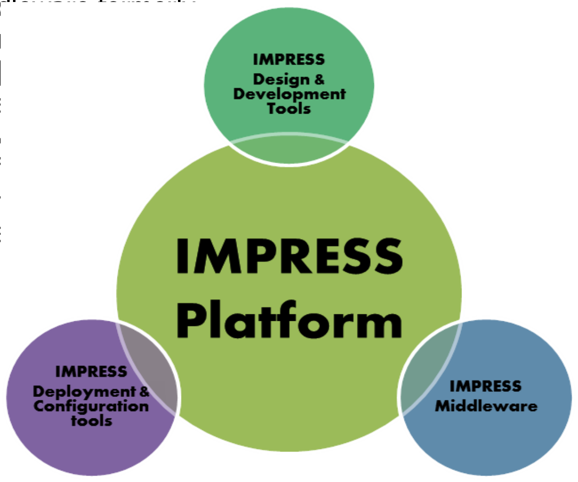
A Configuration Agent is associated with each module of the platform. It exposes configuration and control parameters of a specific module to the Configuration and Composition Manager. The

Configuration Agent operates actually the configuration commands coordinated by Configuration and Composition Manager. The association of an agent to each module makes the system more expandable and scalable from the point of view of configuration issues.

The APIs for interfacing the IMPRESS platform can be divided in three categories:

- APIs for system Integrators: useful to support deployment and installation of the platform. They provide methods for combining different modules and commissioning the specific logic flow.
- APIs for system Managers: useful to set the parameters of the platform modules to make the system effective.
- APIs for system Users: useful to operate on application level functionalities (e.g. for system monitoring and control, fine-grained configuration, etc.).

The IMPRESS platform builds on the LinkSmart Mid
defined within the Hydra project and that is being u
extended in other EU projects ([SEEMPubS] (FP7),
(ARTEMIS) and ebbits (FP7) projects). The LinkSma
provides interoperable interconnection among appli
devices, terminals, subsystems, services and predef
modules. LinkSmart enables a Service Oriented Arc
(SoA) and facilitates interoperability between device
other web services. The green blocks in the figure represent the core of the LinkSmart Middleware. These modules are responsible for the connection of the platform to the Internet Backbone, for the communication and security management (e.g. access control), network administration, etc.


IMPRESS Design & Development Tools
IMPRESS Platform
IMPRESS Deployment & Configuration tools
IMPRESS Middleware

4 Resource adaptation layer (RAI)

RAI aims to abstract the concept of resource (i.e. physical devices or third-party systems), providing a generic "device" that can be used to seamlessly communicate with resources despite of technology-specific implementation details. RAI is located on the extreme edge of the IMPReSS platform, just before the hardware and software resources, and can cooperate with the LinkSmart middleware, which the platform is based on. The heterogeneous nature of physical devices requires finding a way to interact simply with the resources.

A single RAI instance is able to abstract many different resources at the same time. It needs just to execute the specific DeviceManager(s) for the specific resources to abstract. An IMPReSS Service Proxy leverages on RAI APIs and can expose within the IMPReSS ecosystem the resources as "services" to be consumed. The Service Proxy can thus offer a number of services depending on the available resources that RAI can manage.

To generalize the vision of Service Proxy as a service aggregator, we can also consider an architecture as shown in Figure 2, which introduce the Connector concept.

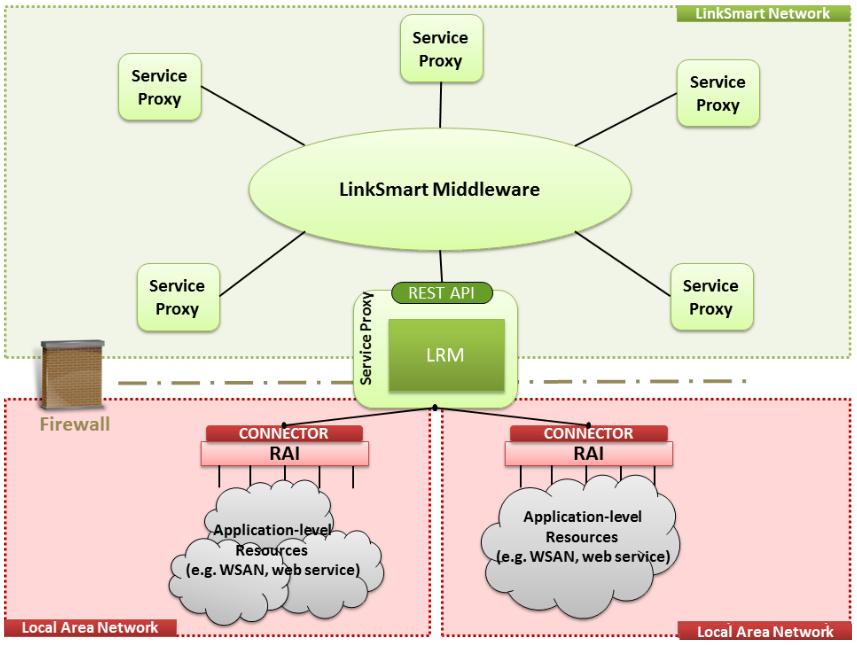


Figure 2: Architecture using RAI connectors

The RAI architecture is composed by three layers, completely decoupled with each other. The lower layer of the architecture consists in a set of technology-specific DeviceManager(s) classes that are responsible for the actual integration of different resources. These components are able to handle specific types of networks. The first implementation of the RAI, the models consist just in a set of Java interfaces that define a basic number of methods/services provided by the most common device types.

The middle layer is the RAI core, which is in charge to map the southbound devices and to notify upper layers about each network changing. The RAI core implements the core interfaces providing an actual implementation of the methods useful to interact with the physical world and get values from physical devices. This component implements the DeviceListener interface, because it has to be aware of the current list of the devices available in the RAI instance, in order to expose them to the other components like the connectors.

The upper layer is responsible for the exposition of the methods/services provided by the resources. This layer is made of the APIs offered by the RAI core, in order to retrieve and manage virtual devices and call their resource-specific methods. The connectors are additional components that are placed upon RAI API (as shown in Figure 3) and are in charge to expose application-level resources, using standard communication protocols like REST, SOAP, XMPP, MQTT, etc.

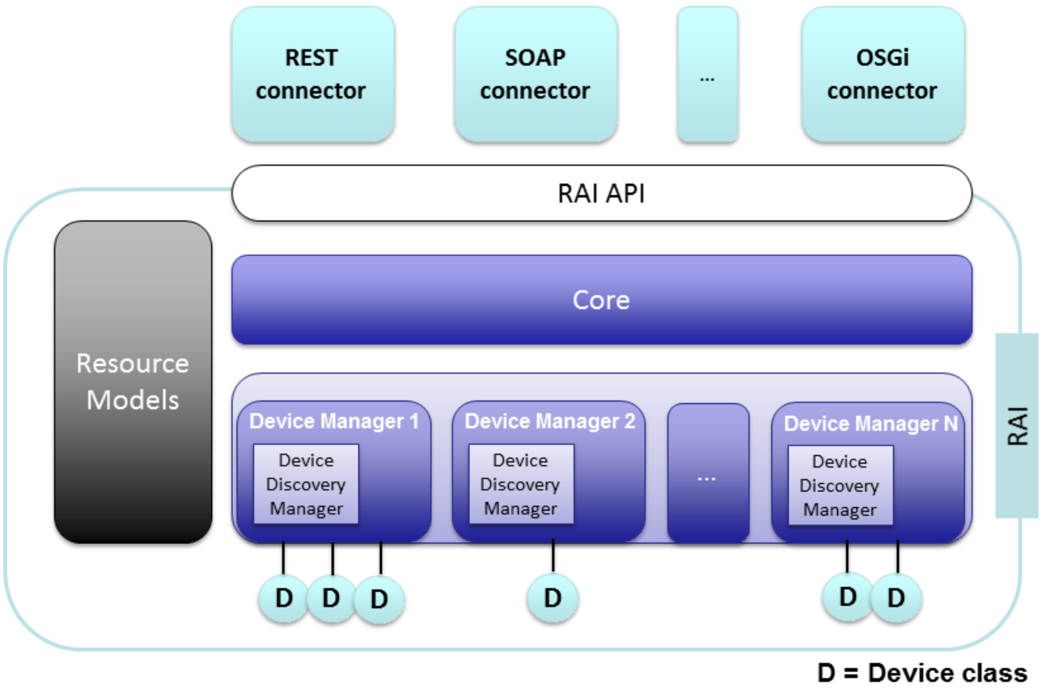


Figure 3. Architecture of the Resource Adaptation Interface

4.1 RAI REST API

In this section are described all the REST resources currently exposed by the RAI REST Connector.

The table below shows a list of REST resources, including a resource description and an example of usage. Currently, the RAI REST Connector returns JSON documents to avoid performance issues, but it will be always possible to modify these resources to return both JSON and XML documents.

Resource	Description	Output example

/devices?deviceType={}&networkType={}	This resource retrieves the detailed list of the devices adapted by the RAI, parsed as a JSON vector. This resource filters device basing on the network and the device type.
	Parameters:
	• **networkType**: the network type to which the device belong (optional)
	• **deviceType**: the device type to which the device belong (optional)
		HTTP GET devices/
		[
		{
		"id": "3300",
		"raiId": "bc71ffa1-20ae-4c2c-8e54-1afb80d3088f",
		"type": "rai:Thermometer",
		"networkType": "rai:Xively",
		"latitude": 43.44984,
		"longitude": -3.83006,
		"updatedAt": "2014-05-14 06:38:08",
		}
		…
		]

/devices/{deviceId}	This resource retrieves details about a specific device, parsed as a JSON.
	Parameters:
	• **deviceID**: is the id assigned by the RAI
		HTTP GET devices/bc71ffa1-20ae-4c2c-8e54-1afb80d3088f
		{
		"id": "3300",
		"raiId": "bc71ffa1-20ae-4c2c-8e54-1afb80d3088f",
		"type": "rai:Thermometer",
		"networkType": "rai:Xively",
		"latitude": 43.44984,
		"longitude": -3.83006,
		"updatedAt": "2014-05-14 06:38:08",
		"temperature": 32
		}

/devicemanagers	This resource retrieves device managers list. The result is a JSON vector containing a list of devices manager available.	HTTP GET /getalldevicemanagers
		[
		{
		"id": "7cfe96f3-6b69-442e-861a-989add208bc5",
		"status": "STARTED",
		"networkType": "rai:Xively"
		}
		……

		]

/devicemanagers/{deviceManagerId}	This resource retrieves device manager details. The result is a JSON containing details about the selected device manager.
	Parameters:
	• **deviceManagerId:** the device manager id assigned by the RAI
		HTTP GET /getalldevicemanagers/7cfe96f3-6b69-442e-861a-989add208bc5
		{
		"id": "7cfe96f3-6b69-442e-861a-989add208bc5",
		"status": "STARTED",
		"networkType": "rai:Xively"
		}

/devicemanagers/{deviceManagerId}	This resource has the same URL of the previous one, but it accepts POST. This can be used to change status to a specific device manager. This resource could be used to change any devices manager field. Possible result are:
	• **200 OK**: everything is correct
	• **503 SERVICE_UNAVAILABLE**: change device manager status fails
	• **404 NOT_FOUND**: the required device manager id does not exist
	Parameters:
	• **deviceManagerId**: is the device manager id assigned by the RAI
		HTTP POST /devicemanagers/{deviceManagerId}
		
		{
		"status": STARTED | STOPPED
		}

5 Resource Management

In the future IoT systems will be open computing platforms that support applications developed by third party developers much in the same way as PCs, tablets, and mobiles phones at the moment. Managing the access to resources between mixed criticality 3rd party applications is a big challenge in these types of open IoT systems. In IMPReSS project, we provide a solution to this challenge by abstracting the resources access and providing platform components for managing the access to application level resources (i.e. sensors and actuators).

