(FP7 614100)

D8.2 Application Architecture for Energy Management

Date 14 May 2014 – Version 1.3

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 version: 1.3 Submission date: 14 May 2014

Document control page

Document file: IMPRESS Deliverable D8.2 v1.3.odt
Document version: 1.3
Document owner: Djamel Sadok (UFPE)

Work package: WP8 – Platform Evaluation and Application Development
Task: T8.2 Redesign of The Application Architecture
Deliverable type: **R**

Document status: [x] approved by the document owner for internal review
[x] approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Lucas Lira Gomes (UFPE)	03/04/2014	First draft.
0.2	Djamel Sadok (UFPE)	21/04/2014	First revision.
0.3	Eduardo Souto (UFAM)	22/04/2014	Second revision.
0.4	Thiago Rocha (UFAM)	25/04/2014	Third revision.
0.5	Lucas Lira Gomes (UFPE)	30/04/2014	Addressed internal reviewers' concerns.
1.0	Lucas Lira Gomes (UFPE)	29/04/2014	Document sent for internal review.
1.1	Lucas Lira Gomes (UFPE)	30/04/2014	Addressed internal reviewers' concerns.
1.2	Thiago Rocha (UFAM)	08/05/2014	Fourth revision.
1.3	Lucas Lira Gomes (UFPE)	14/05/2014	Added details regarding physical devices used in the demo, their deployment, etc.

Internal review history:

Reviewed by	Date	Summary of comments
Ferry Pramudianto (FIT)	02/06/2014	Accepted
Peter Kool (CNET)	02/06/2014	Accepted

Index:

Table of Contents

1. Executive summary

This deliverable describes the initial proposal for the API of all modules which are part of the Impress Platform. These APIs will then be implemented and updated by other work packages, so that applications can request/provide data to the Impress Platform. Alternatively, we will refer to this set of APIs as the Impress Platform public API, since it will be exposed to external applications. In addition, it is important to mention that this initial specification is based on the current needs of the proposed application to be used in the final demo, described in D8.1. As the application, also described in D8.1, and Impress Platform's modules development progresses, this document should get updated in order to reflect all necessary changes in terms of actual implementation. To this extent, it is mandatory for every partner to maintain the others aware of any change in this specification, ideally by updating this document, so that the integration between the envisioned application in D8.1 and the Impress Platform can be carried out in a straightforward way.

Additionally, the architectural design of both the application described in D8.1 and the Impress Platform, will be discussed in more details, in terms of deployment, network and software integration for UFPE and Teatro Amazonas use cases.

To achieve that, we organise this deliverable according to the following. Firstly, section 2 contains additional information on the aim and scope of this deliverable. Section 3, gives a brief overview of the Impress Platform. In section 4, Impress Platform architecture will be discussed in terms of architectural decisions and how these fit the demo's energy use case without compromising the Impress Platform appeal for generic IoT scenarios. Section 5 covers the parameters that will be monitored & controlled by the Impress Platform, regarding physical devices and third-party systems that will be installed in UFPE. Also, topics such as types of monitored rooms, their estimated number, renewable energy sources usage, etc, will be mentioned. Section 6 provides a discussion about how the application in D8.1 is expected to work on the section 4 architectural premises. Section 7 contains the guiding principles that were adopted when specifying the public APIs, so that future updates to the public APIs can be consistent with prior versions. Section 8 contains a detailed specification of the Impress Platform public API. Finally, in Section 9, we provide our summary and conclusion.

2. Introduction

To allow third-party applications to be able to interact with the Impress Platform, hosted whether in a cloud service or locally, a public API specification is necessary to show how requests can be formatted and what to expect as the outcome of a given API call. In that sense, this deliverable aims to provide an initial specification of the Impress Platform public API and to characterize how the API is related to both the application and the Impress Platform architectural design. This specification is guided by the current needs of the application, described in deliverable 8.1. However, this public API specification is yet not sufficiently extensive, as the architecture of both the application and the Impress Platform evolves, extra features that were not forethought can turn to be indispensable or requirements can change to the point of making certain APIs' calls useless or redundant. Thus, these modifications also need documenting. Ideally, this deliverable must evolve to ensure that partners keep track of how they can use the Impress Platform on their external applications.

Additionally, this constant effort to update this specification is highly important and should be a priority for any work package that finds out that modifications in this specification are required. Work package 8, in charge of the application development, will use this document as its primary reference to guide the interaction with the Impress Platform's modules. Divergences between this document and the real implementation of the Impress Platform public API can slow the integration with the application and must, therefore, be avoided.

Finally, minor architectural redesigns were proposed for both the application described in D8.1 and the Impress Platform. This was a result of our better understanding of the implications of the Impress Platform architecture's former vision, proposed in the DOW, when it comes to deployment, network and software integration. Despite that, we also address how to guarantee the extensibility of the Impress Platform, when it comes to interfacing new hardware besides our original scenario with energy-oriented resources (i.e. distributed energy resources, energy storage, energy metering and other sensors and actuators). Thus, permitting Impress Platform to still have appeal for non-energy scenarios.

3. Impress Platform Overview

The IMPRESS platform can be seen as a software platform in the cloud designed to manage ubiquitous sensor data. It was engineered to provide a set of tools to help application developers to build and manage connected smart devices and applications based on connected things.

Figure 1 provides a complete overview of the IMPRESS platform several modules.

