(FP7 614100)

D2.2.2 SDP Final Architecture Report

Date: 07 January 2016 – Version 1.0

Published by the Impress Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2013.10.2 EU-Brazil research and development Cooperation
Target Outcome: b) Sustainable technologies for a Smarter Society

Document control page

Document file: d2.2.2_sdp_final_architecture_report_v1.doc
Document version: 1.0
Document owner: Carlos Kamienski (UFABC)

Work package: WP2 – Requirements Engineering and SDP Architecture
Task: Task 2.3 SDP Architecture
Deliverable type: R (Report)

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	Carlos Kamienski (UFABC)	21/12/2015	First draft ready
0.3	Carlos Kamienski (UFABC)	23/12/2015	Second draft ready

Internal review history:

Reviewed by	Date	Summary of comments
Jussi Kiljander	07/01/2016	Approved with minor comments
Marc Jentsch	05/01/2016	Approved with minor comments

Index:

1. Executive summary

IMPReSS is an EU-Brazil cooperation project aiming at providing a System 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 CO^2 footprint in public buildings, enhancing the intelligence of monitoring and control systems as well as stimulating user energy awareness.

The architecture presented here is the second version of the IMPReSS Software Architecture, used as the reference for building IMPReSS applications. 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' awareness of the IMPReSS platform capabilities so that they can always refer to it 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.

This report covers functional as well as non-functional aspects that are important to support the integration of different tasks involved in this project. The design process of the architecture has been influenced by three key elements: a) the initial version of the architecture, presented in (Kamienski et. al 2014); b) the detailed specification of the individual components; and c) the way the IMPReSS Architecture has been used for developing application for particular scenarios.

Software Architectures have been discussed and used for some time in the software engineering literature and they evolved over the years, adopting key concepts such as views, viewpoints, and frameworks. After an initial process, that involved discussions and brainstorming, this architecture has been conceived.

The IMPReSS Software Architecture is composed of four views, each one representing one stakeholder's view. The stakeholders identified for the IMPReSS Architecture are: a) IMPReSS Partners; b) Application Developers; c) Solution Integrators; d) Final Recipients. The concept of *user* is spread over these four stakeholders and therefore the term has not been adopted to avoid misunderstandings.

The IMPReSS Systems Development Platform (SDP) is divided into two main components, which are the IMPReSS User Interface and the IMPReSS Middleware. Both communicate through the IMPReSS Middleware API. The UI modules runs in foreground and they are directly used by developers for building applications, as well as for monitoring, deployment and maintenance activities. On the other hand, the middleware runs in background where it is invoked by the IDE modules as well as by external software and interacts with resources.

Functional and non-functional requirements have been mapped to the architecture views and modules, in order to guarantee that requirements are fulfilled by one or more components and therefore responsibilities can be tracked through the implementation.

2. Introduction

2.1 Purpose and context of this deliverable

The IMPReSS project aims at solving the complexity of a system development platform (SDP) by providing a holistic approach that includes an Integrated Development Environment (IDE), middleware components, and a deployment tool. The main technical and scientific objectives of the IMPReSS project are:

- Developing an Integrated Development Environment (IDE) to facilitate Model-Driven Development of Smarter Society Services.
- Providing a Service-Oriented Middleware to support Mixed Criticality Applications on Resource-Constrained Platforms.
- Developing easy-to-use and configurable tools for Cloud-based Data Analysis and Context Management.
- Develop Network and Communication management solution to handle the heterogeneity of Internet of Things.
- Creating efficient Deployment Tools for Internet of Things applications.

The project's results will be deployed in the Teatro Amazonas Opera House as an attractive showcase to demonstrate the potential of a smart system for reducing energy usage and CO_2 footprint in an existing public building. Another deployment will be in the campus of the Federal University of Pernambuco.

The IMPReSS platform re-uses and extends results from several existing EU projects on Internet of Things, middleware and energy efficiency and builds on Open Source platforms. The IMPReSS project is carried out by a consortium already experienced with successful EU-Brazil collaboration.

The present document is the output of the task T2.3, whose main goal is to specify the general architecture of the IMPReSS system, including aspects related to the identification of the major system components, how they should interact, and define their external interfaces. It presents the final version of the IMPReSS Software Architecture (or IMPReSS SDP Architecture), which adds upon the first version of the architecture presented in Deliverable D2.2.1 (Kamienski et. al 2014a)

2.2 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.

This deliverable introduces the Final IMPReSS Software Architecture, which is the final view for the design and implementation of the IMPReSS Platform. During the development of the various modules of the IMPReSS Platform different partners refined them so that to implement their intended features. This has been done via a distributed process that generated important feedback for the IMPReSS Architecture since Deliverable D2.2.1 has been published.

This report covers functional as well as non-functional aspects that are of paramount importance to support the integration of different tasks involved in this project

2.3 Document Structure

The remainder of this document is organized in six chapters.

- Chapter 3 (The IMPReSS System Development Platform) describes the IMPReSS concept as background knowledge for the architecture discussion.
- Chapter 4 (Software Architecture and ISO 42010) explains the design process used in the architecture specification.
- Chapter 5 (IMPReSS Software Architecture) presents the architecture of the system.
- Chapter 6 (IMPReSS Middleware Components) introduces some details of four middleware components, namely Context Manager, Data Manager, Resource Manager and Communications Manager.
- Chapter 7 (Scenario-based Architecture Instantiation) presents an instantiation of the IMPReSS architecture focusing on the interaction of some modules for two particular scenarios.
- Chapter 8 (Conclusion) presents the final thoughts about the architecture.

3. The IMPReSS System Development Platform

The IMPReSS development platform consists of a set of technologies organized into a set of modules. In Figure 1, the IMPReSS SDP is presented according the DoW (Description of Work).

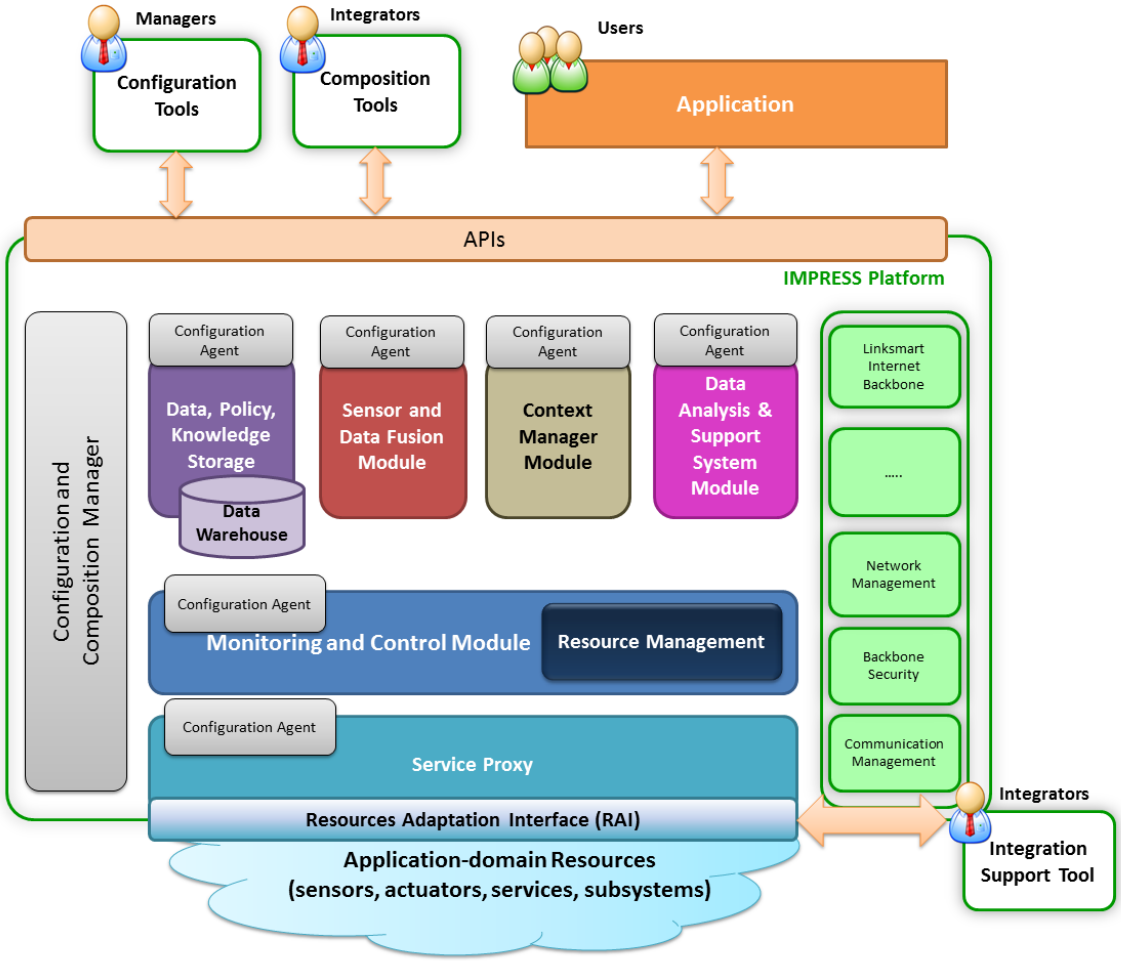

Figure 1 - The IMPReSS Platform (as proposed in the Description of Work – DoW)

The Application-domain Resources represents all the hardware and software that IMPReSS middleware can interoperate with. These entities are physical world devices (e.g. sensors and actuators, as well as hardware in general, such as smart phones and tablets), external and third-parties systems, and open and proprietary services.