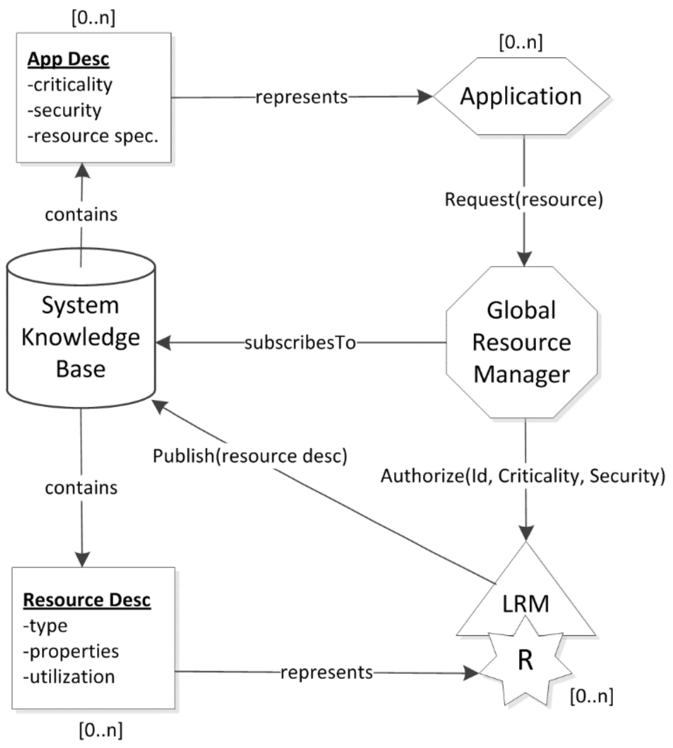


Figure 4. Resource management architecture for mixed critical IoT applications.

The resource management architecture presented in the Figure 4 consists of two levels: global and local. At the global level, the role of the resource management is twofold. First, solve conflicts between applications that either request exclusive access to a same application level resource (ALR) or request access to different resources that might interfere with each other in the real word (e.g. lights, heating systems, etc.). Second, optimize the usage of resources shared between applications so that the performance of the whole IoT system is as optimal as possible. At the global level two components, namely System Knowledge Base and Global Resource Manager are required. The System Knowledge Base stores information about resources, applications and devices in a machine-interpretable format that can be accessed by other IMPReSS components through publish/subscribe communication. The Global Resource Manager is responsible for assigning the actual resources to the applications according to a certain scheduling algorithm.

The Local Resource Manager is responsible for managing access to particular resources and act as a dedicated guard to those local resources. The basic principle in LRM's Scheduler is to guarantee that applications that are more critical are served before less critical ones. It is composed by two sub-components:

- The Scheduler, which regulates the access to resources when multiple applications are authorized to access the same resource. The basic principle in LRM's Scheduler is to guarantee that applications that are more critical are served before less critical ones. A possible approach that can suit IMPReSS goals can be a priority based pre-emptive scheduling. This approach guarantees the more critical applications are served before less critical ones. The main issue of this approach is that it can happen that the less critical applications are not served at all, determining the starvation of lower priority applications when there are many high critical applications. In some cases can be more suitable a round-robin scheduling algorithm, which allow an application to access a resource following the order of the requests. In this case, the scheduler will perform the pre-emption of the running application placing it at the end of a queue of waiting applications and continuing giving access to the next application within the queue.

- The Access Controller, which is responsible for publishing the resource description of the ALR it manages into the System Knowledge Base. This way, the ALRs are "registered" to the Global Resource Manager (GRM). Once GRM grants the access to the ALR (see D4.1.1 for more details), the Access Controller is also responsible for checking if the Application who requests to interact with ALR is the one previously authorized by GRM. If the Application has been authorized, it can proceed calling the specific service provided by the abstracted ALR.

When a new resource (and LRM) is added to the IMPRESS platform, it needs to be first discovered by the Resource Discovery Manager component. The actual methods for resource discovery will be investigated in Task 3.2 and reported in Deliverable 3.2. Once a new resource is found, the Resource Discovery Manager requests the description of the resource and inserts it into the System Knowledge Base. This way other components of the IMPRESS platform are aware of the resources in the network. One of these components is the Global Resource Manager that is subscribed to the resource specifications defined by the applications and will be notified every time a new resource matching the specification is found. This interaction is depicted in the Figure 5.

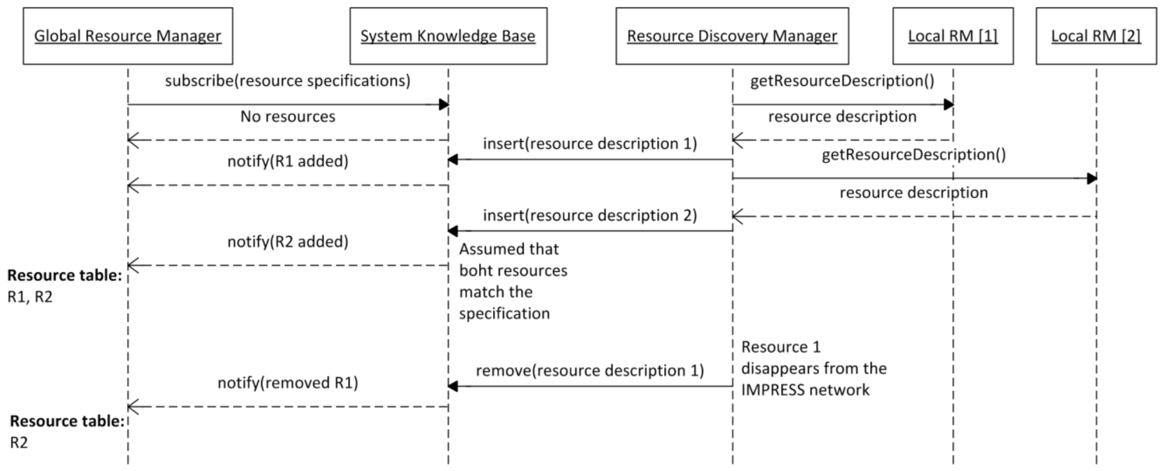


Figure 5. Resource discovery and registration.

Application get access to resources by sending a reserveResource() request to the GRM. The message parameters define the resource specification of interest. The GRM will select the most suitable resource for the application from all the resources matching the given specification and notifies the application about the resource. In order to assign the more suitable resources to the applications, the resource description must be constantly updated with information such as the total service execution time, packet loss rate, service utilization rate, etc. Such kind of information are going to be retrieved by the IMPReSS Network Manager, which will operate at both resource and Service Proxy levels in order to update the resource description on the System Knowledge Base. The role of the Network Manager will be more deeply analysed and defined within Deliverable D3.4. An extended version of the Service Proxy interface will be defined in order to provide relevant services to the Network Manager. Additionally, the GRM informs the LRM about the application authorized to access the resource. This is done with the authorizeAccess() message which defines the application ID, the criticality, and the required security level. Once the application and the LRM have been notified about the pairing the application can start requesting domain specific services provided by the resource. This happens be sending a requestResource() message to the LRM with the actual domain specific operation as payload. If the LRM receives multiple simultaneous requests it schedules them based on the criticality of the application. This way the more critical applications are always served before less critical ones. The

Figure 6 illustrates message exchange between applications, GRM and LRMs in an example scenario where two applications need to access a sensor resource using shared access scheme.