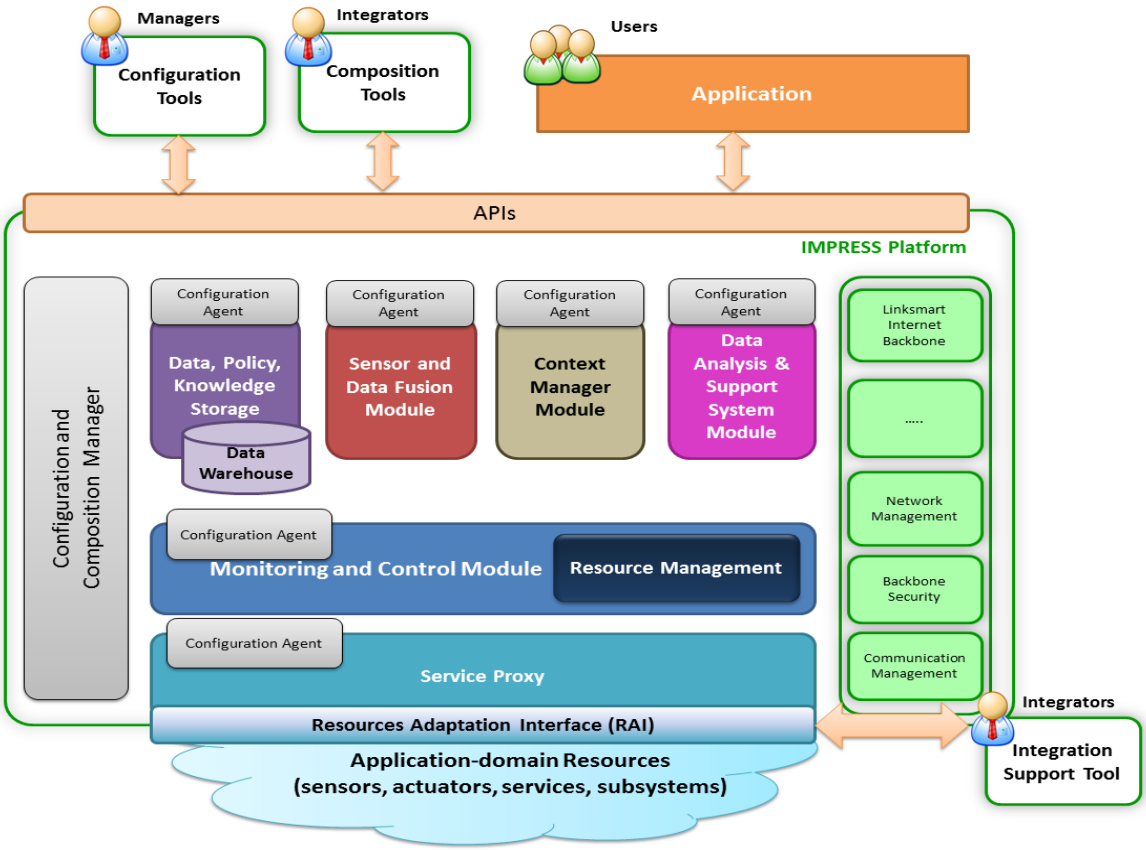

Figure 1: IMPRESS Platform Overview.

For every module within the Impress platform, an API should be provided for an external application to access it. This set of public APIs are depicted as a single block called 'APIs' in Figure 1, but this does not have to be a single software component. Ideally, every module will implement their own API as a part of the module itself, with that the application developers know the endpoint of every module. Impress platform's modules can be used as black boxes via the public API, hiding all the complexities of the provided features from application developers.

However, to be consistent, all Impress modules should use the same type of architectural model for their APIs. Requiring application developers to use SOAP [6] for some modules, for instance, and REST for others, would not be a good approach in terms of consistency. Hence, we endorse the usage of Representational State Transfer (REST). REST is a term coined by Roy Fielding in his Ph.D. dissertation [1] to describe an architecture style of networked systems, being a defacto standard for web services in general.

Additionally, REST architectures are simple, as the whole concept is based on the core of the web, namely the HTTP protocol and its concepts of Resources and URIs. Despite that, there is a wide set of mature frameworks for developing REST solutions in most mainstream programming languages.

Note that for internal communication, among Impress Platform modules, some of these REST APIs aimed for external applications can turn to be useful. If not, modules can also use LinkSmart Middleware, formerly known as Hydra Middleware [8], to share data internally. These two approaches alone, a REST API and LinkSmart Middleware, are sufficient to address Impress Platform needs in terms of distributed software integration for the scope of this project.

4. Impress Platform Architecture

A preliminary version of the Impress Platform architecture was shown in Figure 1 and it will be redesigned as part of this task. The architecture, depicted there, shows an example of exploitation of the development platform for general IoT applications. In our energy scenario for the demo, this architecture is still suitable as, for instance, energy meters can be abstracted as sensors and energy sources can be controlled via actuators. In fact, changes, in terms of software, due to different scenarios is just a matter of implementing proxies, developed as part of WP3, for interfacing with energy-oriented resources (i.e. distributed energy resources, energy storage, energy metering and other sensors and actuators) via the Resource Adaptation Interface (RAI) depicted in Figure 1. In order to add support for other devices (i.e. meters, actuators, etc) in the Impress Platform, a developer would just have to implement a new proxy.