The resources are connected to the IMPReSS middleware through *Service Proxies* that expose their functionalities. Service Proxies use a *Resources Adaptation Interface* (RAI) that allows the IMPReSS middleware to connect the Application-domain Resources and expose their functionalities through a common interface.

The Monitoring and Control Module aims to optimise complex system operations acting on available Application-domain Resources exposed by Service Proxies. This module performs also Resource Management operations for solving conflicts and scheduling and management of mixed-criticality. This will make it possible to simultaneously run various 3rd party applications on the IMPReSS as the mixed criticality middleware ensures that the more critical applications are not interfered by the less critical ones when they need to access the resources. Additionally, in a situation where a resource can be used simultaneously by multiple applications, this allows the applications to efficiently share the available resources instead of having a dedicated resource for each application.

The Data, Policy, and Knowledge Storage is responsible for managing the persistence data and information. This component makes the upper layers and modules independent of where the data is

stored, whether locally or in the cloud. Data and information to be maintained include for instance historical sensor data, analysed information, learned knowledge, policies, configurations, etc. Within this component, 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* manages context information using data extracted from available Application-domain Resources. It associates context information to raw and enhanced values. For example, 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.

The *Data Analysis & Support System Module* extracts in a short time the information coming from large amounts of data, in order to 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 the interconnection of 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.

This framework is inspired by the SNMP (Simple Network Management Protocol) 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 and 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 provide methods for combining different modules and commissioning the specific logic flow. The APIs are useful to set the parameters of the platform modules to make the system effective and to operate on application level functionalities (e.g. for system monitoring and control, fine-grained configuration, etc.)

4. Software Architecture and ISO 42010 Standard

This section presents the main concepts related to software architectures and the ISO 42010 standard and is aimed at levelling the knowledge of the readers on the motivation for and terminology of the area. This is needed because Section 5 extensively uses the concepts exposed in this section.

4.1 Software Architecture

The concept of Software Architecture has been around for some time but still there is no formal and well-accepted definition. Nevertheless, some definitions do exist and they are widely used, such as the one given by Kruchten (Kruchten 2003) and repeated by others:

> *"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."*

The software architecture intuitively denotes the high level structures of a software system. It can be defined as the set of structures needed to reason about the software system, which comprise the software elements, the relations between them, and the properties of both elements and relations (Clements 2010). The term software architecture also denotes the set of practices used to select, define or design software architecture. Documenting software architecture facilitates communication between stakeholders, captures early decisions about the high-level design, and allows reuse of design components between projects (Bass 2012).

Software Architecture also plays a key role as a bridge between requirements and implementation and therefore it assumes higher relevance to the IMPReSS project.

4.2 The ISO/IEC/IEEE 42010:2011 Standard

The ISO 42010 standard (ISO 2011), also called "Systems and Software Engineering - Architecture Description" defines requirements on the description of system, software, and enterprise architectures. It aims to standardize the practice of architecture description by defining standard terms, presenting a conceptual foundation for expressing, communicating and reviewing architectures, and specifying requirements that apply to architecture descriptions, architecture frameworks, and architecture description languages.

The standard defines software architecture as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Although this definition is short, it is coherent with the Kruchten definition presented in section 4.1.

ISO 42010 is based on the older IEEE 1471 standard (IEEE 2003). Following its predecessor ISO 42010, it makes an important distinction between architectures and architecture descriptions. Architecture descriptions are used to manage modern systems to improve communication and co-operation, enabling them to work in an integrated and coherent fashion. An architecture description includes one or more architecture views.

4.3 Architecture Views

A view addresses one or more of the concerns held by the system's stakeholders, expressing the architecture of the system-of-interest in accordance with an architecture viewpoint. An architecture view is a collection of models representing the architecture of the whole system relative to a set of

architectural concerns. There are two key reasons to use architecture views. Firstly, because they can better express the system by using different notations, which make it easier to understand and consequently to implement. Secondly, because views are important mechanisms for achieving separation of concerns in complex systems.

A well-known example of using views is the 4+1 Views of Software Architecture (Kruchten 1995). It describes a view model composed of four views - logical, development, process and physical view – with an additional use case view (the +1).

Viewpoints have two important roles in software architectures: establishing conventions about views and framing concerns for stakeholders. An architecture viewpoint frames one or more concerns. A concern can be framed by more than one viewpoint. A view is governed by its viewpoint: the viewpoint establishes the conventions for constructing, interpreting, and analyzing the view to address concerns framed by that viewpoint. Viewpoint conventions can include languages, notations, model kinds, design rules, and/or modelling methods, analysis techniques, and other operations on views.

4.4 Architecture Framework

An architecture framework establishes conventions, principles, and practices for the description of architectures within a specific domain of application and/or community of stakeholders. A framework provides a generic universe and a common vocabulary within which we can all cooperate together - to address a specific issue.

Frameworks do not have to be comprehensive, but they should be leveraged to provide at least a starter set of the issues and concerns that must be addressed in the development of architecture. Frameworks usually use a set of components:

- Views/Presentation: Provide the mechanisms for communicating the information about the relationships in the architecture.
- Methods: Provide the disciplines for gathering and organizing the data. Construct the views in a way that helps ensure integrity, accuracy, and completeness.
- Knowledge: Support the application of the methods and the use of tools for views.

Over the years different frameworks have been defined, aiming at serving as reusable artifacts by software architects.

4.5 IoT Architectural Reference Model

After much discussion about the core concepts of the IoT (Internet of Things) for several years, in 2009 a group of researchers from more than 20 large industrial companies and research institutions joined forces to lay the foundation for the much needed common ground or a common "architecture" for the Internet of Things: the IoT-Architecture project (IoT-A) was born. IoT-A has become the European Commission's flagship project in the European Union's Seventh Framework Program for Research and Development with respect to establishing an architecture for the Internet of Things (Bassi 2013).

The central decision of the IoT-A project was to base its work on the current state of the art, rather than applying a clean slate approach. As a result, common traits have been derived to form the baseline of the IoT Architectural Reference Model (ARM). This has the major advantage of ensuring that the model is backward-compatible, as well as the adoption of established, working solutions for various aspects of the IoT (Bassi 2013).

Figure 2 depicts a functional model of IoT Architecture emphasizing the communication flow among its components. The Functional Model contains seven longitudinal Functionality Groups (light blue) complemented by two transversal Functionality Groups (Management and Security, dark blue). These transversal groups provide functionalities that are required by each of the longitudinal groups.

The policies governing the transversal groups will not only be applied to the groups themselves, but do also pertain to the longitudinal groups.

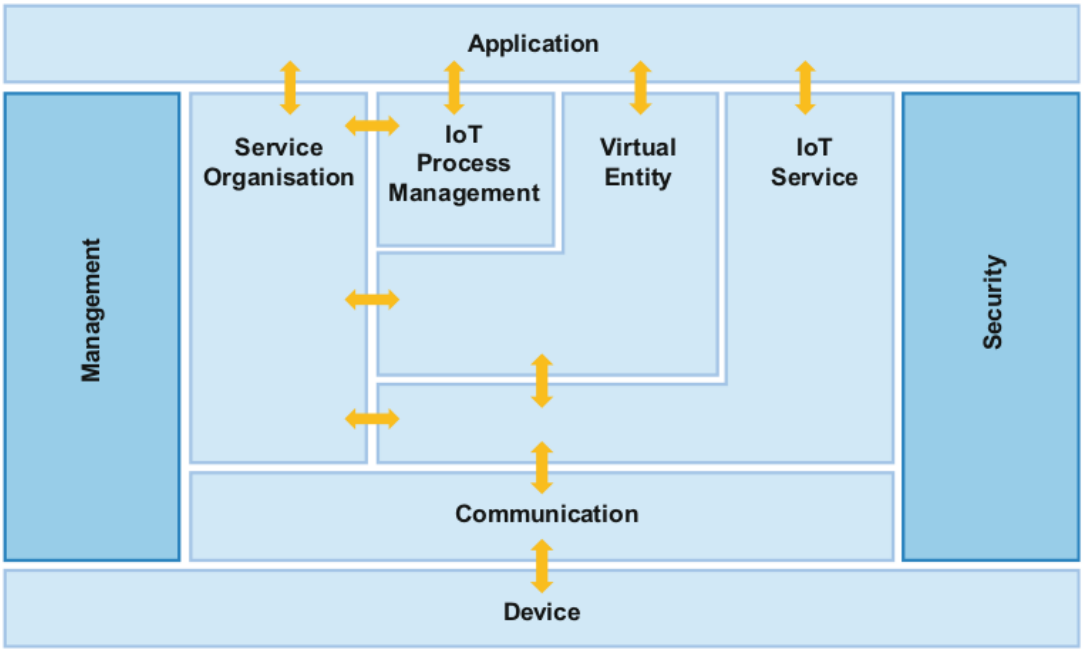

Figure 2 - IoT Architecture

A Physical Entity is represented in the digital world by a Virtual Entity. The IoT Process Management FG relates to the conceptual integration of business process management systems with the IoT ARM. The Service Organisation FG is a central Functionality Group that acts as a communication hub between several other Functionality Groups. The Virtual Entity and IoT Service FGs include functions that relate to interactions on the Virtual-Entity and IoT-Service abstraction levels, respectively. The Communication FG abstracts the variety of interaction schemes derived from the many technologies (Device FG) belonging to IoT systems and provides a common interface to the IoT Service FG. It provides a simple interface for instantiating and for managing high-level information flow. In particular, the following aspects are taken into account: starting from the top layers of the ISO/OSI model it considers data representation, end to end path information, addressing issues (i.e. Locator/ID split), network management and device specific features. The Management FG combines all functionalities that are needed to govern an IoT system. The Security Functionality Group (Security FG) is responsible for ensuring the security and privacy of IoT-A-compliant systems.