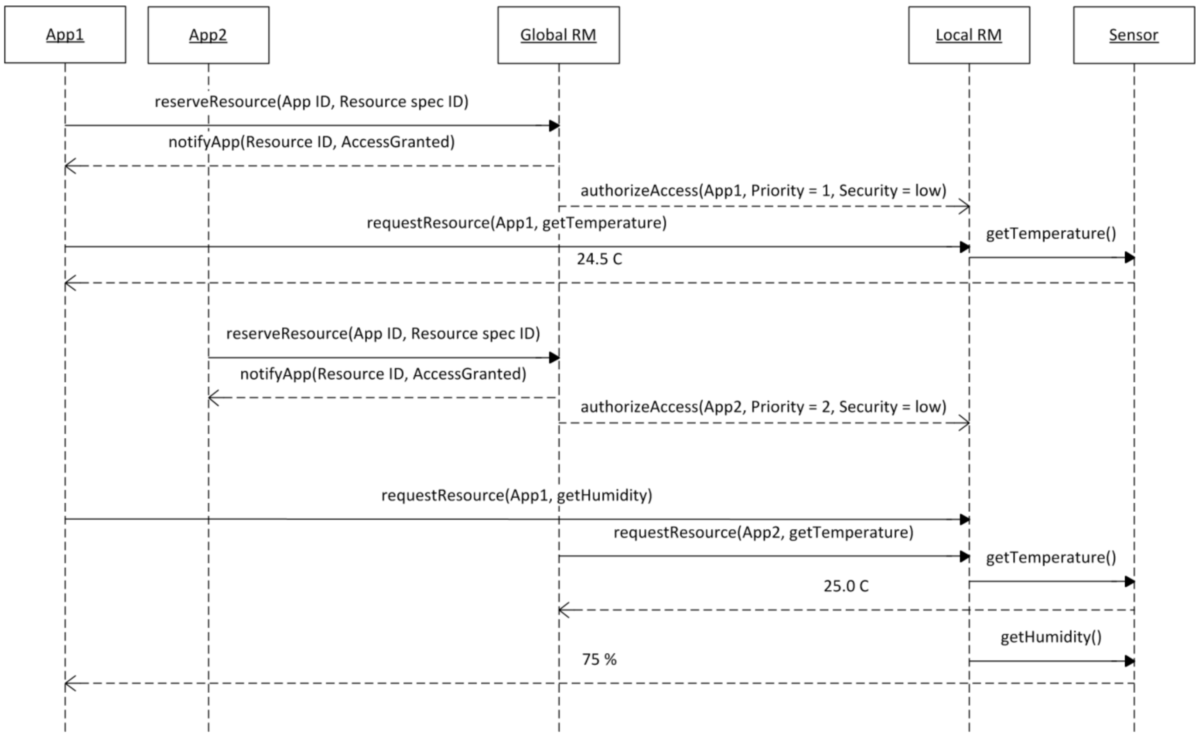

App1
App2
Global RM
Local RM
Sensor
reserveResource(App ID, Resource spec ID)
notifyApp(Resource ID, AccessGranted)
authorizeAccess(App1, Priority = 1, Security = low)
requestResource(App1, getTemperature)
getTemperature()
24.5 C
reserveResource(App ID, Resource spec ID)
notifyApp(Resource ID, AccessGranted)
authorizeAccess(App2, Priority = 2, Security = low)
requestResource(App1, getHumidity)
requestResource(App2, getTemperature)
getTemperature()
25.0 C
getHumidity()
75 %

Figure 6. Message exchange between applications, global RM, local RM, and a resource.

5.1 Local Resource Manager API

In this Section, the APIs exposed by the LRM to the GRM for the control of ALRs are described. The REST interface is described as follows:

GET lrm/request_resource/:operation

POST lrm/authorize_access

POST lrm/deauthorize_access

GET lrm/description

Services

GET lrm/request_resource/:operation

Description:

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

Parameters:

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

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

Returns:

Resource/Operation specific value.

Example request:

GET http://127.0.0.1/lrm/request_resoure/getTemperature

POST lrm/authorize_access

Description:

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

Parameters:

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

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

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

Example request:

POST http://127.0.0.1/lrm/authorize_access

POST Data: appID:app124&priority=1&securityLevel=low

POST lrm/deauthorize_access

Description:

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

Parameters:

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

Example request:

POST http://127.0.0.1/deauthorize_access

POST Data: appID:app12

GET lrm/get_description

Description:

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

Returns:

Description of Resource.

Example request:

GET http://127.0.0.1/lrm/get_description

6 Data Storage and Analytics

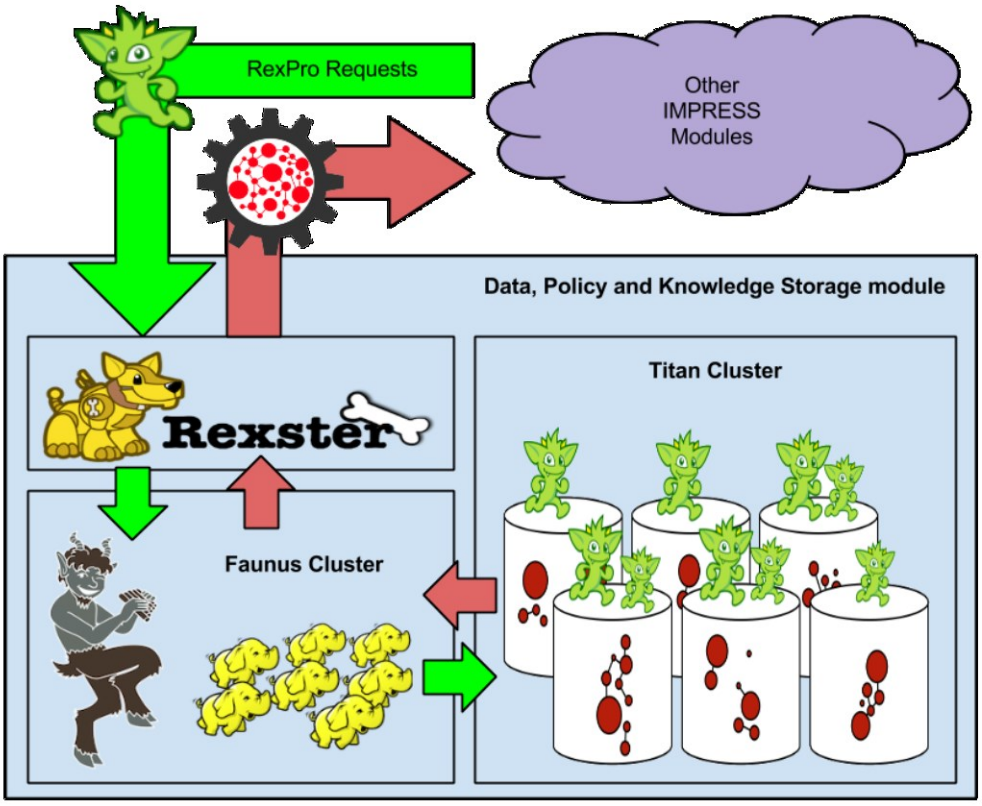
6.1 Data Storage Module

Under the IMPRESS platform, the data semantics and analytics are the fundamental features needed to support the decision making process. Multi sensor data fusion provides a means to fuse raw data into meaningful higher-level information for the users. Moreover, the recognition of the modelled situations requires understanding the technicalities of each sensor, signal processing, and sensor fusion techniques to combine readings from different sensors. In such scenario, where the information about the interconnectivity or the topology of the data is more important than, or as important as, the data itself, the data modelling based on graph has several advantages.

First, graphs provide a natural and flexible way to represent information about real world (i.e. real world objects are nodes and relations between different objects are vertexes/edges). Second, typical graph databases provide built-in structures (i.e. nodes and edges) to represent graphs. Whereas in other databases, relationships between entities in the data model would have to be handled by the modeler at the model level. In other words, new tables or columns, at least in the SQL case, would have to be maintained only for the sake of being used as query indirection stages that point to other entities, probably via foreign keys. For these reasons, the data modelling adopted in the project is based on a property graph representation. In the realm of graphs' morphism, a property graph is a vertex/edge-labeled/attributed, directed, multi-graph. On sufficiently powerful machines, local graph databases can support a couple billion edges while distributed systems can handle hundreds of billions of edges. However most distributed graph-based NoSQL databases, like Neo4j[1], does not provide the means for global graph algorithms to be per-formed within a reasonable milliseconds time scale, in a hundreds of billions of edges scenario. Since the data storage in IMPReSS aims to support data processing for machine learning and data fusion techniques, be able to have a continuous feedback loop that works almost in quasi real time and have a global view of the current and past state of the system is required. Considering this practical concern, we adopt Titan[2] that supports several storage backend such as Cassandra, which is a column-family NoSQL database developed and open sourced by Facebook in 2008, Hbase[3], which is an open source implementation of Google's BigTable, Oracle Berkeley DB[4] and Akiban Persistit[5]. To support distributed batch processing and support graph analytics in a timely manner, we use Faunus[6]. Faunus is able to distribute queries steps in the available Titan clusters and performs load balancing of the workload. Therefore, drastically reducing latency for database operations in graphs with billions of edges and nodes. Faunus works on top of Hadoop[7], which is an open source project backed by the Apache Foundation and based on Google's Map-Reduce white paper. For querying, we use a domain-specific language, the Gremlim language[8], which can perform complex operations in multi-relational graphs, called property graphs. Gremlim is based on the Groovy language[9]. With Gremlin it is possible to perform operations such as the addition or removal of nodes/edges, manipulate the graph indexes, complex graphs transversals, etc. Also, it is part of the Blueprints[10] stack. Finally, we chose to use a Rexter server[11], which is also part of the

[1] http://www.neo4j.org/
[2] http://thinkaurelius.github.io/titan/
[3] https://hbase.apache.org/
[4] http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html
[5] http://www.akiban.com/
[6] http://thinkaurelius.github.io/faunus/
[7] https://hadoop.apache.org/
[8] https://github.com/tinkerpop/gremlin
[9] http://groovy.codehaus.org/
[10] http://thinkaurelius.github.io/titan/
[11] https://github.com/thinkaurelius/titan/wiki/Rexster-Graph-Server

Blueprints stack, to be the interface exposed for developers to execute database operations, via Gremlim queries. the Rexter server allows developers to communicate with Blueprints-enabled graphs in a language agnostic fashion. That is, we could change the underlying NoSQL graph database at any time, without requiring any source code change in the clients of the Data, Policy and Knowledge Storage module. Using Rexter, developers can access the Blueprints API over HTTP/REST directly or by using libraries that support Blueprints API. The Rexter server supports both a JSON-based REST interface and a binary protocol called RexPro. In our architecture, we favoured the RexPro case due to its smaller footprint. When a Gremlim query is received, the Rexter server passes it to one of the Faunus clusters and waits for the response, which is then replied back to the requester.



