(FP7 614100)

D2.2.1 SDP Initial Architecture Report

27 February 2014 – 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: D2.2.1 SDP Initial Architecture Report.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
0.5	Carlos Kamienski (UFABC)	12/02/2014	First draft ready
0.6	Carlos Kamienski (UFABC)	14/02/2014	Second draft ready
0.8	Carlos Kamienski (UFABC)	25/02/2014	Third draft ready
1.0	Carlos Kamienski (UFABC)	28/02/2014	Final version submitted to the EC

Internal review history:

Reviewed by	Date	Summary of comments
Stenio Fernandes (UFPE)	13/02/2014	Minor corrections
Matts Ahlsén (CNET)	18/02/2014	Approved with comments
Jesper Thestrup (IN-JET)	27/02/2014	Approved with comments

Index:

1. Executive summary

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

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

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

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 two key elements: the original specification and the requirements recently gathered. The starting point for the initial specification of the IMPReSS Architecture is the original platform as proposed in the IMPReSS project proposal (i.e. DoW – Description of Work). Also, the software architecture both is influenced and influences the requirements, whose preliminary version has already been presented by the IMPReSS consortium.

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 IDE and the IMPReSS Middleware. Both communicate through the IMPReSS Middleware API. The IDE runs in foreground and it is directly used by developers for building applications, whereas the middleware runs in background in 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 aim of the IMPRESS project is to provide a Systems Development Platform (SDP) which enables rapid and cost effective development of mixed criticality complex systems involving Internet of Things and Services (IoTS) and at the same time facilitates the interplay with users and external systems. The IMPRESS development platform will be usable for any system intended to embrace a smarter society. The demonstration and evaluation of the IMPRESS platform will focus on energy efficiency systems addressing the reduction of energy usage and CO_2 footprint in public buildings, enhancing the intelligence of monitoring and control systems as well as stimulating user energy awareness.

The IMPRESS project aims at solving the complexity of system development platform (SDP) by providing a holistic approach that includes an Integrated Development Environment (IDE), middleware components, and a deployment tool. The 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. The main beneficiaries of the document are the workpackages 2, 3, 4, 5, 6 and 7 that will implement a prototype of the system based on the architecture described here.

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 Initial IMPReSS Software Architecture, which is the starting point for the design and implementation of the IMPReSS Platform. During the development of the various modules of the IMPReSS Platform different groups of partners will refine them so that they are able

to implement the module's functionalities. This is a distributed process that will generate important feedback for the IMPReSS Architecture that will be presented in its final form in Deliverable D2.2.2.

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. The starting point for the initial specification of the IMPReSS Architecture is the original platform as proposed in the IMPReSS project proposal (i.e. DoW – Document of Work). After a discussion and brainstorming process, the IMPReSS architecture has been conceived.

The IMPReSS Software Architecture is composed of four views, each one representing one stakeholder's view. The software architecture both is influenced and influences the requirements, which have been preliminarily presented in Deliverable D2.2.1 (IMPRESS 2014).

2.3 Document Structure

The reminder of this document is organized in four 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 (Requirements vs. Architecture) shows the most important requirements in IMPReSS Architecture.
- Chapter 7 (Conclusion) presents the final thoughts about the proposed architecture.

3. The IMPReSS System Development Platform

The IMPReSS development platform consists of a set of technologies organized into a set of modules. In **Error! Reference source not found.** the IMPReSS SDP is presented according the DoW (Description of Work).