Since they have similar purposes, the IoT Reference Architecture share similarities with the IMPReSS Architecture and it will be helpful in driving forthcoming decisions.

5. IMPReSS Software Architecture

The IMPReSS Initial Software Architecture has been inspired by the original IMPReSS platform description, as illustrated by Figure 1. However, some modifications were made due to new requirements and features, as well as a more mature view of the IMPReSS needs. In the remaining part of this section, subsection 5.1 introduces the four IMPReSS stakeholders and section 5.2 defines architecture views, which are in turn described from section 5.3 through section 5.6.

5.1 IMPReSS Stakeholders

The IEEE Std. 1471 definition of stakeholder (IEEE 2000) was adopted: "an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system". For IMPReSS, the choice of stakeholders was of paramount importance, due to its direct translation into architecture views.

Four types of stakeholders have been identified, who may deal with the IMPReSS SDP. Each stakeholder has interests and concerns, which influence the requirements and also the architecture design. These stakeholders are:

- **Partner:** The IMPReSS Partner who contributes to the development of the IMPReSS System Development Platform (SDP). Partners considered here are the European ones - FIT, CNET, IN-JET, ISMB, VTT – and the Brazilian ones - UFPE, UFAM, TAO, CHESF, ENG, UFABC. IMPReSS Partners have a natural broader view of the internal components of the architecture, because they need to put them to work together by orchestrating components and dataflows.
- **Developer:** The Application Developer who uses the IMPReSS SDP to develop IMPReSS-enabled Applications. Target applications are energy efficiency systems addressing the reduction of energy usage and CO_2 footprint, within the context of the Internet of Things (IoT).
- **Integrator**: The Solution Integrator who installs, configures, deploys application, and connects them to other external services and hardware components. Different people or organizations may play the role of integrators. Integrators must have special interfaces (GUIs actually, in different flavors, such as Web-based and smartphone/tablet apps) with the system so that they are easily able to configure the system to operate under different circumstances in different environments.
- **Recipient**: The Final Recipient, who is affected by the solution, such as university professors, students and staff, employees of a company (with different skills and positions), audience of a theater, or even house owners. These people can interact with the solution by means of different interfaces (web-based, apps) for configuring certain parameters and receiving real time information.

The term "user" was intentionally avoided because it can assume different meanings that vary according to different contexts. For example, the typical user of IMPReSS is an Application Developer rather than an end user, because the purpose of IMPReSS is to build a development platform, which by definition is used by developers.

5.2 IMPReSS Architecture Views and Layers

The IMPReSS Software Architecture adopts four views, one for each stakeholder identified in section 5.1. No particular viewpoints are specified, but since stakeholders are in the center of the views, their concerns are represented in the architecture. Figure 3 presents the interaction of the four views, the external components (hardware and software) and the dataflow between stakeholders. Partners, Developers and Integrators have to deal with Physical and Digital resources. The formers are hardware components, mainly sensors and actuators, but also different types of equipment and appliances that may take part in IMPReSS-enabled installations, such as air conditioners and heaters.

Figure 3 starts with the Partner's View following a right-to-left direction dataflow. IMPReSS Partners have the responsibility to perform and fulfill the activities comprised by the workpackages and tasks listed in the DoW. Depending on the task, partners can use digital and physical resources to achieve the goal of the IMPReSS project. In the end, the System Development Platform (SDP) will be developed and used by Application Developers, showed in the Developer's View. Developers also must interact with physical and digital resources when developing their applications, which in turn are used by the Solution Integrator. Integrators also configure physical resources and connect external services (digital resources) to deploy ready-to-use solutions to the Final Recipient. Recipients access the solution in order to take advantage of its features.

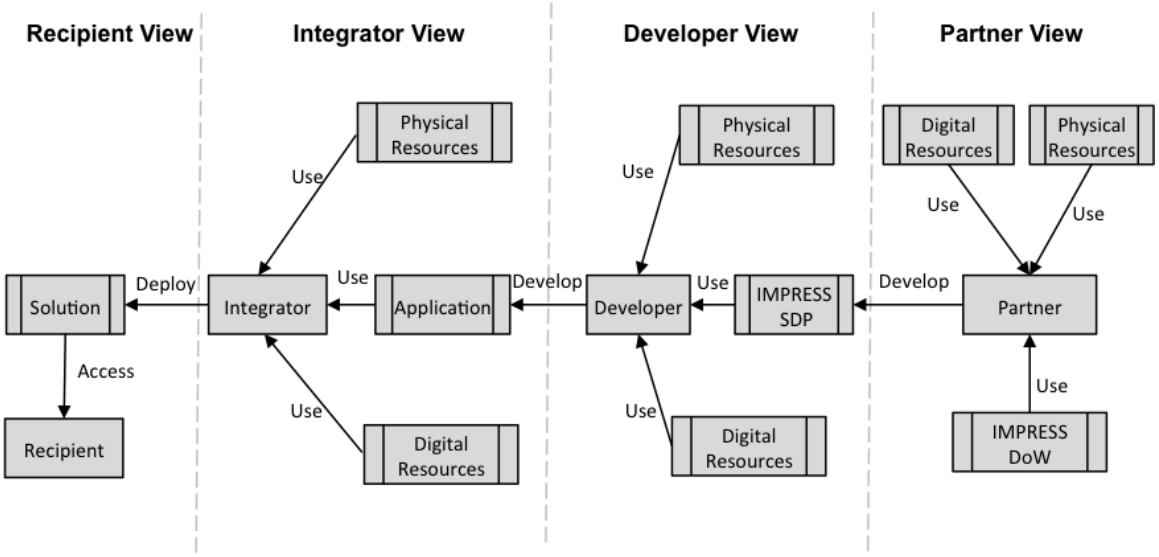

Figure 3 - IMPReSS Architecture Views

The IMPReSS SDP (or platform) that will be used by Developers is composed by two broad software components, namely the IMPReSS User Interface (UI) and the IMPReSS Middleware. The UI runs in foreground and it is directly used by developers for building applications, whereas the middleware runs in background and it is invoked by the UI module as well as by external software. Therefore one can identify three layers in the IMPReSS Architecture (Figure 4):

1. Application/Solution: applications and solutions are placed in the same layer because they are basically the same software, where applications have a broader range of UI options since they are used by Integrators.
2. SDP: Composed by UI and middleware, the SDP uses resources and generates applications (that in turn generate solutions).
3. Resources: Provide data to the Platform (middleware, more specifically) and receive commands from it.

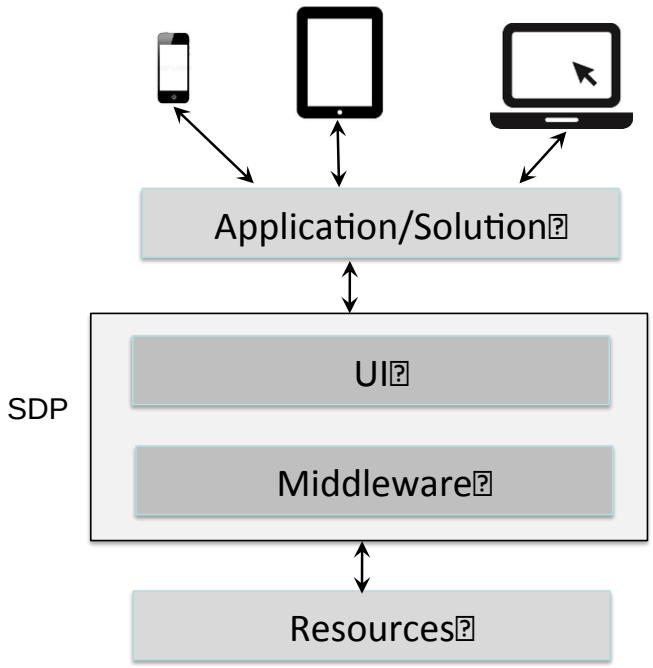

Figure 4 - IMPReSS Architectural Layers

5.3 IMPReSS Partner's View

IMPReSS Partner's View (Figure 5) shows that partners have the most complete view of the IMPReSS Architecture. The UI contains a series of modules for allowing users to interact with the platform and the middleware contains modules with background management responsibilities. The main focus of the IMPReSS Platform has been on the development of Middleware components and graphical interfaces for demonstrators. Therefore, here the UI components will only the described briefly.