6.1.1 REXSTER Basic REST API

6.1.1.1 GET Operations

returns	uri	description
graphs	`/graphs`	get all the graphs
graph	`/graphs/<graph>`	get the graph named
vertices	`/graphs/<graph>/vertices`	get all vertices
vertices	`/graphs/<graph>/vertices?key=<key>&value=<value>`	get all vertices for a key index given the specified`<key>/<value>`
vertex	`/graphs/<graph>/vertices/<id>`	get vertex with id`<id>`

vertices	`/graphs/<graph>/vertices/<id>/out`	get the adjacent out vertices of vertex `<id>` [4]
vertices	`/graphs/<graph>/vertices/<id>/in`	get the adjacent in vertices of vertex`<id>` [4]
vertices	`/graphs/<graph>/vertices/<id>/both`	get the both adjacent in and out vertices of vertex `<id>` [4]
long	`/graphs/<graph>/vertices/<id>/outCount`	get the number of out vertices of vertex `<id>` [4]
long	`/graphs/<graph>/vertices/<id>/inCount`	get the number of in vertices of vertex `<id>` [4]
long	`/graphs/<graph>/vertices/<id>/bothCount`	get the number of adjacent in and out vertices of vertex `<id>` [4]
longs	`/graphs/<graph>/vertices/<id>/outIds`	get the identifiers of out vertices of vertex `<id>` [4]
longs	`/graphs/<graph>/vertices/<id>/inIds`	get the identifiers of in vertices of vertex `<id>` [4]
longs	`/graphs/<graph>/vertices/<id>/bothIds`	get the identifiers of adjacent in and out vertices of vertex `<id>` [4]
edges	`/graphs/<graph>/edges`	get all edges
edges	`/graphs/<graph>/edges?key=<key>&value=<value>`	get all edges for a key index given the specified`<key>`/`<value>`
edge	`/graphs/<graph>/edges/<id>`	get edge with id`<id>`
edges	`/graphs/<graph>/vertices/<id>/outE`	get the out edges of vertex `<id>` [4]
edges	`/graphs/<graph>/vertices/<id>/inE`	get the in edges of vertex `<id>` [4]
edges	`/graphs/<graph>/vertices/<id>/bothE`	get the both in and out edges of vertex `<id>` [4]
indices	`/graphs/<graph>/indices`	get all the indices associated with the graph

elements	`/graphs/<graph>/indices/index?key=<key>&value=<value>`	get all elements with `<key>`property equal to`<value>` in `index`
long	`/graphs/<graph>/indices/index/count?key=<key>&value=<value>`	get a count of all elements with`<key>` property equal to `<value>`in `index`
keys	`/graphs/<graph>/keyindices/`	get the combination of vertex and edge keys
keys	`/graphs/<graph>/keyindices/vertex`	get vertex keys
keys	`/graphs/<graph>/keyindices/edge`	get edge keys
prefixes[1]	`/graphs/<graph>/prefixes`	get the list of SailGraph prefixes
prefix [1]	`/graphs/<graph>/prefixes/prefix`	get a specific prefix value

6.1.1.2 POST Operations

returns	uri	description
vertex	`/graphs/<graph>/vertices`	create a vertex with no specified identifier
vertex	`/graphs/<graph>/vertices/<id>`	create a vertex with id `<id>` [2]
vertex	`/graphs/<graph>/vertices/<id>?<key>=<value>&<key'>=<value'>`	create a vertex with id `<id>` and the provided properties (or update vertex properties if vertex already exists). [2]
edge	`/graphs/<graph>/edges?_outV=<id>&_label=friend&_inV=2&<key>=<key'>`	create an out edge with no specified identifier from vertex `<id>` to vertex `2` labeled “friend” with provided properties. [2]
edge	`/graphs/<graph>/edges/3?_outV=<id>&_label=friend&_inV=2&<key>=<key'>`	create an out edge with id `3` from vertex `<id>` to vertex `2` labeled “friend” with provided properties. [2]
edge	`/graphs/<graph>/edges/3?<key>=<key'>`	update the properties of the edge with id `3`
index [3]	`/graphs/<graph>/indices/index?class=vertex`	create a new manual index named `index`

void	`/graphs/<graph>/keyindices/vertex/<key>`	create a new key index for a vertex
void	`/graphs/<graph>/keyindices/edge/<key>`	create a new key index for an edge
void [1]	`/graphs/<graph>/prefixes?namespace=http%3A%2F%2Fwww.ggl.com&prefix=pf`	add a new SailGraph prefix with `namespace`http://www.ggl.com and `prefix` pf

6.1.1.3 PUT Operations

returns	uri	description
vertex	`/graphs/<graph>/vertices/<id>?<key>=<value>&<key'>=<value'>`	replaces the all existing properties of the vertex `<id>`with those specified
edge	`/graphs/<graph>/edges/<id>?<key>=<value>&<key'>=<value'>`	replaces the all existing properties of the edge `<id>`with those specified
void	`/graphs/<graph>/indices/index?key=<key>&value=<value>&id=<id>`	put vertex with id `<id>` into`index` at `<key>/<value>`

6.1.1.4 DELETE Operations

returns	uri	description
void	`/graphs/<graph>/vertices/<id>`	remove vertex `<id>`
void	`/graphs/<graph>/vertices/<id>?<key>&<key'>`	remove properties `<key>`and `<key'>` from vertex`<id>`
void	`/graphs/<graph>/edges/3`	remove the edge with id`3`
void	`/graphs/<graph>/edges/3?<key>&<key'>`	remove properties `<key>`and `<key'>` from edge`3`
void	`/graphs/<graph>/indices/index`	drop the index named`index`
void	`/graphs/<graph>/indices/index?key=<key>&value=<value>&class=vertex&id=<id>`	remove the vertex `<id>`from `index` at`<key>/<value>`
void [1]	`/graphs/<graph>/prefixes/prefix`	remove the specified prefix

6.2 Data analytics module

This section shows an overview of the Data Analytics module API. The main goal of this API is to provide to the developers tools related to data analytics, precisely statistics, optimization, data mining and machine learning algorithms. This API will be part of the IMPRESS Cloud and will be available through a web service. Figure 7 shows the Data Analytics module API Overview.

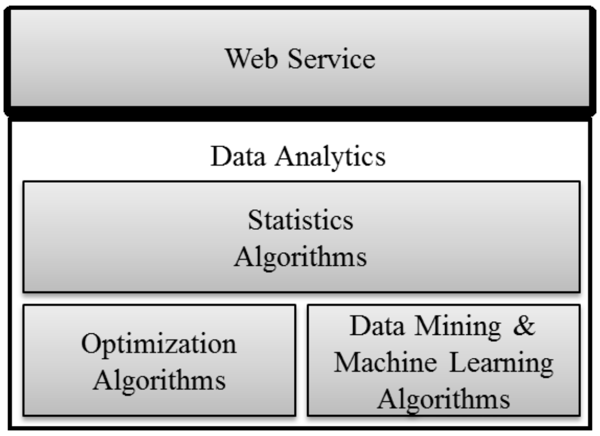


Figure 7. Data Analytics API Overview.

There is a web service, which implements the Data Analytics module API, in order to allow developers to have access to the Data Analytics module API. Deliverable 8.2 presents details about how the web services URLs should be formatted in order to guarantee consistency among IMPReSS' different modules. The second part of Figure 7 shows how the data analytics process is divided at the IMPReSS Platform. Data Analytics entails the process of examining large amounts of data to uncover hidden patterns, unknown correlations and other useful information that can be used to generate better decisions. With big data analytics, data scientists and other users can analyze huge volumes of data that conventional analytics and business intelligence solutions cannot do. This API offers four groups of algorithms to analyze raw data: statistics, optimization, data mining and machine learning algorithms. These groups are better depicted in Figure 8.

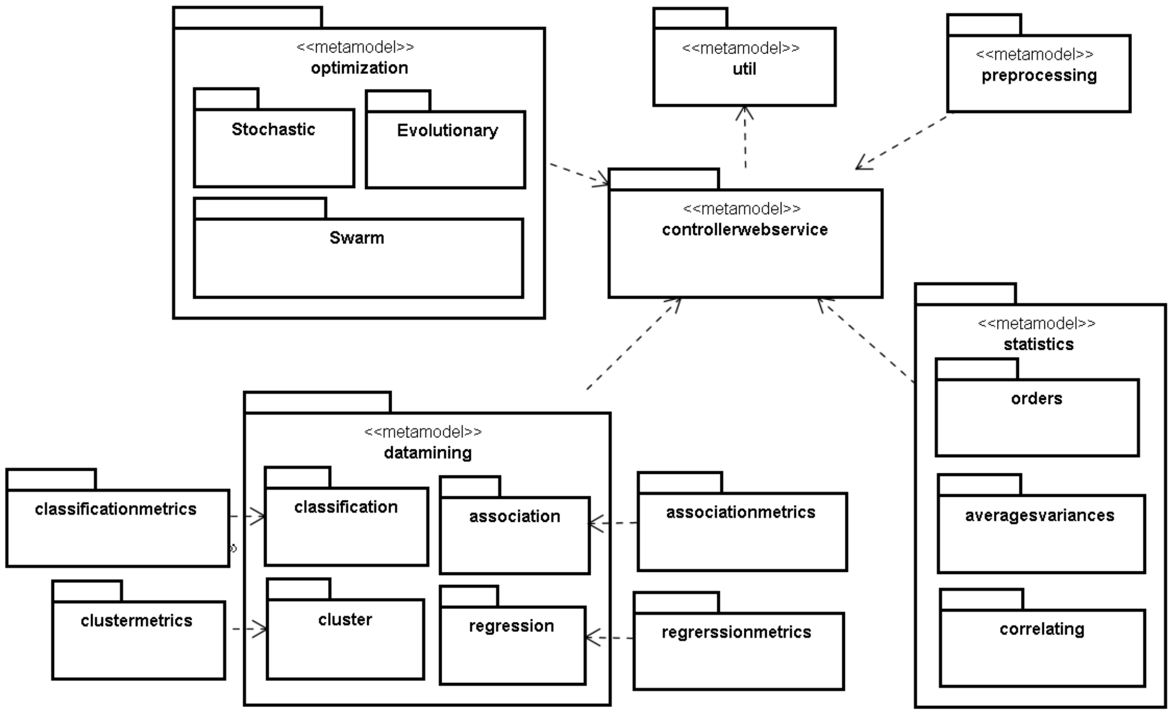


Figure 8. Data Analytics module API package structure.

The design of this API follows patterns from the Java Specification Request (JSR)[1] through the creation of packages. Furthermore, data mining algorithms were taken from the scikit-learn library[12], statistics algorithms from pymc library[13] and optimization algorithms from Clever Algorithms[2].
Before describing data mining, optimization and statistics, there are other packages that compose the Data Analytics module API. The first one is the **impress.util** package that consists of some utility classes, which may be used to assist the other packages, for example, a class can be used to call a method to format data. The second package is the **impress.controllerwebservice** that has classes to expose the Data Analytics module API through a web service and makes it available to developers. Deliverable 8.2 has more details about the web service specification. The last package is the **impress.preprocessing** that consists of several classes to allow data preprocessing work before any data analytics algorithm is called. **Impress.preprocessing** algorithms are shown in Table 1. The preprocessing step may involve several strategies, for instance, when there are data with missing features, imputation methods should be employed to substitute missing values with meaningful estimates in order to allow machine-learning algorithms to be used. In addition, when data is available in different scales, normalization and standardization preprocessing should be used to allow the effective use of learning methods based on distance measures, such as Euclidian distance.

Table 1: Preprocessing algorithms.