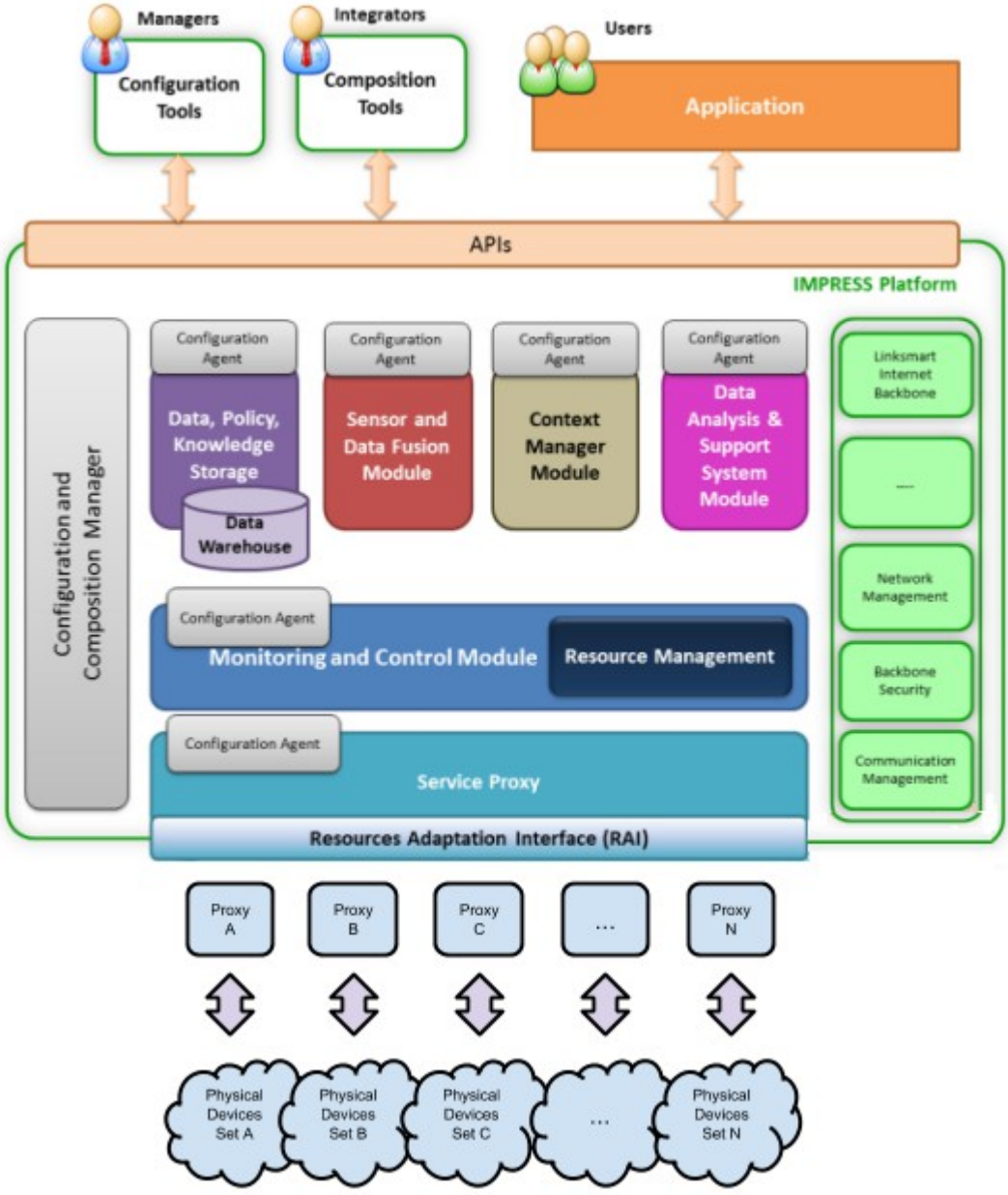

Figure 2: IMPRESS Platform Architecture.

As a result of that, the Impress Platform maintains a modular design, avoiding embedding scenario specific logic in their core modules. This design decision enforces reusability; as proxies, depicted in Figure 2, can be shared and used in a plug-and-play manner; allows extensibility; since new proxies can be developed independently as components outside the Impress Platform; therefore permitting heterogeneous devices to cooperate in a transparent way.

The only requirement, however, is for these proxies to be able to communicate with the Service Proxy in the Impress Platform, via the Resource Adaptation Interface (RAI) API. So that the Impress Platform can abstract its internal semantics and business logic from devices' technical details. In this case, proxies main purpose would be just translating the Resource Adaptation Interface (RAI) messages into specific actions for these devices (e.g. turn on/off an appliance, etc) and formatting data gathered from devices, in order to send it to the Impress Platform. This sense and actuate cycle is generic enough to permit a broad set of use cases in terms of supporting third parties systems.

Monitoring and control modules, leveraging the work done in WP4, should not be changed also to support a given scenario, as they are part of the core modules of the Impress Platform. The key point here is that the Service Proxy module, which is part of WP3, should be able to provide resources meta-data that are generic enough for the Impress Platform to treat different devices, interfacing Impress Platform via the Resource Adaptation Interface (RAI), as generic sensors and actuators, despite hardware vendors' specific details. For data fusion and data analysis modules, likewise, nothing should change. Algorithms in these areas, whether for prediction or classification tasks, are well established and should be available regardless the scenario, depending on the applications developers to understand what algorithm fits better their use case in lieu of defining a one-size-fits-all model for carrying out prediction and classification tasks in the energy scenario.

5. UFPE's Demo Architecture

Aspects regarding physical devices and third-party systems deployment at UFPE will be covered in this section, so that partners can get a grasp of how to proper conceive their modules to facilitate software/hardware integration.

5.1 UFPE's Monitoring Scope

The Federal University of Pernambuco (UFPE) is a public university, established in 1946. The university has three campuses: Recife, Vitória de Santo Antão and Caruaru. For the scope of this project, only UFPE's main campus in Recife, which has 10 departments in 149 hectares, will be considered.

Despite that, it is important to mention that GPRT, which is a two-floor building occupying a total of 380 m² within UFPE, will be part of the demo. GPRT's floor plans are depicted in both Figure 3 and 4, one for each floor. Additionally, one department, among the ten in Recife's Campus, is envisaged to take part into the demo too, but there was no formal request yet, to those in charge of those departments, so that experiments with Impress Platform could be extended to them. This issue, however, will be addressed in the subsequent months. Note that, due to Recife's campus extensive area, it would not be sensible to propose more than one department, since the deployment and hardware cost would be too high.