IMPReSS assumes that data is stored somewhere in the cloud, using conventional databases or novel ones (such as big data). Local storage can also be used as a particular case and for auxiliary purposes. Please notice that different cloud models may be used, so that public, private, hybrid, and community (NIST 2011) cloud data storages are possible. Also, IMPReSS does not adopt a "one size fits all" approach for data storage, making it possible for different database models to be used for different middleware modules. Modules in the UI component of the IMPReSS Platform have counterparts in the Middleware component and they communicate through the Middleware API.

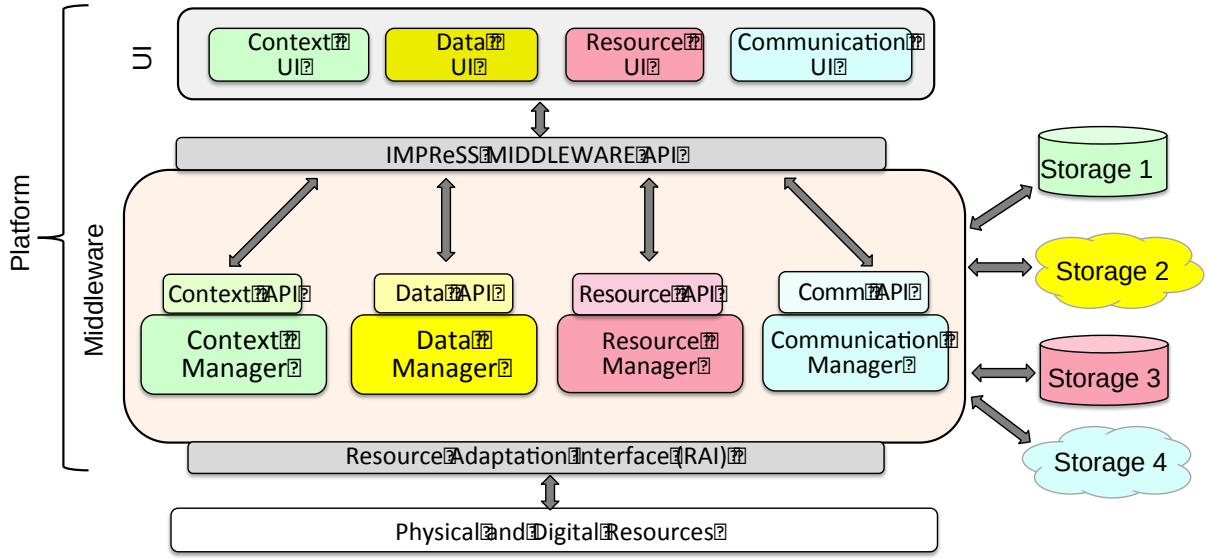

Figure 5 - IMPReSS Partner's View

Both UI and Middleware are comprised of four main modules, which are related to each other:

- Context UI: A graphical tool aimed at managing context information, for allowing Developers to specify which features of context-awareness they need in their applications, ranging from template specification for smart entities and situations to context modeling and rule authoring. In other words, the Context UI exposes to Developers all context-related features of the IMPReSS Platform that they choose to add into their applications. Based on the model defined by Developers, this tool communicates with the background context manager module that implements the templates, rules, sensor and data fusion, context model, and the context-reasoning engine. Developers must also select and developed particular configuration options to be disclosed to Integrators and even Recipients. The Context UI is presented in Deliverable D6.5 (Kamienski et. al 2015b)
- Data UI: A graphical tool aimed at allowing Developers to enter the needed configuration for the data analysis and support module that uses supervised and unsupervised learning for helping IMPReSS applications to make more informed decisions, based not only on real time but also historic data. The Data GUI will configure and interact to the Data Manager module that runs in the IMPReSS Middleware.
- Resource UI: A graphical tool aimed at allowing Developers to specify all particular information needed for the mixed criticality resource management, which may be performed through parameterization or through a specially designed applications classification language. This language is used for describing the run-time requirements of an application in terms of its priority, device access scheme (exclusive or shared) and security. The Resource GUI outputs this information formally as an application criticality description that will be understood by the Resource Manager in the IMPReSS Middleware.
- Communication UI: A graphical tool for allowing Developers to specify all information needed for dealing with communication in the IMPReSS Middleware. This tool is called integration support tool in the IMPReSS DoW and it will provide a collection of templates for different technologies.

The IMPReSS Platform Middleware modules offer background services for their UI counterparts:

- Context Manager: This module encompasses all background software components that a typical context-aware middleware offers to its users (Perera 2013), such as context templates, context models, context reasoning engine, and algorithms for sensor and data fusion. It also interacts with the Storage Manager to data storage and retrieval. Resources might be accessed directly or preferentially through the Resource and Communication Managers. Section 6.1 summarizes the main features of the Context Manager.

- Data Manager: This module provides all software components needed to implement a data and knowledge repository as well as data mining and machine learning algorithms to be used by IMPReSS applications. The machine learning algorithms used to process context-aware information for energy efficiency systems are within the Data Manager. Section 6.2 summarizes the main features of the Data Manager.
- Resource Manager: This module contains all software components needed for managing mixed-criticality resources, such as device and subsystem resource management, resource management and access scheduler, and security features for resource-constrained subsystems. When it comes to implementation, the architecture of both Resource Manager and Communication Manager are tightly related and are presented together in section 6.3.
- Communication Manager: This module implements all communication features of the IMPReSS Platform, such as resource and service discovery and communication and networks management. Also, it plays the role of a proxy (an intermediate module) for the other modules to the Resource Adaptation Interface (RAI). When it comes to implementation, the architecture of both Resource Manager and Communication Manager are tightly related and are presented together in section 6.3. Also, the Resource Adaptation Interface (RAI) is considered an integral part of the Communication Manager.

All UI and Middleware modules, as well as the IMPReSS Middleware API and the Resource Adaptation Interface, will be further specified and refined during the project and documented in the final architecture report.

5.4 IMPReSS Developer's View

Figure 6 depicts the IMPReSS Developer's View, highlighting the UI modules (Context, Data, Resource and Communication) described in section 5.3. Developers have access to the graphical interface and they can also add new modules and integrate them to the application connecting them through the Middleware API. The internal details of the IMPReSS Middleware are hidden from Developers, since the Middleware API provides everything they need. Developers are also aware of the existence of external storage sources and physical and digital resources that must be programmed and tested to work with the Application.

Developers may play the role of Integrators and in the case they have the same view presented in section 5.5.

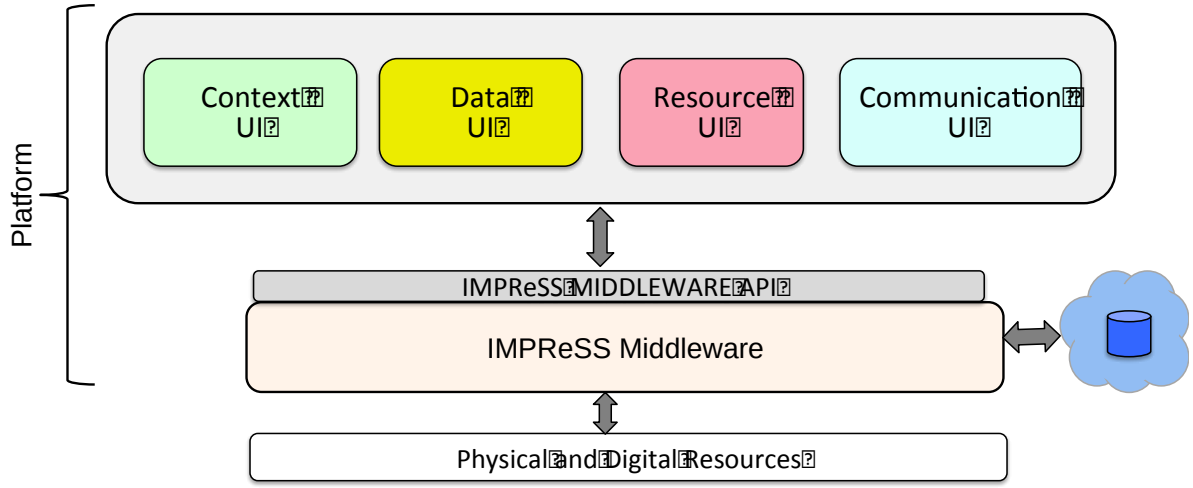

Figure 6 - IMPReSS Developer's View

5.5 IMPReSS Integrator's View

IMPReSS Integrator's View is depicted in Figure 7. Integrators are aware of the Application, which is made available to them by Developers using the IMPReSS UI. During the development of the

application, Developers provided special interfaces for Integrators to be able to configure, install and deploy it. Integrators are aware of the Application and Middleware, since they have to install them and the procedures may be more or less automated for different Applications. Integrators are also aware of the existence of external storage sources and physical and digital resources because they need to interconnect them to the Application and to the Middleware through configuration parameters.