Classes	Description
Binarizer([threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold
Imputer([missing_values, ...])	Imputation methods for completing missing values
KernelCenterer	Center a kernel matrix
LabelBinarizer([neg_label, ...])	Binarize labels in a one-vs-all fashion
LabelEncoder	Encode labels with value between 0 and n_classes-1.
MinMaxScaler([feature_range, copy])	Standardizes features by scaling each feature to a given range.
Normalizer([norm, copy])	Normalize samples individually to unit norm
StandardScaler([copy, ...])	Standardize features by removing the mean and scaling to unit variance
binarize(X[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix
label_binarize(y, classes[, ...])	Binarize labels in a one-vs-all fashion
normalize(X[, norm, axis, copy])	Normalize a dataset along any axis
scale(X[, axis, with_mean, ...])	Standardize a dataset along any axis

Figure 9 consists of several data mining and machine learning algorithms, as well as metrics that each group uses to evaluate their algorithms. These algorithms and metrics will be explained below:

- **impress.datamining.classification**: This sub-package defines classes that implements classification algorithms.
- **impress.datamining.association:** This sub-package defines classes that implements association algorithms.
- **impress.datamining.cluster:** This sub-package defines classes that implements clustering algorithms.
- **impress.datamining.regression:** This sub-package defines classes that implements regression algorithms.
- **impress.datamining.classification.metrics:** This sub-package defines metrics classes to evaluate data mining classification algorithms.
- **impress.datamining.association.metrics:** This sub-package defines metrics classes to evaluate association algorithms.

[12] http://scikit-learn.org/stable/index.html
[13] http://pymc-devs.github.io/pymc/

- **impress.datamining.cluster.metrics:** This sub-package defines metrics classes to evaluate data mining clustering algorithms.
- **impress.datamining.regression.metrics:** This sub-package defines metrics classes to evaluate data mining regression algorithms.

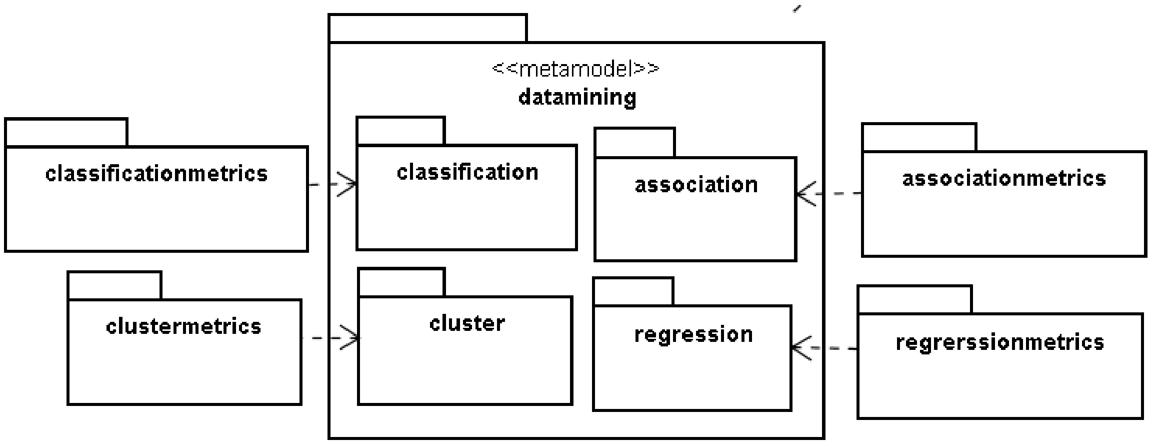


Figure 9. Data Mining package structure.

The algorithms of each package cited above are shown in the tables bellow:

Table 2. Classification algorithms.

Classes	Description
RandomForestClassifier([...])	Ensemble of decision tree classifiers generated by random forest.
RandomTreesEmbedding([...])	Ensemble of totally random trees.
AdaBoostClassifier(...[, criterion])	Ensemble of classifiers generated by the AdaBoost ensemble generation method.
GradientBoostingClassifier([loss, ...])	Ensemble of classifiers generated by the Gradient Boosting method.
GaussianHMM([n_components, ...])	The Hidden Markov Model classifier generated with Gaussian emissions
GaussianNB	The Naive Bayes Classifier generated with Gaussian functions
NearestCentroid([metric, ...])	The prototype-based Nearest centroid classifier.
SVC([C, kernel, degree, gamma, coef0, ...])	The Support Vector Machines classifier C-Support Vector Classification for nonlinear separated problems.
LinearSVC([penalty, loss, dual, tol, C, ...])	Linear Support Vector Classification for linear separated problems.
DecisionTreeClassifier([criterion, ...])	A decision tree classifier.

Table 3. Evaluation metrics for classification algorithms.

Classes	Description
accuracy_score(y_true, y_pred[, ...])	Accuracy classification score.
average_precision_score(y_true, y_score)	Compute average precision (AP) from prediction scores
classification_report(y_true, y_pred)	Build a text report showing the main classification metrics
confusion_matrix(y_true, y_pred[, ...])	Compute confusion matrix to evaluate the accuracy of a classification
jaccard_similarity_score(y_true, y_pred)	Jaccard similarity coefficient score

log_loss(y_true, y_pred[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds
precision_score(y_true, y_pred[, ...])	Compute the precision
recall_score(y_true, y_pred[, ...])	Compute the recall
roc_auc_score(y_true, y_score)	Compute Area Under the Curve (AUC) from prediction scores

Table 4. Clustering algorithms.

Classes	Description
DBSCAN([eps, min_samples, metric, ...])	Perform DBSCAN clustering from vector array or distance matrix.
KMeans([n_clusters, init, n_init, ...])	The simple K-Means clustering algorithm
MiniBatchKMeans([n_clusters, init, ...])	Mini-Batch K-Means clustering algorithm
MeanShift([bandwidth, seeds, ...])	MeanShift clustering algorithm
Ward([n_clusters, memory, ...])	Ward hierarchical clustering: constructs a tree and cuts it.
NearestNeighbors([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.

Table 5. Evaluation metrics for Clustering algorithms.

Classes	Description
adjusted_mutual_info_score(...)	Adjusted Mutual Information between two clusters
adjusted_rand_score(labels_true, ...)	Rand index adjusted for chance
completeness_score(labels_true, ...)	Completeness metric of a cluster labeling given a ground truth
homogeneity_score(labels_true, ...)	Homogeneity metric of a cluster labeling given a ground truth
mutual_info_score(labels_true, ...)	Mutual Information between two clusters
normalized_mutual_info_score(...)	Normalized Mutual Information between two clusters
consensus_score(a, b[, similarity])	The similarity of two sets of biclusters.

Table 6. Regression algorithms.

Classes	Description
RandomForestRegressor([...])	Ensemble of random forest regressor.
AdaBoostRegressor(...[, criterion, ...])	Ensemble based on AdaBoost regressor.
GradientBoostingRegressor([loss, ...])	Gradient Boosting for regression.
IsotonicRegression([y_min, y_max, ...])	Isotonic regression model.
LinearRegression([...])	Ordinary least squares Linear Regression.
LogisticRegression([penalty, ...])	Logistic Regression classifier.
RandomizedLogisticRegression([...])	Randomized Logistic Regression
KNeighborsRegressor([n_neighbors, ...])	Regression based on k-nearest neighbors.
RadiusNeighborsRegressor([radius, ...])	Regression based on neighbors within a fixed radius.
SVR([kernel, degree, gamma, coef0, tol, ...])	epsilon-Support Vector Regression.

Table 7. Evaluation metrics for Regression algorithms.

Classes	Description
explained_variance_score(y_true, y_pred)	Explained variance regression score function

mean_absolute_error(y_true, y_pred)	Mean absolute error regression loss
mean_squared_error(y_true, y_pred)	Mean squared error regression loss
r2_score(y_true, y_pred)	R^2 (coefficient of determination) regression score function.

Package **impress.optimization** consists of several optimization algorithms, such as Tabu Search, Evolutionary Algorithms and Particle Swarm Optimization, as depicted in Figure 10 and explained below.

- **impress.optimization.stochastic:** This sub-package defines classes that implements stochastic optimization algorithms.
- **impress.optimization.evolutionary:** This sub-package defines classes that implements evolutionary optimization algorithms.
- **impress.optimization.swarm:** This sub-package defines classes that implements swarm optimization algorithms

The algorithms of each of these packages are shown in the tables bellow:

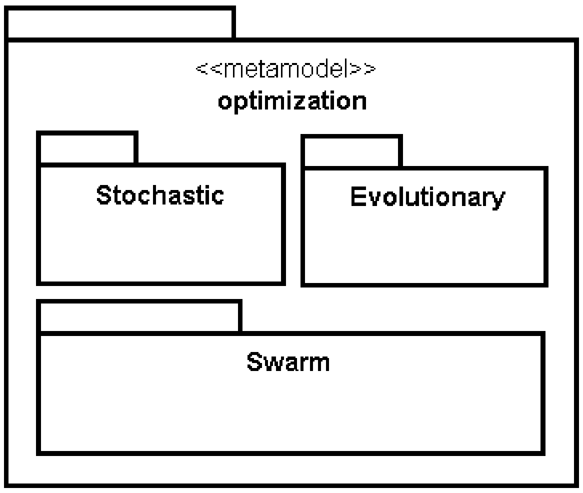


Figure 10. Data analytics module optimization package structure.

Table 8. Stochastic algorithms.

Classes	Description
tabusearch(<parameters>)	Tabu Search is a Global Optimization algorithm and a Metaheuristic or Meta-strategy for controlling an embedded heuristic technique.
randomsearch(<parameters>)	Random search belongs to the elds of Stochastic Optimization and Global Optimization.
scattersearch(<parameters>)	Scatter search is a Metaheuristic and a Global Optimization algorithm.

Table 9. Evolutionary algorithms.

Classes	Description
geneticalgorithm(<parameters>)	The Genetic Algorithm is an Adaptive Strategy and a Global Optimization technique.
nsga(<parameters>)	The Non-dominated Sorting Genetic Algorithm is a Multiple Objective Optimization (MOO) algorithm and is

	an instance of an Evolutionary Algorithm from the field of Evolutionary Computation.
spea(<parameters>)	Strength Pareto Evolutionary Algorithm is a Multiple Objective Optimization (MOO) algorithm and an Evolutionary Algorithm from the field of Evolutionary Computation.

Table 10. Swarm algorithms.