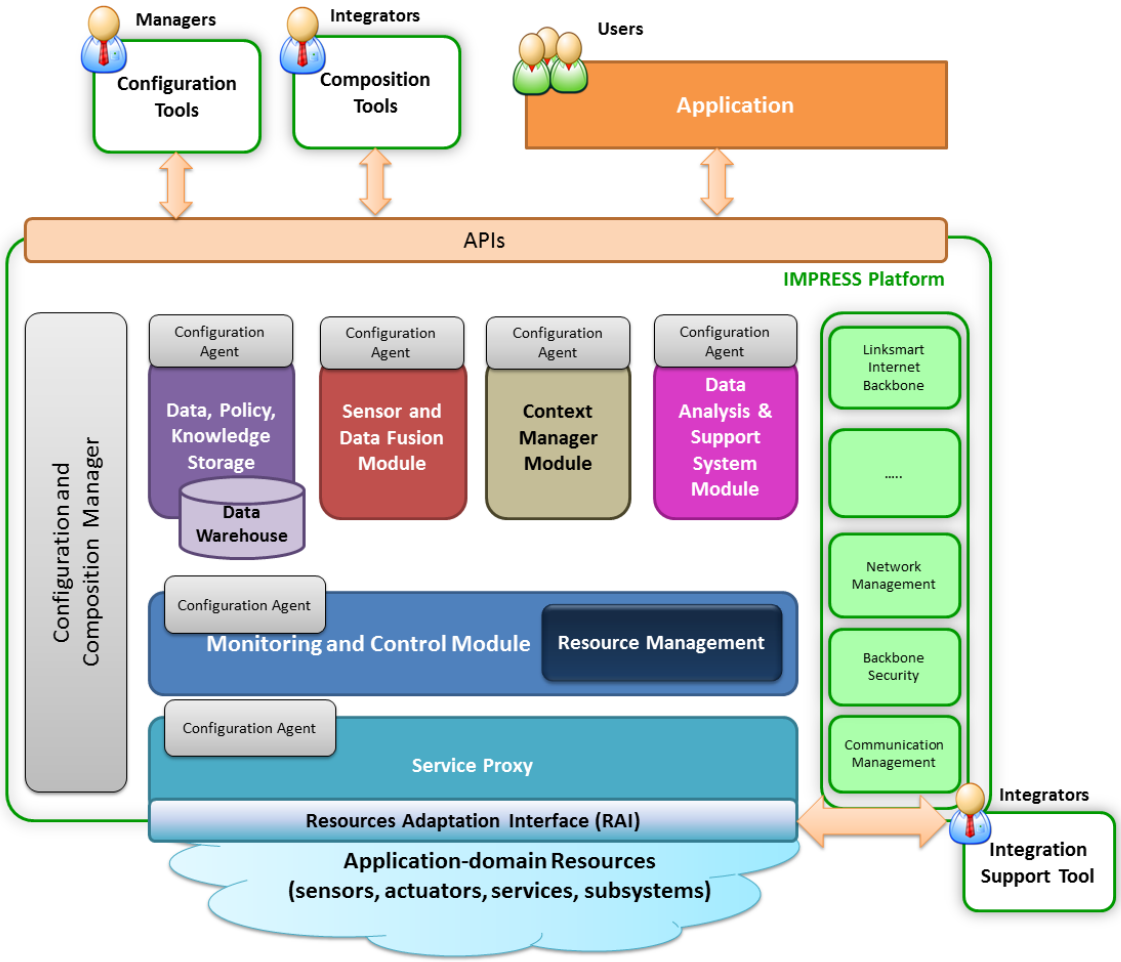

Figure 1 - The IMPReSS Platform (as proposed in the Description of Work – DoW)

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 uses a *Resources Adaptation Interface* (RAI) that allows the IMPReSS middleware to connect the Application-domain Resources and expose their functionalities through a common interface.

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 allow the system 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 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 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 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 insure 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.