Despite that, GPRT entails thirteen ambients, from meeting rooms to offices. All those populated mainly with computers (e.g. PCs, notebooks, servers, etc) and individual split air conditioners. Other appliances including printers, a microwave, a compact refrigerator, TVs, networking devices (i.e. routers, switches, etc), are also present.

As for other departments, they are mainly composed of classrooms, libraries, bathrooms, teachers' rooms, meeting rooms and computer labs. In terms of electronic equipments, there are HVAC (heating, ventilation, and air conditioning) systems, computers (e.g. PCs, notebooks, servers, etc), TVs, video projectors, printers, networking devices (i.e. routers, switches, etc), just to name a few. Therefore, in essence, there is not much difference from GPRT to other UFPE's departments, in terms of electronics that could be monitored and controlled by the Impress Platform.

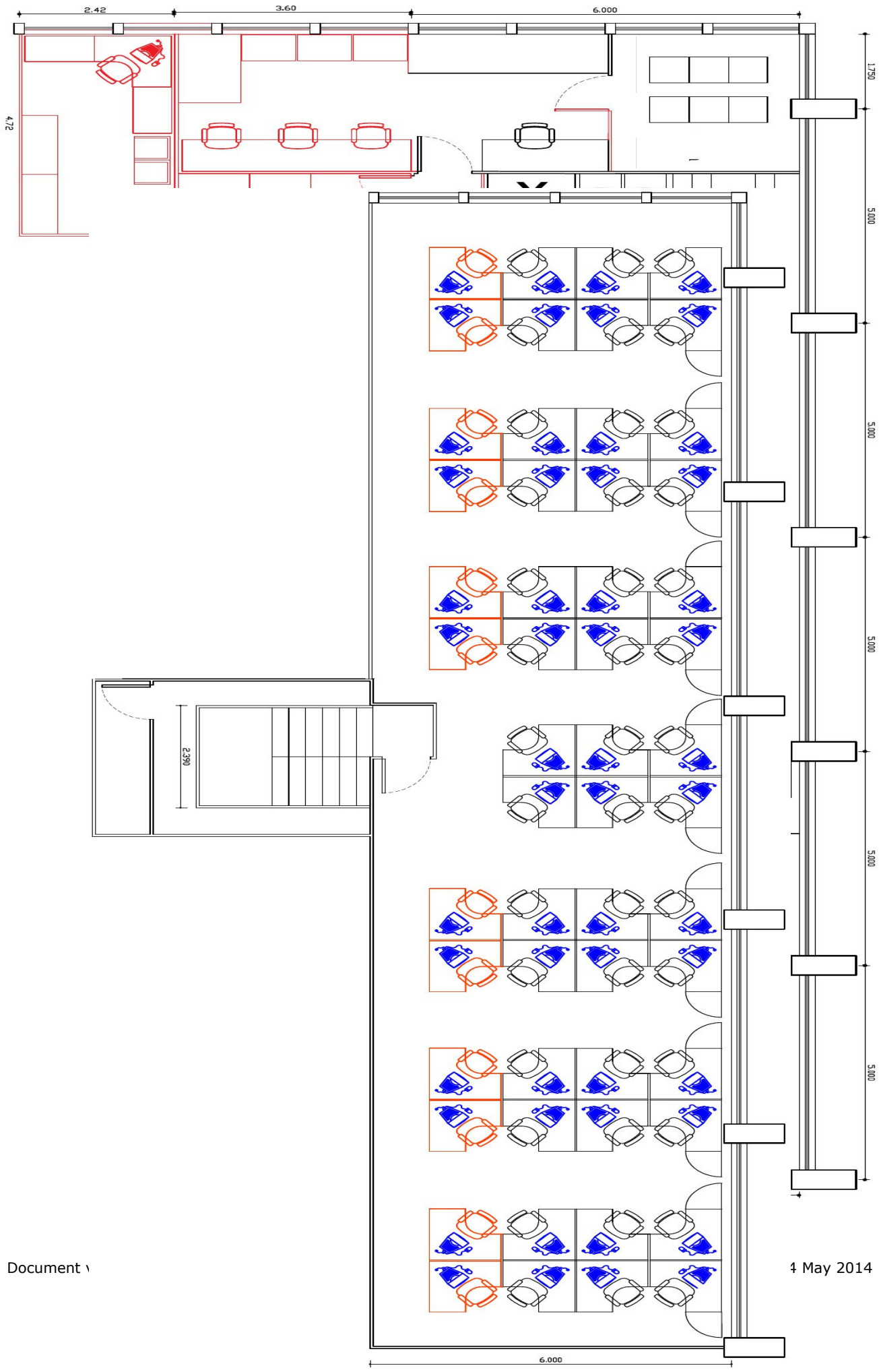
2.42
3.60
6.000
4.72
1750
5.000
5.000
5.000
5.000
5.000
2.390
6.000

Figure 3: GPRT's ground floor floor plan.

Figure 4: GPRT's first floor floor plan.