Integrators may play the role of Developers, using the IMPReSS Platform to develop their own Applications. For that particular case, their view is the normal Developer's View presented in section 5.4. Alternatively, Integrators may be software developers using different non-IMPReSS-UI-based platforms and they can connect them to the application through the Middleware API. Examples of non-IMPReSS-UI-based platforms are third-party software commonly used by Integrators or they own in-house developed software. By doing that they are able to enhance an IMPReSS Application with features that have not being considered by both Partners and Developers.

Integrators access IMPReSS Applications through specially designed interfaces, such as Web or apps for smartphones and tablets. Their non-IMPReSS-UI-based applications they access through their own development tools.

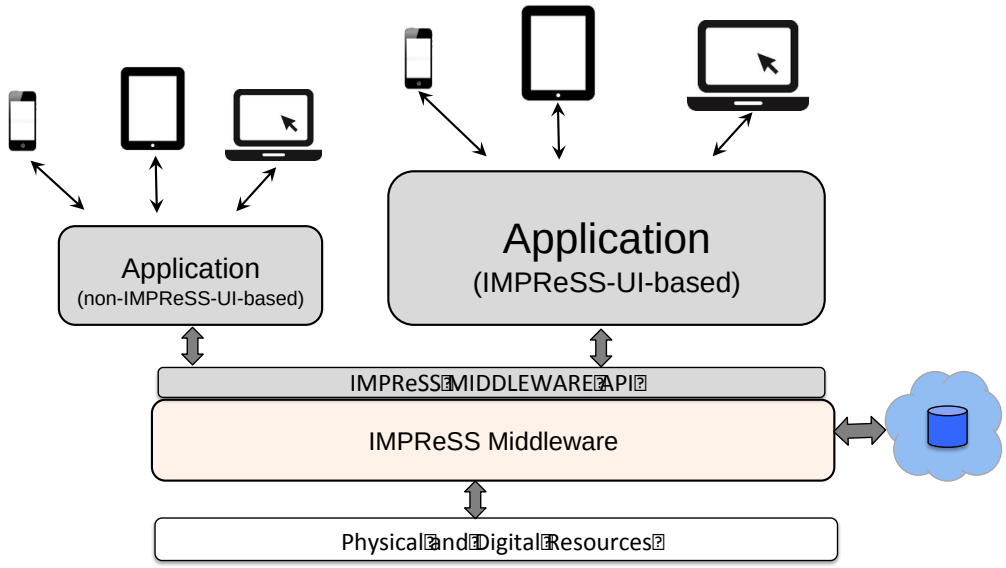

Figure 7 - IMPReSS Integrator's View

5.6 IMPReSS Recipient's View

The IMPReSS Recipient's View is depicted in Figure 8. Recipients have a more limited view of an IMPReSS Application, which is called Solution after being deployed and eventually enhanced and customized by Integrators. Recipients are the end-users or final beneficiaries of the technologies developed by the IMPReSS project. They live, work, or have fun in physical spaces where energy efficiency is considered of paramount importance and thus are immersed in pervasive environments, where sensors and actuators are spread all over the place (physical resources).

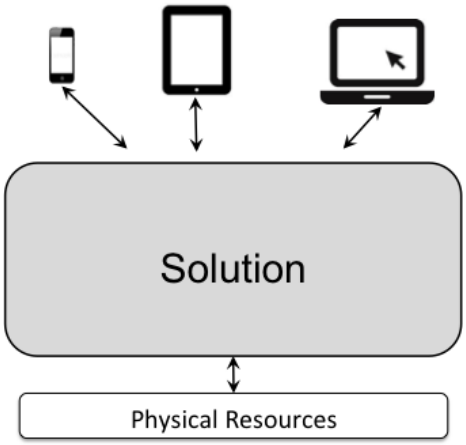

Figure 8 - IMPReSS Recipient's View

6. IMPReSS Middleware Components

This section provides some additional level of details on the four middleware components. Interested readers are recommended to obtain the specific deliverables of each component, references through the text.

6.1 Context Manager

The IMPReSS Context Manager that provides context templates, modeling, reasoning, and algorithms for sensor data fusion. The design of the Context Manager and its implementation are presented in Deliverable D6.3 (Kamienski et. al 2014a) and Deliverable D6.4 (Kamienski et. al 2015a) respectively.

6.1.1 Design Choices

An extensive research in the state of the art in context modelling and reasoning has been undertaken in order to understand the options, choices, tradeoffs, and challenges in context modelling and reasoning better. Based on these findings, two main design choices have been made:

- Object-oriented context modelling: the use of the same paradigm used for modelling and developing systems in the last decades strongly influenced that choice. Moreover, there exists already very mature underlying technology such as persistence storage and rule engines that are designed based on object-oriented approach.
- Rule-based context reasoning: given that object-oriented context modelling was chosen, the natural choice is to use rule-based reasoning, which is commonly used and offers different existing tools that integrate with object-oriented programming languages. However, other approaches for context reasoning may be incorporated later in case the use of rule-based reasoning proofs itself to be insufficient to obtain reasonable results.

6.1.2 Entities and Templates

IMPReSS aims at providing reusable templates for energy efficiency context management, making it easier and faster to add context-awareness features in building automation applications. The design of context templates is characterized by context entities, their relationships, and their attributes. Through an extensive requirements analysis, we identified seven entities that commonly exist in typical context-aware building automation applications: Subject (people), Resource (sensors/actuators), Place (rooms, floors), Fusion (data aggregation), Rule (decisions), Action (commands to actuators) and Activity (schedule).

6.1.3 Context Manager Architecture

Figure 9 depicts the Context Manager Architecture and its relationships with other components of IMPReSS. The IMPReSS Context Manager modules are:

- Context API: part of the IMPReSS Middleware API, it exposes a REST interface, allowing other modules to interact with the Context Manager. Through the Context API entity templates are configured in the Context Storage.
- Context Storage: responsible for storage and retrieval of context entity templates, via the Context API. Any RDBMS with an Object-Relational Mapping (ORM) system may be used. We used EclipseLink as ORM, together with PostgreSQL RDBMS.
- Reasoner: The Context Reasoner infers logical consequences from a set of facts. The Reasoner is invoked by the Fuser and reads entities from the Context Storage. When it is invoked with a set or parameters it searches the entire set of rules for a match, i.e., a rule that matches the parameters. In case of rule conflicts, the Reasoner must select only one rule to be executed based on some resolution mechanism. As a result of firing a rule, one or more actions are performed and they usually refer to changing the configuration of devices or equipments for dynamically adapting behavior, e.g. turning off an elevator or lowering

the temperature of an air conditioner. The Reasoner performs this task by sending command messages to actuators through the Communication Proxy. Our implementation is based on Drools[1] (Expert and Workbench).

- Fuser: responsible for data fusion, which means the use of a set of techniques for combining data from multiple sources or computing statistics. The Fuser is directly connected to the Communication Proxy for receiving real time sensor data and when fusion criteria are met it activates the Reasoner and stores the fused results. Also, fused data may become a virtual sensor and be redirected back to the Fuser. Multiple fusion criteria may be active concurrently and therefore this module plays a key role for the performance of the Context Manager. In our implementation, fusion is performed by Esper[2], which can perform Complex Event Processing CEP) by filtering, analyzing, and fusing events in various ways, configurable through an SQL-like language.
- Scheduler: Manages the agenda for prescheduled events (such as classes in a university) and fires the Reasoner for taking appropriate actions.
- Communication Proxy: encapsulates communication with resources, interfacing with the Communication Manager and with a MQTT[3] broker.
- Local Data Storage: implements internal data storage, for sensor data, fused data and event logging.

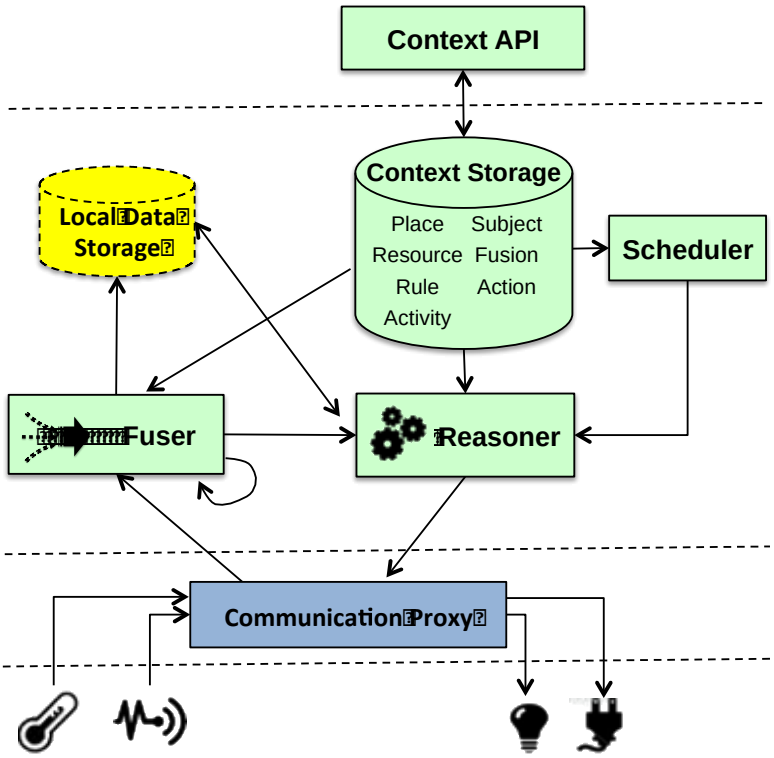

Figure 9 - IMPReSS Context Manager

6.2 Data Manager

The Data Manager consists of a set of technologies responsible for managing and storing data, as well data mining and machine learning algorithms. These technologies are based on a NoSQL database, more specifically, a graph-based one.