Classes	Description
pso(<parameters>)	Particle Swarm Optimization belongs to the field of Swarm Intelligence and Collective Intelligence and is a sub-field of Computational Intelligence.
beesalgorithm(<parameters>)	The Bees Algorithm beings to Bee Inspired Algorithms and the field of Swarm Intelligence, and more broadly the elds of Computational Intelligence and Metaheuristics.
antcolonysystem(<parameters>)	The Ant Colony System algorithm is an example of an Ant Colony Optimization method from the field of Swarm Intelligence, Metaheuristics and Computational Intelligence.

Package **impress.statistics** consists of several statistics algorithms, such as orders, average, variance and correlating, as depicted in Figure 11 and explained below.

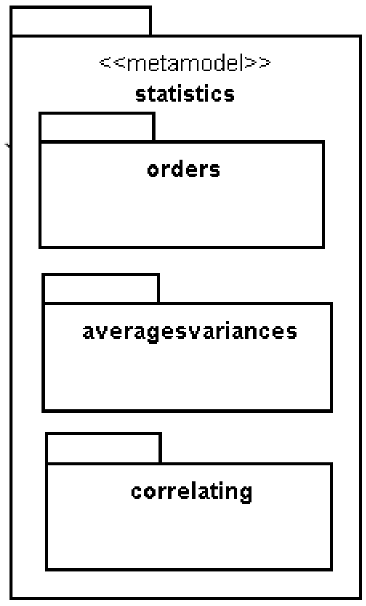


Figure 11. IMPRESS statistic package structure.

- **impress.statistics.orders**: This sub-package defines classes that implement the extraction of basics information, i.e. minimum and maximum.
- **impress.statistics.averagevariances**: This sub-package defines classes that implement the extraction of average and variances of the data.
- **impress.statistics.correlating:** This sub-package defines classes that implement the extraction of correlation coefficients.

The algorithms of each package are shown in the tables bellow:

Table 11. Order algorithms.

Classes	Description

amin(a[, axis, out, keepdims])	Returns the minimum of an array or minimum along an axis.
amax(a[, axis, out, keepdims])	Returns the maximum of an array or maximum along an axis.
nanmin(a[, axis, out, keepdims])	Returns minimum of an array or minimum along an axis, ignoring any NaNs.
nanmax(a[, axis, out, keepdims])	Returns the maximum of an array or maximum along an axis, ignoring any
ptp(a[, axis, out])	Range of values (maximum - minimum) along an axis.
percentile(a, q[, axis, out, overwrite_input])	Compute the qth percentile of the data along the specified axis.

Table 12. Average/Variance algorithms.

Classes	Description
median(a[, axis, out, overwrite_input])	Computes the median along the specified axis.
average(a[, axis, weights, returned])	Computes the weighted average along the specified axis.
mean(a[, axis, dtype, out, keepdims])	Computes the arithmetic mean along the specified axis.
std(a[, axis, dtype, out, ddof, keepdims])	Computes the standard deviation along the specified axis.
var(a[, axis, dtype, out, ddof, keepdims])	Computes the variance along the specified axis.
nanmean(a[, axis, dtype, out, keepdims])	Computes the arithmetic mean along the specified axis, ignoring NaNs.
nanstd(a[, axis, dtype, out, ddof, keepdims])	Computes the standard deviation along the specified axis, while
nanvar(a[, axis, dtype, out, ddof, keepdims])	Computes the variance along the specified axis, while ignoring NaNs.

Table 13. Correlating algorithms.

Classes	Description
corrcoef(x[, y, rowvar, bias, ddof])	Returns correlation coefficients.
correlate(a, v[, mode, old_behavior])	Cross-correlation of two 1-dimensional sequences.
cov(m[, y, rowvar, bias, ddof])	Estimates a covariance matrix, given data.

7 Context Management

Context-aware systems can be defined as systems that are able to adapt their operations to the current context conditions without explicit user intervention[3]. The context life cycle is composed of four phases, which are context acquisition, context modelling, context reasoning and context distribution. The essential part of context aware applications is the ability of the applications to infer the context based on sensor data and the context model as well as applying the necessary actions.
Different context models have been proposed and tested, such as key-value, markup, graphical, object oriented and ontology-based models. Among them, the latter is considered to have high and formal expressiveness and to allow the use of ontology reasoning techniques, although with lower performance.

IMPReSS aims to support inexperienced developers by providing a visual tool, named IoTLink, for creating context aware applications rapidly. Using IoTLink, developers could define the relationships among sensors, actuators, and their semantics in terms of application domain objects. IoTLink will then generate the necessary software artefacts consisting the representation of the physical domain objects. These software representations are linked to the corresponding sensors and actuators that are able to determine the context of the domain objects. This approach encapsulates the complexity of communicating with sensors and actuators and allows developers to interact with the domain objects directly as illustrated in Figure 12.

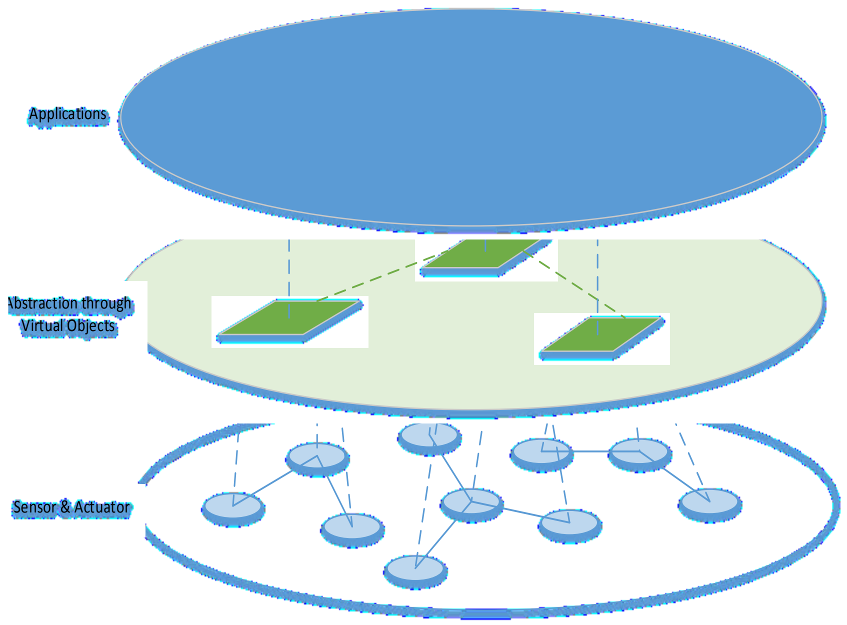


Figure 12. Abstracting sensors and actuators through domain objects.

7.1 IoTLink Implementation

There has been some efforts to define IoT Meta models which suggest how physical objects could be represented by software services e.g. Ebbits[14] an European research project aims at developing IoT platform for business applications, IoT-A, a European research project aim at standardizing IoT architecture. IoT-A has investigated the different IoT architectures and conclude them as a IoT Architecture Reference Model (ARM)[4]. Figure 2 illustrates a simplification of the IoT-A Meta-model showing that a physical object must be uniquely identifiable, has physical qualities that partly can be sensed by sensors, and has some capabilities or services that could affect the environment. Physical objects can be represented by Virtual objects, which act as their proxy allowing applications to retrieve their states and consume their services.

[14] www.ebbits.eu

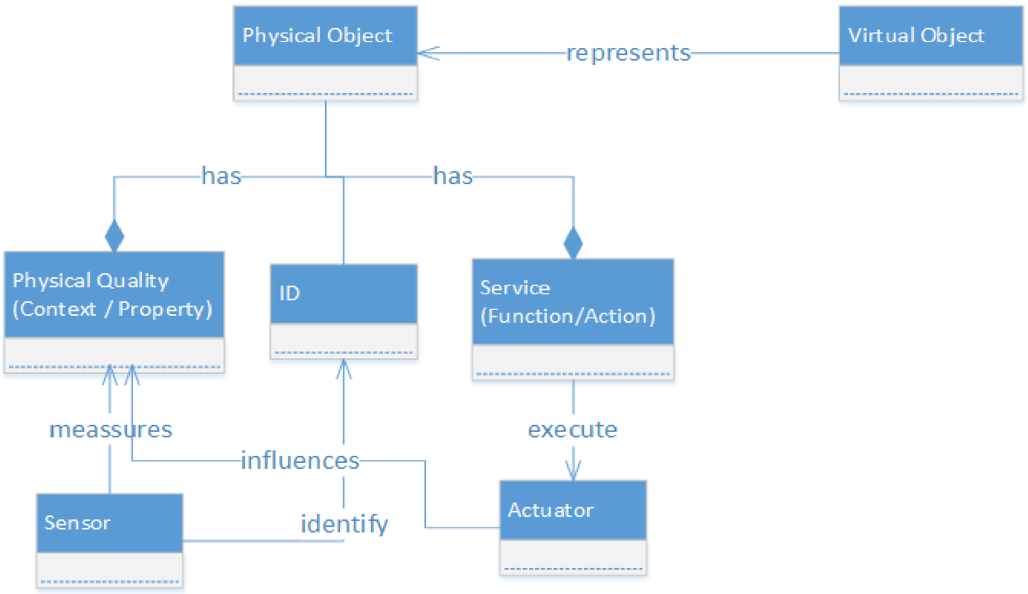


Figure 13. IoT Meta model based on the IoT-A project [5].

Based on this conception, we designed IoTLink that allows developers to compose software representation of physical objects through a model driven approach. The model contains a domain model representing the "Things" and their attributes. The attributes can be linked to data sources such as physical sensors, web services that are able to determine their actual states. Using this model, software artefacts can be generated and used as proxies for the "Things". Determining the actual states of the "Things" from sensor values may not work straightforward since sensor hardware has physical limitations and may contain measurement noise. Moreover, sometimes several type of sensors must be combined to be able to determine the state of physical objects. For instance to measure emotion of a user, several bio-readings such as respiration rate, heart rates, skin conductance may be collected and through intelligent algorithms, the system could conclude the stress level[6]. Thus, in within IoTLink, we need to introduce sensor fusion modules, which can be used to pre-process and fuse sensor readings before these values can represent the actual state of a physical object. As an example, when a power sensor is used to measure the energy consumption of a lamp, the sensor values must be calibrated depending on operating temperature. Thus, the sensor values are process by a sensor fusion module containing the calibration algorithm then the result can be assign as the actual energy consumption of the lamp.