5.2 Monitored & Controlled Parameters

For achieving the scenario based descriptions described in D8.1, only energy consumption have to be monitored. Likewise, Impress Platform should be also able to turn on/off devices remotely. Despite that, as other applications can be made for performing if-this-than-that (IFTT) activities, as described in next section, the following parameters were judged to be also useful and will, therefore, be provided:

1. Temperature
2. Light Intensity
3. Humidity
4. Presence

5.3 Monitored & Controlled Devices

The main point is to have smart energy sockets that are able to interrupt and measure remotely energy consumption of appliances connected to them. Similarly, smart energy switches would provide the same functionality, but for lamps. Therefore, ideally, all energy socket and light switch within GPRT will be replaced by a smart equivalent. In this sense, there are in-wall and external plug/extension cord solutions. In-wall solutions are more intrusive, as current energy sockets will have to be replaced by them but, on the other hand, they are more aesthetically pleasing as there is no apparent difference from normal energy sockets. As for external plugs/extension cord, they are more noticeable, but have the advantage of not requiring any infrastructural change. For UFPE's demo, we endorse the use of external plugs, as it will be easier to prototype and deploy the demo. Regarding light switches, there is no option though, as light switches cannot be plugged like in the energy socket case.

In terms of external plugs solutions, there are several alternatives available for purchasing, like Circle's family from Plugwise [9], WeMo Insight Switch from Belkin [10], Xbee Smart Plug from Digi [11], DSP-W215 Wi-Fi Smart Plug from D-Link [12], SP-1101WSmart Plug Switch from Edimax [13], SmartPower Outlet from SmartThings, etc. Most of them, however, does not provide an API for third-part applications, like a proxy or so, to monitor and control them. Therefore, solutions that provide software development kits (SDK) to interoperate with their systems, are a must for our use case, as reverse engineering proprietary application protocols is out of the question. If purchasing ready-to-use smart energy sockets and light switches turns out to be the best option, which of them, however, is an open issue that will be addressed in the following months.

Those smart devices will then feed Impress Platform with periodic reports, allowing Impress Platform to provide functionalities as cited in sections 4 and 6. In order to provide extra measurements (e.g. temperature, light intensity, humidity and presence), like mentioned in subsection 5.2, WSAN motes with such capabilities will be deployed in a per room manner. In this sense, a General Sensors Proxy, as depicted in Figure 5, is needed to forward WSAN Motes messages to the Impress cloud, via the Resources Adaptation Interface (RAI), and vice versa. Likewise, there will be also a proxy for smart energy sockets and light switches, named Energy Sockets Proxy and Light Switches Proxy respectively.

For other department, a smaller area for experimentation may have to be specified due to the typical size of departments, as the cost of replacing light switches entirely and purchasing external energy socket plugs can be prohibitive.

This approach, of having smart energy sockets/light switches that are able to interrupt and measure remotely the energy consumption of appliances connected to them, would permit a real-world usage of the Impress Platform in terms of energy efficiency automation.

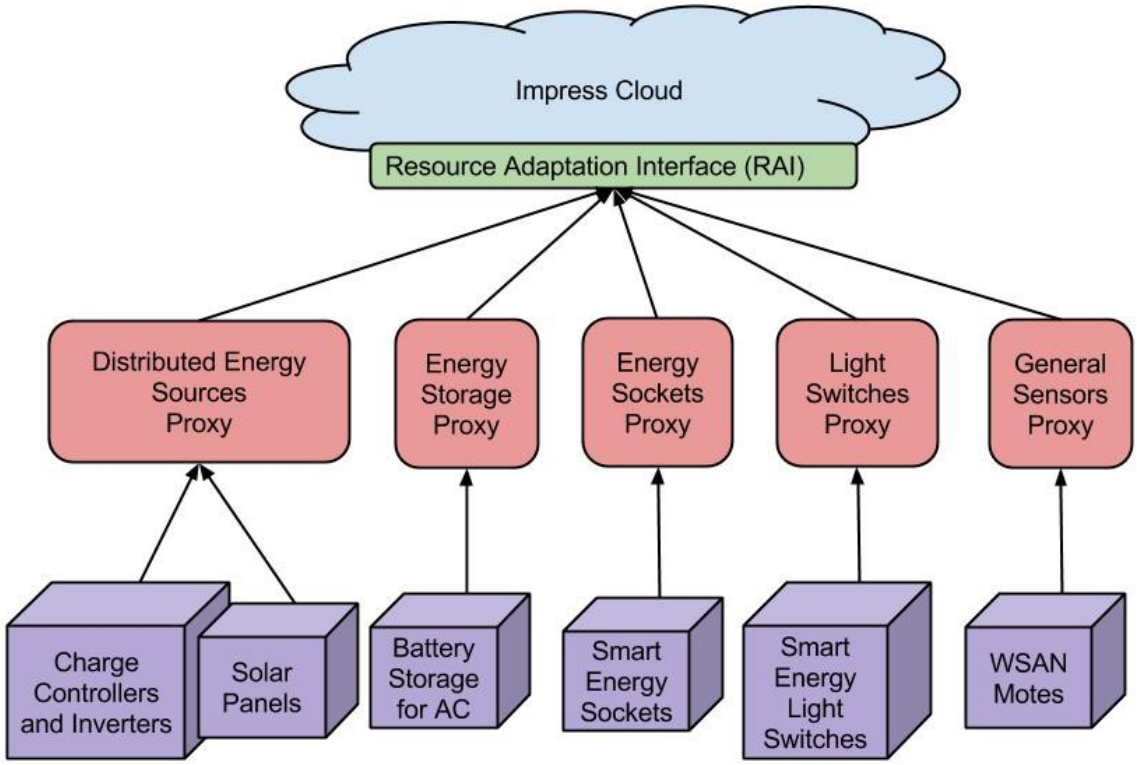

Figure 5: Physical devices integration with the Impress Platform via the Resource Adaptation Interface (RAI).

Additionally, some technical complexities will have to be addressed due to the flexibility of Impress Platform, as there is no way to know in advance which appliance is plugged in which smart energy socket. Therefore, for electronics that are usually deployed in a static manner (e.g. PCs, refrigerators, TVs, etc), Impress Platform should provide some way to map an energy socket with a given appliance in a logical level. This logical mapping, however, is not suitable for every usage, as is common to have spare energy sockets for alternating between devices such as chargers, research equipment and anything that does not stays plugged to an energy socket at all times.