[1] www.drools.org
[2] www.espertech.com
[3] mqtt.org

6.2.1 Data Model and Storage

Under the IMPReSS platform, the data semantics and analytics are 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 modeled 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 vertexes and relations between different objects are 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. Or 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, implemented by most well-known NoSQL graph databases (i.e. Titan, Neo4J and OrientDB.). In the realm of graphs' morphism, a property graph is a vertex/edge-labeled/attributed, directed, multi-graph. The data modelling is based on sensor readings arranged in a certain physical environment. The setting may have an infinite number of areas, which in turn may or may not embody other areas within it. Each area may contain an indefinite number of devices that belong to a sensor network. These devices will perform several measurements of the various parameters throughout the day, while it is necessary to store a history of such readings possibly for an indefinite time, depending on application requirements. A generic description of the IMPReSS scenario is shown in Figure 10.

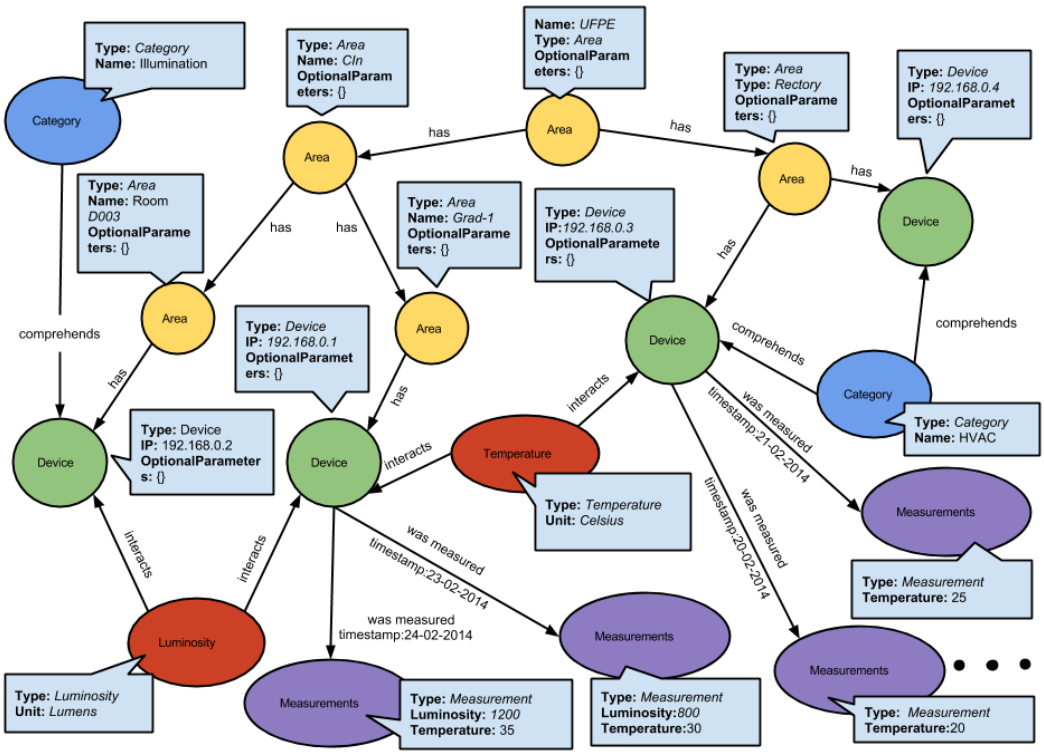

Figure 10 - IMPReSS Data Manager: Data Model for the Data, Policy and Knowledge

Graph databases are perhaps the most popular graph computing technology. They provide transactional semantics such as ACID, which is typical of local databases, and eventual consistency, which is typical of distributed databases. Different from in-memory graph toolkits, graph databases use the disk to store the graph data. 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, does not provide the means for global graph algorithms to be performed within a reasonable milliseconds time scale, in a hundreds of billions of edges scenario. And since WP5 tasks leverage heavily in the data processing for the 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, mainly due to global graph algorithms, is invaluable. Considering this practical concern, we adopt the Titan[4] open implementation as distributed graph-based NoSQL database. The Data Manager Data Model and graph-based implementation are presented in Deliverable D5.1.2 (Gomes et. al 2015).

6.2.2 Data Analysis

Machine Learning algorithms are used to solve tasks for which the design of software using traditional programming techniques is difficult. Machine failures prediction, filter for electronic mail messages and user behaviour identification are examples of these tasks. Several different machine learning algorithms have been proposed in the literature. These algorithms may be divided into three categories: regression, classification and clustering algorithms. This categorization takes into account whether or not one of the following aspects is considered: use of labeled training examples, and real or discrete outputs. Machine Learning algorithms for IMPReSS are presented in Deliverable D5.3 (Souto et. al 2015).

In clustering, samples in the training set are not labeled or classified. The objective is to form clusters or natural groupings of the input samples. Cluster analysis can be used to provide insight in the distribution of data, as a pre-processing level for other algorithms, etc. According to the literature, different clustering algorithms lead to different results.

On the other hand, labeled training samples are available in classification and regression problems. The objective of these algorithms is to find the best functional relationship between input and output, called target or decision function. In regression problems, the outputs are continuous values while the outputs are discrete values in classification problems. Again, several regression and classification algorithms are available in the literature. These methods may achieve different performances when evaluated in different problems.

Based on this context, the algorithms provided in the web application are divided into regression, clustering and classification algorithms. It is important to mention that the algorithms were developed using scikit-learn[5] - an open source and commercially usable library based on Python. Another important characteristic is the fact that scikit-learn is broadly used by companies like Evernote, Spotify and DataRobot. Moreover, this library has a large open source community support and documentation. In the next section, each algorithm used at the application will be explained.

6.3 Resource Manager and Communication Manager

When it comes to implementation, given the nature of mixed criticality systems implemented by the Resource Manager, its implementation is very closely related to the Communication Manager. Therefore, their features are presented together here, according to Deliverable 4.3 (Kiljander et. al 2015).

6.3.1 Mixed Criticality Resource Management Architecture

The mixed criticality resource management architecture (with some other closely related IMPReSS components) is depicted in the Figure 11. It consists of following entities: applications, IoT resources, Global Resource Manager (GRM), Local Resource Manager (LRM) and the Development & Management Tools. These components work in co-operation with the other IMPReSS modules in order to provide mixed criticality resource management functionality at the application and device

[4] http://thinkaurelius.github.io/titan/
[5] http://scikit-learn.org/

levels. In addition to these component an important part of the mixed criticality resource management approach are the application and resource descriptions.