A software representation of physical objects must provide an interface or a specific data format that can be accessed and processed by external applications. Therefore, we introduce output components that can be used to serialize the state of the physical objects through different data format and service protocols e.g. database entries, XML that can be accessed from a REST based service. Due to these considerations we define a Meta model (published in [7]) as a ground work for the development of IoTLink. The Meta-model consist of the main components that the users must define in their application model. First, the users must define Virtual Objects to represent "Things". Virtual objects may have properties and functions as well as classes to group them. The properties can be linked to connections, which represent the link to sensors. Several concrete connections are implemented and derived from the connections class. The link between connections and propertied could go through concrete sensor fusion components. The virtual objects can be serialized through output components. The output components also allow external applications to interact with the virtual objects i.e. by consuming their services.

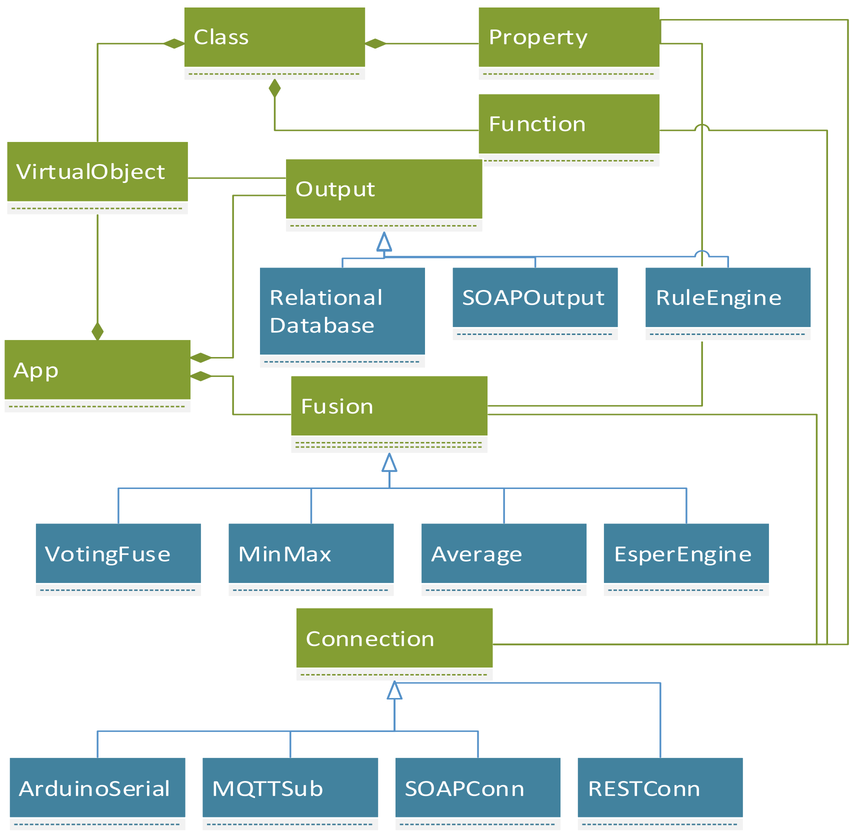


Figure 14. Logical view of the EMF meta-model

After the Meta model has been defined, IoTLink was developed by following a human centered approach. First, a low fidelity wireframe was developed using Balsamiq[15] and validated by 8 users using a cognitive walkthrough[8] approach to identify usability problems. Based on this initial feedback, we improved the user interface design and the Meta-model. After the wireframe design was quite mature, a high fidelity prototype of IoTLink was built and evaluated. We have published the result of this first iteration in[7].

We chose to implement the IoTLink as an eclipse plugin since Eclipse already offers many features required to support the productivity of a system development that are required for extending the generated code.

The IoTLink's development extensively explored the Eclipse Modeling Project[16], which already provide frameworks for developing a custom modeling language, a model transformation, and a code generator. After a careful investigation the following plugins were selected for developing IoTLink

- Eclipse Modeling Framework (EMF) to define the meta model of the modeling language
- Eclipse Graphical Modeling Framework (GMF) to create the graphical editor
- Extended Editing Framework (EEF) to create the property editor for the EMF elements
- Acceleo to create a model transformation from the EMF objects into Java code.

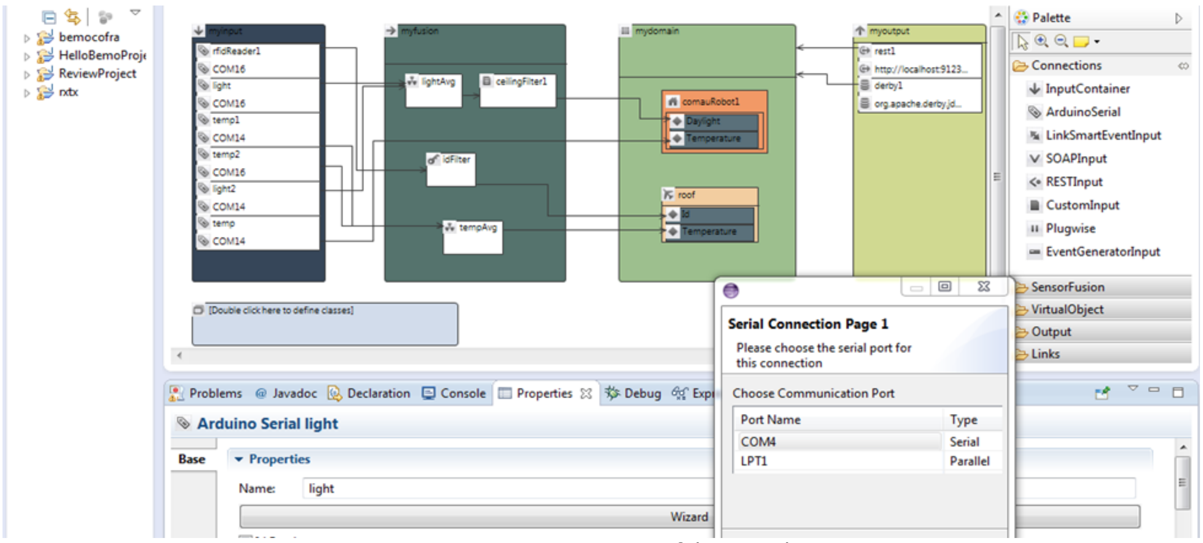
The Meta model is defined using a simplified UML called EMFCore (ECore) which is a standard model required by EMF[17]. Then, the ECore model is derived by the GMF to define the Graphical definition model, called "gmfgraph", which determines the visual elements to be displayed on the main anvas, the relationships and constrains between diagram, and the diagrams' behavior. Further, GMF creates a Tooling definition model, called "gmftool", which defines the notations to be displayed on the palette menu. The gmfgraph and gmftools are then mapped in a mapping configuration, called "gmfmap" which is used by GMF to decide on what notation should be shown on the main canvas when an item from the palette menu is dragged and dropped to the main canvas. To create a more visually attractive

[15] www.balsamiq.com
[16] www.eclipse.org/modeling/
[17] http://www.eclipse.org/modeling/emf/?project=emf

property sheet for each diagram, we use the EEF plugin. EEF derives the Meta model to generate an EEF model. An EEF model defines the widgets used in the property sheet of each notation.
As depicted in Figure 15, IoTLink consists of four main containers:

1. The connection container holds the components to communicate with sensors or other data sources.
2. The sensor fusion container holds the modules to process the sensor values before they are assigned to the properties of the virtual objects.
3. Virtual Object container contains the representation of "Things".
4. The output container defines how the representation of the "Things" can be accessed by external applications.

The grouping in the containers is aligned with the proposed Meta model shown in Figure 14.



Serial Connection Page 1

Please choose the serial port for this connection

Choose Communication Port

Port Name	Type
COM4	Serial
LPT1	Parallel

Name: light

Figure 15. Latest iteration of the IoTLink.

7.1.1 The connection components

The connection components are responsible for building connection to a data source, which could be physical devices, sensor networks, or software data source such as web services. Currently, we have implemented several components that are widely used for IoT prototyping such as:

- *ArduinoSerial* enables communication with Arduino (www.arduino.cc) boards. Arduino has been widely used for rapid hardware prototyping.
- *SOAPInput* enables connection to a SOAP based web service, which are widely used among various enterprise applications and recently has been proposed for devices (DPWS). The SOAPInput uses an XPath (www.w3.org/TR/xpath/) expression to parse the incoming soap objects.
- *RESTInput* provides a simple and lightweight alternative to SOAP based web service. RESTInput allows the users to poll a resource on a specific URL. It also uses XPath and JSONPath to parse the incoming XML and JSON respectively.
- *OPCClient* enables the communication to industrial devices through an OPC middleware, which is widely used in industrial environment. The OPCClient component can be configured to poll an OPC variable by providing tag of the variable.
- MQTTInput, this connection receives data from an MQTT broker[9]. MQTT is an emerging communication standard for IoT that adopts publish subscribe paradigm. MQTT features a small footprint and three level of QoS, which makes it ideal to run on devices with limited resources and unreliable network with low bandwidths.

7.1.2 Defining Complex Event Processing

For processing and combining sensor data, IoTLink includes a complex event processing (CEP) engine called Esper (esper.codehaus.org). It requires users to define how sensor streams should be processed by the engine in a query language called Event Processing Language (EPL). EPL Statements may consists of one or more views. Views may represent windows of stream as well as statistics of streams. For instance an EPL statement such as "win:length(5)" contains a view of sliding window with the length of

5. ESPER also allows aggregation and grouping using "group by" and "having" clause, which is useful to perform calculations of values based on particular group.

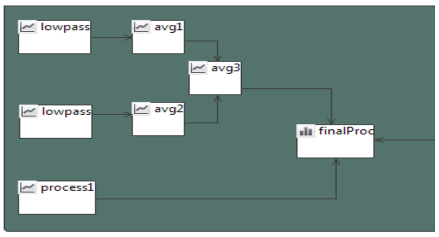


Figure 16. A network of sensor fusion modules

Sensor fusion modules can be combined as a network of processes when necessary as depicted in Figure 16. This allows data to be processed through a network of modular algorithms until the desired information is obtained.