5.4 Renewable Energy Considerations

For this aspect, solar energy panels will be installed at GPRT. Therefore, Impress Platform should be able to monitor and control energy sources too. Consequently, charge controllers and inverters will have to be interfaced by a proxy as stated in section 4, which in this case is the Distributed Energy Source Proxy depicted in Figure 5. Also, a battery storage system, for alternating current (AC), will be deployed and integrated with a charge controller and inverter, therefore providing a backup source in the face of energy outages.

6. Impress Application Architecture

During the application development a software architectural pattern called MVC (Model View Controller) will be used. MVC divides a given software application into three interconnected parts and it isolates business logic from the user interface. Using MVC the Model represents the information of the application and the business rules used to manipulate the data, the View corresponds to elements of the user interface and the Controller manages the communication between the Model and the View. In the case of the application that will be developed in this WP, the Model part will be responsible to communicate with the Impress Platform modules through a public REST API.

A public API has to be provided to allow the application developers to create their requests to the Impress Platform, for more on these technicalities, refer to sections 6 and 7. Despite that, this public API, as mentioned in Section 3, does not need to be implemented as a single software component. All Impress Platform modules can be running in the same machine though, but that is not mandatory. In the case of deploying Impress Platform modules in different machines, however, the endpoints of every module should be also publicly available. Otherwise, application developers would not know where to direct a given API call that is described as part of the public API documentation. So given that application developers know the endpoint of every module and the Impress Platform's modules can be used as black boxes via the public API, as depicted in Figure 6, hiding all the complexities of the provided features from application developers.

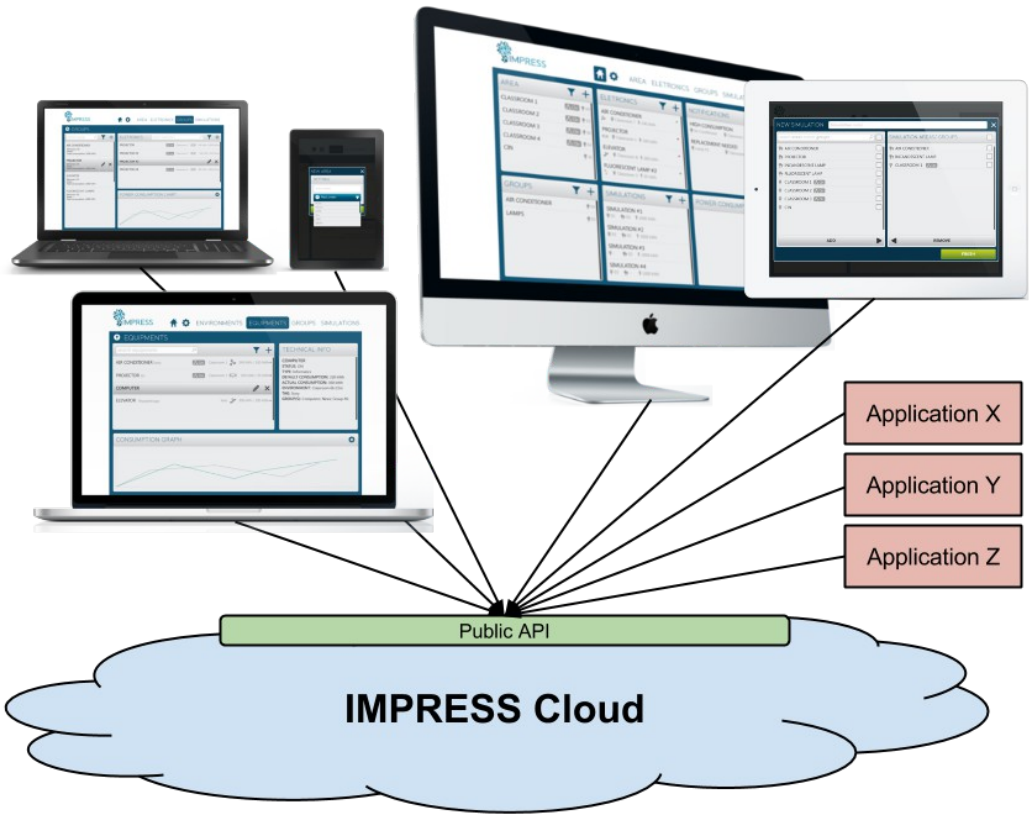

Figure 6: Applications interfacing IMPRESS Platform via a Public API.

In practical terms, several applications would be accessing Impress public API to get data from Impress Platform modules and configure them, via their Configuration Agents depicted in both Figure 1 and 2. Note that there is no need to develop a single application that uses all Impress features. In fact, as can be seen in D8.1, the proposed application in it is focused in the data analysis aspect of the energy scenario, being handy for understanding how different appliances, whether currently in usage or possible options for purchasing, can impact the overall energy consumption in the long run.

Obviously, more control-oriented applications, external to the proposed application in D8.1, will be running in background enforcing, for instance, expected behaviours, like turning off light bulbs from a given room if nobody is present or automatically deciding when to turn on or off exterior lamps, given the timezone of the deployed system. Static use cases like these, which are expected in terms of automation, should not have to be configured again every time Impress Platform is adopted. Especially because they are predictable for most scenarios, probably with little or no adjustments at all.