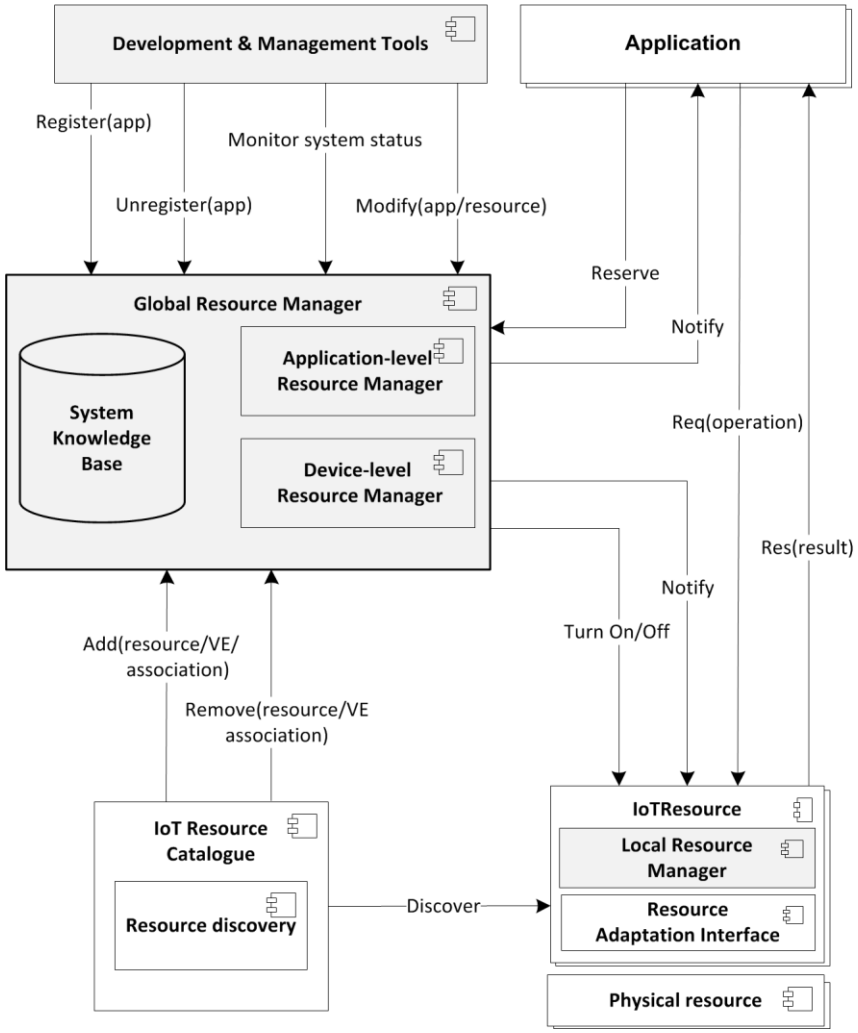

Figure 11 - Mixed criticality resource management architecture

Applications are software processes that provide a certain service for the user by utilizing IoT resources (i.e., sensors and actuators) available in the given IoT environment. The IMPReSS SDP provides two options for the developers. The preferred way to create applications is to use the IMPReSS UI modules. However, it is also possible to write applications manually in any programming language. In either case the application needs to utilize the mixed criticality middleware in order to be able to access the IoT resources provide by the IMPReSS platform.

In the IMPReSS architecture the component responsible for virtualizing the resources and exposing the resource functionality for applications is called IoTResource. There is one IoTResource for each physical resource in the IoT system. The IoTResource consists of LRM and Resource Adaptation Interface (RAI) components of which the LRM is the one responsible for mixed criticality management aspects. The RAI component is described in more detail in Deliverable D3.1 (Ferrera et. al 2014).

6.3.2 Global Resource Manager

At the system level, mixed criticality resource management is executed by the Global Resource Manager. Its main goal is to optimize the behaviour of the IoT system by providing resource management functionality at application and device levels. At the application-level the GRM discovers suitable resources for each application and controls which applications can access which resources in order to make sure that the behaviour of more critical applications is not compromised.

At the device-level the role of the GRM is to make sure that more critical devices are supplied with power in the case of power shutdown. The device-level resource manager is based on the assumption that in the case of a power outage the devices are supplied with power from backup a battery or a generator. Whenever the available energy in the supply drops below a certain limit the application-level resource manager turns of devices with a criticality level below a predefined threshold.

The Global Resource Manager internal architecture is depicted in Figure 12. It consists of three functional software components, called Application-level Resource Manager, Device-level Resource Manager and System Knowledge Base, and three interface modules called Resource Catalogue Interface, Global Resource Manager Protocol and System Knowledge Base Protocol.

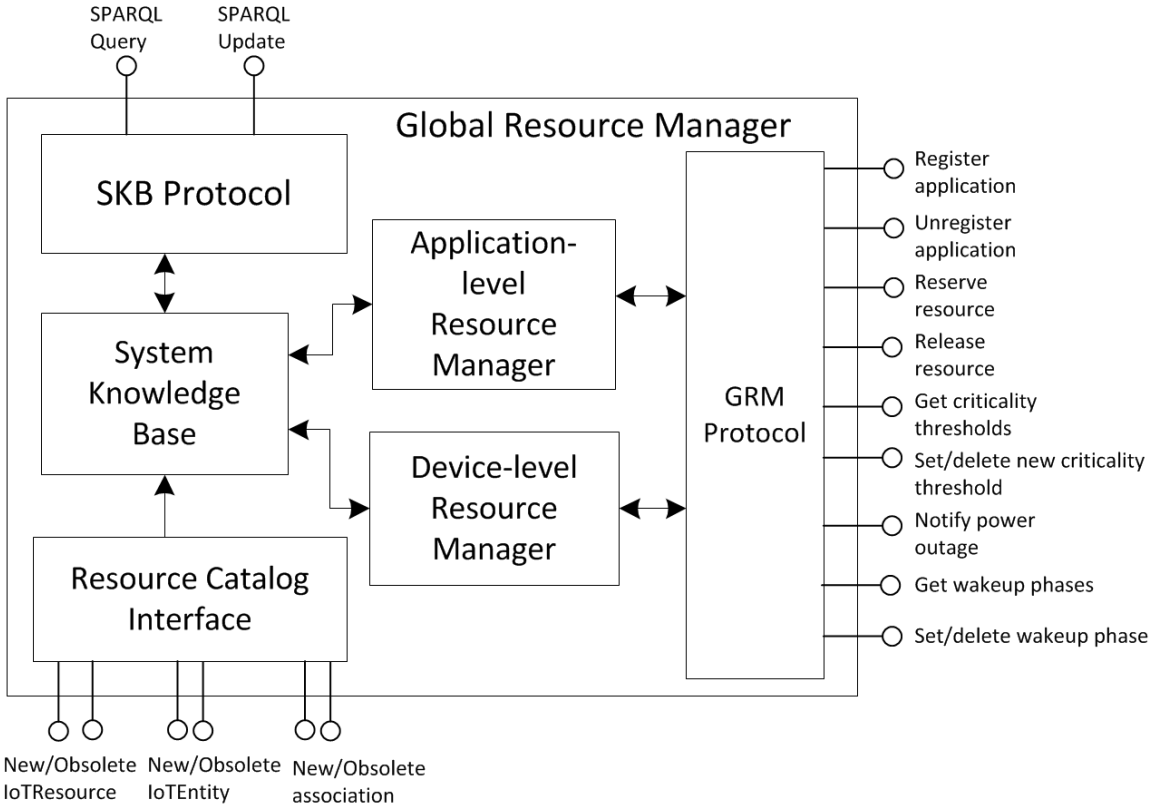

Figure 12 - Global Resource Manager internal software architecture.

6.3.3 Local Resource Manager

At local level the resource management is performed by the LRM. The role of the LRM is twofold. At the application-level it 1) controls that only applications that are authorised by the GRM access the given resources and 2) schedules that request send by the applications (shared access scheme) so that the most critical applications are served first. At the device-level it provides an interface for Global Resource Manager to control which devices are provided with power in the case of a power outage.

The LRM is implemented with Java programming language using Jersey[6] RESTful Web Services framework. The internal architecture of the IoTResource component (including the LRM) is presented in the Figure 13.

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

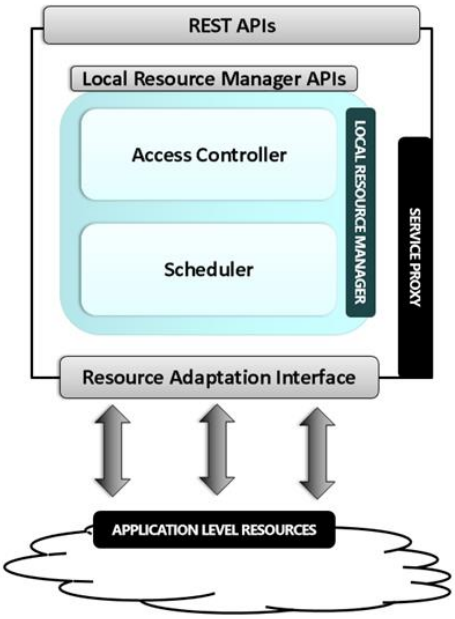

Figure 13 - IoTResource architecture.

7. Scenario-based Architecture Instantiation

The IMPReSS Software Architecture, as depicted by the Partners' view in Figure 5, does not determine any particular way wherein its components can interact. Given that IMPReSS Components provide broad generic services, they can be interconnected in different ways through their REST APIs. In order to be able to see how the components can interact, a specific scenario is needed, which here are defined as two different applications: Energy Saver and Alarm System.

7.1 Energy Saver and Alarm System Scenarios

In the beginning of the scenario there are two applications (Energy saver and Alarm system) and three types of IoT Resources (Presence sensor, Lights, Smart plugs) deployed into the system. The first phase of the scenario is to show how the Energy Saver application works. The lights are switched off when there is no class scheduled. When there is a class, presence sensors detect if a row of seats in the classroom is empty. In the empty areas of the classroom, the lights are automatically switched off.

Alice wonders how much energy will be consumed by the lights in future. For this, a pre-defined model for machine learning has been fed with relevant data, using the Data Manager. Alice can now request forecasts about energy consumption in the room in the IoT Catalogue, which gets the data from the Data Analysis tools.