7.1.3 Defining Virtual Object Component

In the virtual object container, developers are able to define the representation of the physical objects belonging to two different types. First, StaticObject represents stationary relation between physical objects and the sensors and actuators that observe them e.g. a room that has a temperature sensor attached on the wall of the room can be represented by a static object. Second, objects that only have temporary relations to the sensors e.g. occupants who move from one room to another can be observed by the sensors located nearby.
Similar to object oriented programming, each virtual object must have a class that defines its structure. The structure of a class includes properties and functions. Properties may have a type of primitive data types such as int, float, double, String, boolean, byte or a type of another class. The latter ones are called Complex Properties. These classes are defined in the TemplateContainer, which opens a separate diagram when the users double click on it. When the classes are defined, on the main canvas, the users could add concrete objects and assign a class to them. When a class is assigned, the structure of the class is applied to the object. This is useful for maintaining structural changes to a large amount of objects.
When a sensor is used to observe a specific property of a physical object, the developer can model this using IoTLink by linking the relevant input component to the property of the virtual object. This mapping is used by the code generator to route the values of the corresponding sensor to the object being observed. When several sensors are required to determine a specific property of a physical object, it can be modelled by linking the respective input components to a sensor fusion component, which then linked to the virtual object. The objects may also contain functions that can be mapped to actuators, which are used by the generated code to forward the function calls to the relevant actuators.

7.1.4 Output Components

The output components define how the virtual objects should be exposed to the external applications. We have implemented several components including storing the states of the objects to a relational database, exposing the objects as SOAP, or resources through REST, publishing the objects to MQTT broker, and sending the objects to a Drools rule engine. As these components work differently, each output results in different behavior for instance the SOAP based Web service provides a method to get different objects based on their classes. e.g. if there is a static object with a name of "object1" and has a type of "Class1", the output component will generate a Web Service method named getClass1(String Id). To retrieve object1, users could invoke getClass1("object1"). The REST component represents the virtual objects as web resources that can be retrieved through specific URLs. For instance, given an object with an id of "object1" and class of "Class1" and the application is run on the local host, the REST component generates the following URLs:

http://localhost/virtualobject/ class1/object1.

Moreover, the REST component generates parameterized URLs to invoke the functions of the virtual objects e.g.:

http://localhost/virtualobject/class1/object1?setOn =true.

The database component uses Eclipselink (www.eclipse.org /eclipselink/), an implementation of Java Persistence API (JPA) to interact with a database engine. The generated classes are annotated and automatically mapped to tables by Eclipselink. When the state of the object has changed, the snapshot is stored in the history table. The MQTTOutput component provides an event publisher to publish the state of the virtual objects through an MQTTevent broker [10]. The publisher could be configured to publish events with the two topic formats. First, a flat structure topic that only includes the class of the object, the object id, and the property as follows:

baseTopic/virtualobject/[ObjectClass]/[Obj.Id]/[PropName]

The topic structure allows developers to subscribe all events based on the class of the virtual objects using wildcard topic. The second format follows a hierarchical structure containing the objects id as shown by the following example:

baseTopic/virtualobject/[Obj.Id1]/[Obj.Id2]/…./[PropName]

Where the subsequent object is a child object of the prior object. The second topic pattern allows the application to subscribe to all events belong to an object and its children.

7.1.4.1 Drools rule engine

Context aware applications are required to perform an actuation based on the states of the physical objects. This can be implemented with different approaches such as hard coding the application logic using a programming language. To increase the flexibility of the applications, one could parameterize the variables that change frequently in the future. However, this approach is still not sufficiently flexible for extreme cases where the logic of the applications could change quite frequently. In context of research environment where various application logics must be evaluated, this happens very likely. E.g. to optimize energy consumptions in a building, several control strategies must be evaluated.

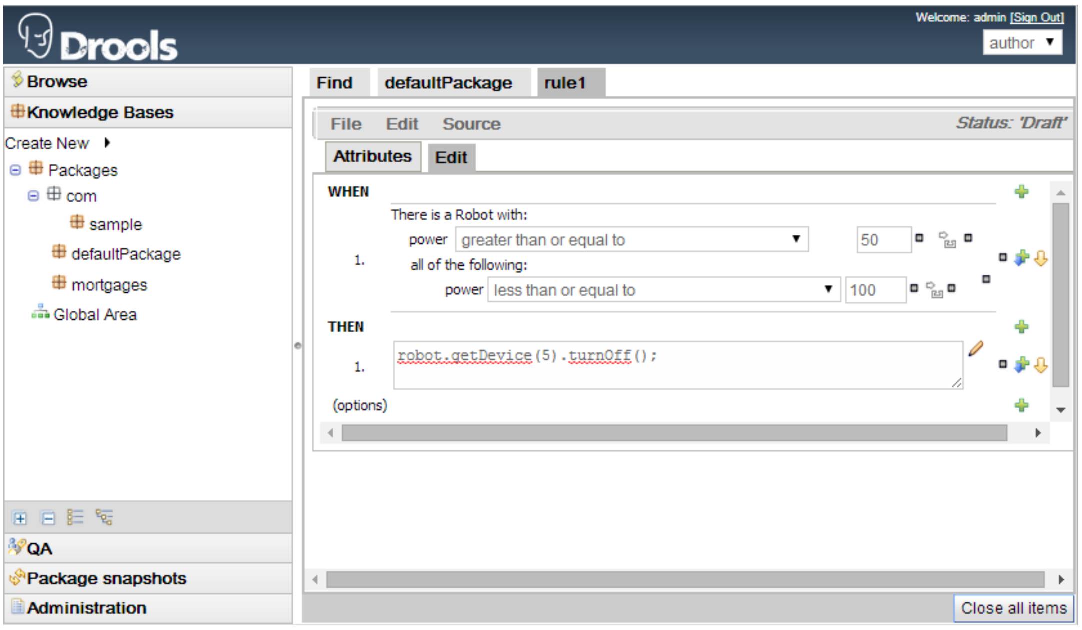
Rule engines are designed to decouple business logic from the rest of the applications enabling the business rules to change without even restarting the applications, which is suitable for addressing this requirement. Many rule engines are used in business environments. These engines are categorized based on how they execute the rules. The Forward Chaining engines execute consequence based on conditions expressed in the rule which implies "IF then ELSE" type of logic. The Backward Chaining engines, also called goal oriented, try to resolve the facts that fit a goal. Drools [11] is able to perform reasoning using both approaches and draw conclusion based on the facts and rules fed to the engine. Drools allows the rules to be defined in different language dialects. The native Drools dialect is called *mvel,* which follows a simple structure as depicted in Figure 17.

```
rule "name"
   attributes
   when
      LHS (conditions)
   then
      RHS (consequences)
end
```

Figure 17. Drools rule language format

Drools evaluate the rules when new facts are inserted to the engine or when the facts have changed. During the rules evaluation when the conditions of the rules are met, the consequence part of the rule is executed. If conditions of several rules are met, Drools apply a conflict resolution strategy by changing the order of executions based on the salience value of the rules. This requires developers to provide the priority of the rules when they define the rules.

The rules could be stored in a database and maintained using Drools Guvnor, which offers a web, based interface for defining and editing rules as depicted in Figure 18. The web-based interface is able to provide the users with a domain specific language (DSL) which can be tailored close to a natural language. However, a mapping between the phrases used in the DSL and the rule language must be provided by the developers.
The *DroolsOutput* component provided by IoTLink can be configure to retrieve the rules from a Guvnor Database and instantly apply any rule changes when the generated applications are still running. This allows different control strategy to be investigated during the prototyping phase. Enabling this feature, the *DroolsOutput* component must be configured with the URL of the Guvnor. When the Java code is generated, the code generator also generates a jar file containing the domain model of the application prototype that could be imported to the Guvnor to enable type safe feature when defining the rules.



WHEN
1. There is a Robot with:
power greater than or equal to 50
all of the following:
power less than or equal to 100
THEN
1. `robot.getDevice(5).turnOff();`
(options)

Figure 18. Defining rules responding to the state of the physical objects.

7.1.5 Generated Application

IoTLink is able to generate Java artefacts based on user-defined model. For each data source, sensor fusion and output component a Java class is generated. These classes are used by the controller class named MainApp, which initializes the concrete objects. The MainApp holds the link between domain objects, data sources, sensor fusion modules, and output. When data source objects receive data from physical objects, they are pushed to the sensor fusion modules, to which they are connected. The data could go through several levels of fusion depending on how the sensor fusion components are modeled. Once the sensor data is processed, it is pushed to the MainApp. If the sensor data does not need to be processed through sensor fusion modules, the data is pushed directly to the MainApp. Since the MainApp maintains the link between modules, it is able to assign these data to the corresponding virtual objects. When the virtual objects are updated, the output components are notified so that they can push the data if necessary e.g. the Database could persist the changes, MQTT broker could notify the subscribers, and the Drools could update the objects in its knowledge base. This is however not required by the output components that must be pulled e.g. SOAP- and RESTOutput.

8 References

1 Hornick, M.: ‘Java Specification Request 73: Java Data Mining (JDM)’, JSR-73 Expert Group, 2004
2 Brownlee, J.: ‘Clever algorithms: nature-inspired programming recipes’ (Jason Brownlee, 2011. 2011)
3 Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D.: ‘Context aware computing for the internet of things: A survey’, Communications Surveys & Tutorials, IEEE, 2014, 16, (1), pp. 414-454
4 Bauer, M., Bui, N., De Loof, J., Magerkurth, C., Nettsträter, A., Stefa, J., and Walewski, J.W.: ‘IoT Reference Model’: ‘Enabling Things to Talk’ (Springer, 2013), pp. 113-162
5 IoT-A: ‘Deliverable D1.3 – Updated reference model for IoT ’, in Editor (Ed.)^(Eds.): ‘Book Deliverable D1.3 – Updated reference model for IoT ’ (2012, v1.5 edn.), pp.
6 Haag, A., Goronzy, S., Schaich, P., and Williams, J.: ‘Emotion recognition using bio-sensors: First steps towards an automatic system’: ‘Affective dialogue systems’ (Springer, 2004), pp. 36-48
7 Pramudianto, F., Indra, I.R., and Jarke, M.: ‘Model Driven Development for Internet of Things Application Prototyping’, in Editor (Ed.)^(Eds.): ‘Book Model Driven Development for Internet of Things Application Prototyping’ (Knowledge Systems Institute Graduate School, 2013, edn.), pp.
8 Spencer, R.: ‘The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company’, in Editor (Ed.)^(Eds.): ‘Book The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company’ (ACM, 2000, edn.), pp. 353-359
9 Locke, D.: ‘MQ Telemetry Transport (MQTT) V3. 1 Protocol Specification’, IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html, 2010
10 Hunkeler, U., Truong, H.L., and Stanford-Clark, A.: ‘MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks’, in Editor (Ed.)^(Eds.): ‘Book MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks’ (IEEE, 2008, edn.), pp. 791-798
11 Bali, M.: ‘Drools JBoss Rules 5.0 Developer's Guide’ (Packt Publishing Ltd, 2009. 2009)