One option, would be to embed most common use cases in Impress Platform, but that would again tie Impress Platform to specific scenarios. The best alternative, in the sense of keeping Impress Platform both extensible and separating concerns properly, would be for external applications to set these expected behaviours independently, with the help of the IDE Framework for Model-driven development which will be implemented as part of WP7. The IDE is particularly useful when it comes to if-this-than-that (IFTT) application logic, which is the case of the kind of expected behaviour that we cited here. That, however, does not solve the problem of new adopters of the Impress Platform having to configure all automatic behaviour again. So, in order to tackle this aspect, common use cases implemented as model-driven specifications could be packaged and provided to the public, that could than choose what to use by their own and, if one of these packages does not suit a specific use case, tweak them to do so independently.

7. Guiding Principles for the Impress Platform Public API

7.1 API Security Policy

All API access intended to be exposed for applications must be over HTTPS. Whether Impress is a public or private cloud, security still a concern to be taken into account, so we endorse HTTPS usage for Impress Platform public API. HTTPS is a URI scheme which has identical syntax to the standard HTTP scheme. However, HTTPS enforces the use of an added encryption layer of SSL/TLS, therefore creating a secure channel over an insecure network. This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks.

7.2 Schema

All data must be sent and received as JSON [2], blank fields should be included as "NULL" instead of being omitted and timestamp, which must be in UTC and represented in the ISO 8601 (i.e. YYYY-MM-DDTHH:MM:SSZ) format.

7.3 URIs

The URIs by which resources are named are an important part of a REST API, since naming properly resources can make an API intuitive and easy to use [3]. One of the most important conventions in the REST architectural model is that URIs should represent nouns, not verbs. There are two reasons for this, the first one is that the actions that can be performed are already defined by the HTTP verbs. For instance, a GET request can be used to retrieve resources. Second, since a REST call requires a request based on the standard HTTP's methods, it only makes sense that requests are performed against a noun as opposed to another verb, for example, consider the following URL:

GET */MyRestService/getDevices*

This URI is saying to get the getDevices resource and this is clearly redundant. A better way to express this would be through a URI like this:

GET */MyRestService/devices*

Another convention is to use plural nouns for collections and perform actions on them, for example consider the following URL:

GET */MyRestService/devices*

Return a list of all devices.

GET */MyRestService/devices/20*

Returns a single device with id 20.

The last convention is that URIs for REST services should grow hierarchically, for example:

GET */prediction/:device_id/measurements/:measurement_variable_id*

This way the user first pass the device_id followed by a measurement_variable_id, making the REST API more clear and easier to use.

7.4 Client Errors

There are three types of client errors on Impress Platform public API calls:

1. Sending invalid JSON [2] will result in a 400 Bad Request response.

Example:

HTTP/1.1 400 Bad Request
Content-Length: 35

{"message":"Problems parsing JSON."}

2. Sending the wrong type of JSON [2] values will result in a 400 Bad Request response.

Example:

HTTP/1.1 400 Bad Request
Content-Length: 40

{"message":"Body should be in JSON format."}

3. Sending invalid fields will result in a 422 Unprocessable Entity response.

Example:

HTTP/1.1 422 Unprocessable Entity
Content-Length: 149

```
{
  "message": "Validation Failed",
  "errors": [
    {
      "resource": "device",
      "field": "title",
      "code": "missing_field"
    }
  ]
}
```

All error objects have resource and field properties so that your client can tell what the problem is. There's also an error code to let you know what is wrong with the field. These are the possible validation error codes:

Error Name	Description
missing_resource	This means a resource does not exist.
missing_field	This means a required field on a resource has not been set.
invalid_format	This means the formatting of a field is invalid.

7.5 HTTP Verbs

APIs should make an appropriate usage of the HTTP verbs.

Verb	Description
HEAD	Can be issued against any resource to get just the HTTP header info.
GET	Used for retrieving resources.
POST	Used for creating resources.
PATCH	Used for updating resources with partial JSON [2] data. A PATCH request may accept one or more of the attributes to update the resource. PATCH is a relatively new and uncommon HTTP verb, so resource endpoints also accept POST requests.
PUT	Used for replacing resources or collections. For PUT requests with no body attribute, be sure to set the Content-Length header to zero.
DELETE	Used for deleting resources.

8. REST API Documentation

This topic specifies a RESTful API for creating and managing IoT resources, including storage, data analysis, and networking components.

This specification for the Impress Platform public API includes:

- Common behaviors that apply across all requests and responses.
- Resource models, which describe the JSON [2] data structures used in requests and responses.
- The requests that may be sent to cloud resources, and the responses expected.

This API operates on the following endpoints or resources, one for each of the modules in the Impress Platform:

- Context Manager
- Data Analysis & Support System
- Data, Policy, Knowledge Storage
- Monitoring and Control
- Sensor and Data Fusion
- Service Proxy

8.1 Context Manager

This module will not be used directly by the application proposed in deliverable 8.1, therefore it is not clear right now what interfaces should be exposed by it. To that extent, the Impress IDE that will be developed, in WP7, is a better candidate to specify a possible API for this module as the model-driven development methodology, employed in WP7's IDE, will be able to create applications using this module.

8.2 Data Analysis & Support System

8.2.1 Predict Device's Measurement Variables

GET */prediction/:device_id/measurements/:measurement_variable_id*
Return all predicted data points for a :measurement_variable_id of a given :device_id.

8.2.2 Parameters

Name	Type	Description
numberOfDataPoints	integer	The number of data points provided for the given time range.
upTo	string	This specifies the time range used for prediction, from the current time up to the timestamp specified. This is a timestamp in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ.