An important limitation is that presence detection sensors often gives false positives (empirical evaluation), so that the second phase of the scenario demonstrates how new IoT Resources can be easily deployed to enhance existing systems. No modifications need to be made to the existing applications because of the IMPReSS platform. So, Bob, an integrator, decides to add Kinects to improve presence detection. Bob searches for a driver available for the Kinects (this one has been already developed by Alice or another developer and inserted in the repository of the ones available) and then he uses it for the integration. The driver is installed at runtime in his RAI instance, using the configuration tool. Finally, the IoT Resource Catalogue, interacting with the Local Resource Manager, will discover the IoT resource, abstracted using the RAI. By checking the Kinect is appearing in the Resource Catalogue, Bob verifies that the driver installation succeeded. Bob opens the Context Manager, through the Context Modelling Tool (Context UI), and notices that the context "row occupied" is defined through interpretation of the corresponding presence sensor. He extends this context so that is considers a fusion of values from the existing presence sensors and the Kinects.

As part of the configuration phase the Kinects must be added to the mixed criticality management, which determines which power consumers shall be switched off in case of a power outage. For this, all managed devices are connected to switchable smart plugs. Bob opens the Mixed Criticality Resource Manager and sees that so far only an emergency light and two servers have assigned high priority, the other power consumers in the classroom have low priority. Alice assigns the Kinects also low priority. Afterwards, she performs a few tests for the use case again and empirically discovers that there are less false positives. She uses the Context Manager to analyze the false positives. Using the context graph allows her to easily track the context and find out why at a certain point the light was switched off.

In the third phase of the scenario, mixed criticality resource management features are emphasized. First application level mixed criticality is demonstrated. It is shown how the Alarm system takes control of the lights whenever alarm signal is received from sensor (or simulated with an alarm button). When alarm ends the alarm application releases the lights and the control is granted to the energy saver application. Then device level mixed criticality is illustrated by a power outage. The Mixed Criticality Manager switches off the low priority devices immediately. After some time when the fuel has decreased to a certain level, the Mixed Criticality Manager checks again if the power outage still occurs. If yes, it switches off the next class of devices. After the power outage has ended, the Mixed Criticality Manager makes sure to switch on all devices after the grid has stabilized. Higher priority devices are switched on before lower priority ones. The priority of devices

can be easily changed so that it can be adjusted for the needs of the different departments. This is explored in the Mixed Criticality GUI.

7.2 Interaction of Architectural Components

Figure 14 depicts how the components of the IMPReSS Architecture can be interconnected for implementing the Energy Saver and Alarm System Scenario. IMPReSS components are represented in blue, whereas customized components are represented in grey. It can be observed that not necessarily all sub-components are represented in that picture, in order to make it cleaner and to be easier understood by the reader. For example, it is mentioned that Bob uses the Context Manager through the Context UI (Modelling Tool), even though UI components are not represented in the picture. The Energy Saver Application represents a customization of the Context UI for that particular purpose.

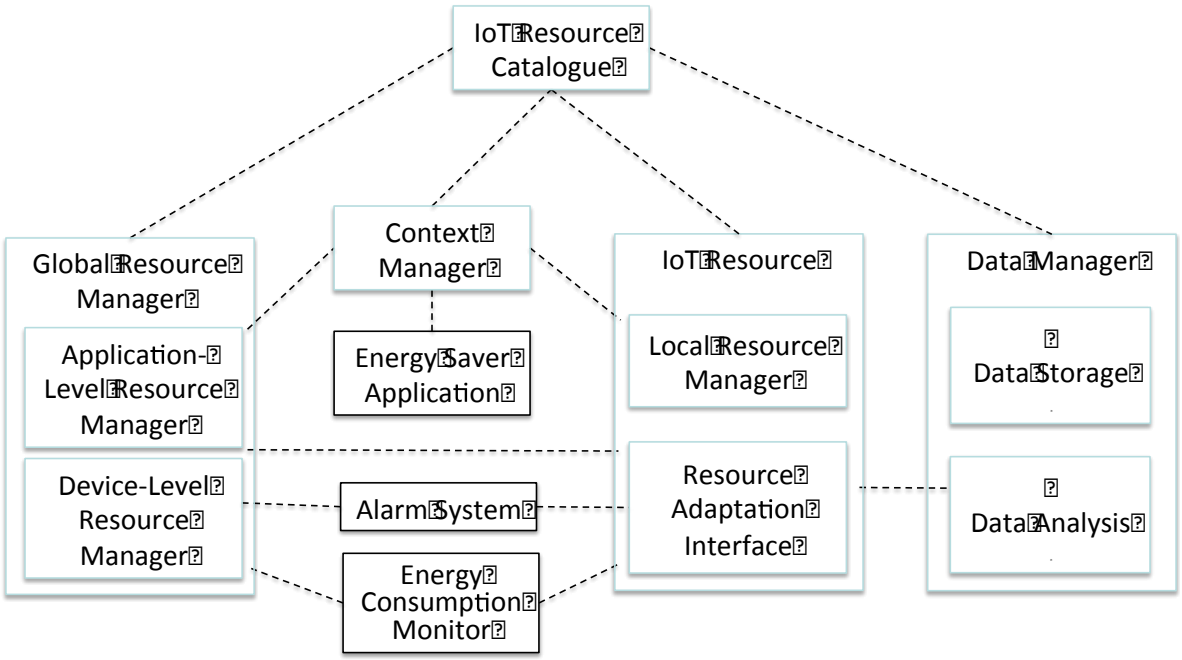

Figure 14 - IMPReSS Architecture – Instantiation for the Energy Saver and Alarm Scenarios

8. Conclusion

This report describes the thoughts and views that lead to the design of the second and final version of the IMPReSS Software Architecture. This initial architecture, described in the DoW, and the initial IMPReSS software architecture, serving as a comprehensive and unique view of the big picture, played a key role in maintaining partners aware of the IMPReSS SDP. Integration was a key concern when adopting a highly distributed software development process and that was the approach followed in the IMPReSS project.

The design of this final version of the IMPReSS Software Architecture involved an extensive learning process about the existing knowledge held by the partners and expressed in the original IMPReSS platform, which was presented in the project proposal. Also, it required certain control of the evolution of the architecture and its approaches since the initial architecture has been published. This architecture has been used for supporting the design of scenarios, prototype implementations, demos, reviews, troubleshooting and performance analysis studies.

9. References

(Bass 2012) Bass, Len; Paul Clements, Rick Kazman (2012). Software Architecture In Practice, Third Edition. Boston: Addison-Wesley. pp. 21–24.

(Bassi 2013) Alessandro Bassi ;Martin Bauer; Martin Fiedler ; Thorsten Kramp ; Rob van Kranenburg ;Sebastian Lange ; Stefan Meissner (Springer 2013). Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model

(Clements 2010) Clements, Paul; Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford (2010). Documenting Software Architectures: Views and Beyond, Second Edition. Boston: Addison-Wesley.

(Ferrera et. al 2014) Ferrera, E., et. al (2014), "Resource Adaptation Interface Framework", IMPReSS Consortium, Deliverable D3.1, June 2014.

(Gomes et. al 2015) Gomes, L. et. al (2015), "Updated Data Analysis & Knowledge Repository Technical Specifications & Guidelines", IMPReSS Consortium, Deliverable D5.1.2, March 2015.

(Hailpern 2006) Hailpern, B., Tarr, P. (2006), "Model-driven development: The good, the bad, and the ugly", IBM Systems Journal, 45(3), pp. 451-461.

(IEEE 2000) IEEE, "Recommended Practice for Architectural Description of Software-intensive Systems", IEEE Std 1471:2000.

(ISO 2011) ISO, "Systems and software engineering — Architecture description", ISO/IEC/IEEE 42010:2011.

(Kamienski et. al 2014a) Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Zyrianoff (2014), SDP Initial Architecture Report, IMPReSS Consortium, Deliverable D2.2.1, February 2014.

(Kamienski et. al 2014b) Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Zyrianoff (2014), Context Management Framework Architecture and Design of Context Templates, IMPReSS Consortium, Deliverable D6.3, November 2014.

(Kamienski et. al 2015a) Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Zyrianoff (2015), Implementation of Context Reasoning Engine, IMPReSS Consortium, Deliverable D6.4, March 2015.

(Kamienski et. al 2015b) Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Zyrianoff (2014), , Deliverable D6.5, March 2015.

(Kiljander et. al 2015) Kiljander, J., et. al (2015), "Resource Management & Access Scheduler", IMPReSS Consortium, Deliverable D4.3, October 2015.

(Kruchten 1995) Kruchten, P., "The 4+1 View Model of Architecture", IEEE Software, vol. 12, no. 6, pp. 42-50, November 1995.

(Kruchten 2003) Kruchten, P., "The Rational Unified Process: An Introduction", Addison-Wesley, 3rd ed., 2003.

(NIST 2011) Mell, P., Grance, T., "The NIST Definition of Cloud Computing", NIST Special Publication 800-145, 2011.

(Perera 2013) Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D., "Context Aware Computing for The Internet of Things: A Survey", Accepted for IEEE Communications Surveys and Tutorials, 2013.

(Souto et. al 2015) Souto, E. et. al, "Data Mining and Machine Learning Tools", IMPReSS Consortium, Deliverable D5.3, February 2015.