8.2.3 Response

GET */prediction/20/measurements/5?*numberOfDataPoints=10&upTo=2014-09-10T20:30:10Z

```
Status: 200 OK
{
  deviceId: 20
  currentTime: 2014-09-10T14:30:10Z
  upTo:  2014-09-10T20:30:10Z
  numberOfPredictedDataPoints: 10
  predictedDataPoints : [45,58,62,46,55,58,38,99,44,65]
}
```

8.3 Data, Policy, Knowledge Storage

As described in deliverable 5.1, the Data, Policy, Knowledge Storage module will be interfaced via a Rexter server, which is able to to execute graph-based database operations in graph-based NoSQL Databases, via Gremlim [7] queries. Or, in other words, 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 change to those that interface with the Data, Policy and Knowledge Storage module. Rexter provides two alternatives to query/generate graphs in graph-based NoSQL Databases: a REST interface [3] and a binary protocol called RexPro [4]. Among those, we endorse RexPro [4] usage, since it is more lightweight than HTTP calls via Rexter REST [3] interface. For more info on this module and the technologies involved, please refer to deliverable 5.1.

However, it worth to mention that it is not necessary to hardcode neither Rexter REST nor RexPro requests, in your source code, when willing to interface the Data, Policy, Knowledge Storage module. Rexter uses a standard called Blueprints, popular among graph-based NoSQL databases. Therefore, there are already compliant implementations in several programming languages, despite Java's official support. Please refer to [5] for alternative libraries in Python, Ruby, .NET/C#, etc.

8.4 Monitoring and Control

8.4.1 Discovering all devices' states

GET *resources/*
Return a list of all devices' states.

8.4.2 Response

GET *resources/*

```
Status: 200 OK
[
  {
    deviceId: 20
    deviceIsOn: True
  },
  {
    deviceId: 10
    deviceIsOn: False
  },
  {
```

```
    deviceId: 55
    deviceIsOn: True
  },
]
```

8.4.3 Discovering a single device's state

GET *resources/:device_id/*
Return a given :device_id state.

8.4.4 Response

GET *control/20/*

```
Status: 200 OK
{
  deviceId: 20
  deviceIsOn: True
}
```

8.4.5 Editing a single device's state

PATCH *resources/:device_id/*
Edit a given :device_id state.

8.4.6 Parameters

Name	Type	Description
deviceIsOn	boolean	Whether the device is switched on or off.

8.4.7 Response

PATCH *resources/20/*

```
{
  "deviceIsOn": True
}

Status: 200 OK
{
  deviceId: 20
  deviceIsOn: True
}
```

8.5 Sensor and Data Fusion

This module will not be used directly by the application proposed in deliverable 8.1, therefore it is not clear right now what interfaces should be exposed by it. To that extent, the Impress IDE that will be developed, in WP7, is a better candidate to specify a possible API for this module as the model-driven development methodology, employed in WP7's IDE, will be able to create applications using this module.

8.6 Service Proxy

This module will not be used directly by the application proposed in deliverable 8.1, therefore it is not clear right now what interfaces should be exposed by it. To that extent, the Monitoring and Control module that will be developed, in WP3 and WP4, is a better candidate to specify a possible API for this module as the model-driven development methodology, employed in WP7's IDE, will be able to create applications using this module.

9. Conclusions

In this deliverable, we presented an initial specification for the Impress Platform public API. This API will be used by applications to interface Impress Platform modules' services. Thus, being essential for the application proposed in deliverable 8.1 that will leverage on Impress Platform reusable modules to provide energy management and data analytics capabilities to end-users.

This initial specification entails two parts: the Impress Platform public API documentation, which is a Restful API, and a set of guidelines. These guidelines aim to facilitate extending this document's APIs, as requirements change, without abdicating proper consistency in terms of HTTP verbs usage, error handling, schema, and security policies among Impress Platform modules' individual implementations.

Consistency, in this sense, is important since it makes the whole API more predictable, for developers implementing applications. The resulting middleware is expected to be easy to interface, therefore possibly leading to a higher adoption rate of the Impress Platform, as hard to use technologies tend to fade to oblivion in the long run regardless of their power.

Additionally, aspects regarding physical devices and third-party systems deployment at UFPE were covered, so that partners can get a grasp of how to proper conceive their modules to facilitate software/hardware integration.

Finally, minor changes in the architecture were proposed, in order to guarantee that Impress Platform will remain appealing for non-energy scenarios, by preventing to embed concerns from specific scenarios in the core of the Impress Platform.

10. References

[1] Fielding, Roy Thomas. *Architectural styles and the design of network-based software architectures*. Diss. University of California, 2000.

[2] JSON. http://www.json.org/

[3] Rexter REST API Documentation. http://github.com/tinkerpop/rexster/wiki/Basic-REST-API

[4] Rexter RexPro API Documentation. http://github.com/tinkerpop/rexster/wiki/RexPro

[5] Rexter Documentation. http://github.com/tinkerpop/rexster/wiki

[6] SOAP. http://www.w3.org/TR/soap/

[7] Gremlin – Graph Database Language. http://github.com/tinkerpop/gremlin

[8] Hydra Middleware. http://www.hydramiddleware.eu/news.php

[9] Plugwise's Circle. http://www.plugwise.com/products/#filter=meters-switching

[10] Belkin's WeMo Insight Switch. http://www.belkin.com/us/F7C029-Belkin/p/P-F7C029/

[11] Digi's Xbee Smart Plug. http://www.digi.com/products/wireless-modems-peripherals/wireless-range-extenders-peripherals/xbee-smart-plug-zb

[12] D-Link's DSP-W215 Wi-Fi Smart Plug. http://www.dlink.com/us/en/home-solutions/connected-home/smart-plugs/dsp-w215

[13] Edimax's SP-1101WSmart Plug Switch. http://www.edimax.com/en/produce_detail.php?pd_id=517&pl1_id=33&pl2_id=154

[14] SmartThings' SmartPower Outlet. https://shop.smartthings.com/#/products/smartpower-outlet