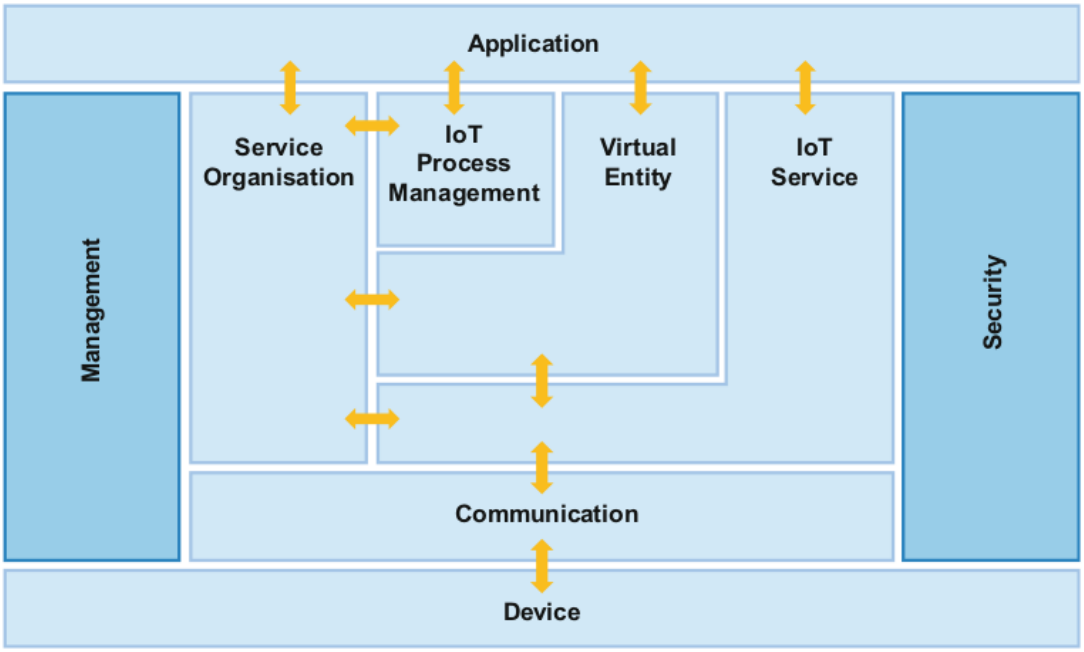

Figure 2 - IoT Architecture

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 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 home 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

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.

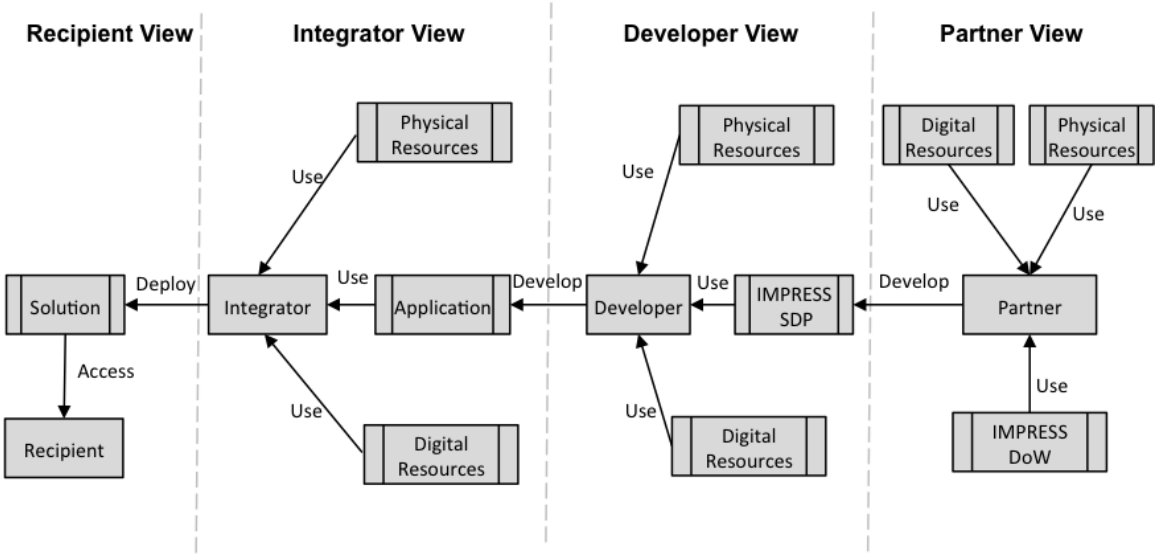

Figure 3 - IMPReSS Architecture Views

The IMPReSS SDP (or platform) that will be used by Developers is composed by two broad software components, namely the Integrated Development Environment (IDE) and the IMPReSS Middleware. The IDE runs in foreground and it is directly used by developers for building applications, whereas the middleware runs in background in it is invoked by the IDE module as well as by external software and interacts with resources. 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 GUI options since they are used by Integrators.
2. SDP: Composed by IDE 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.

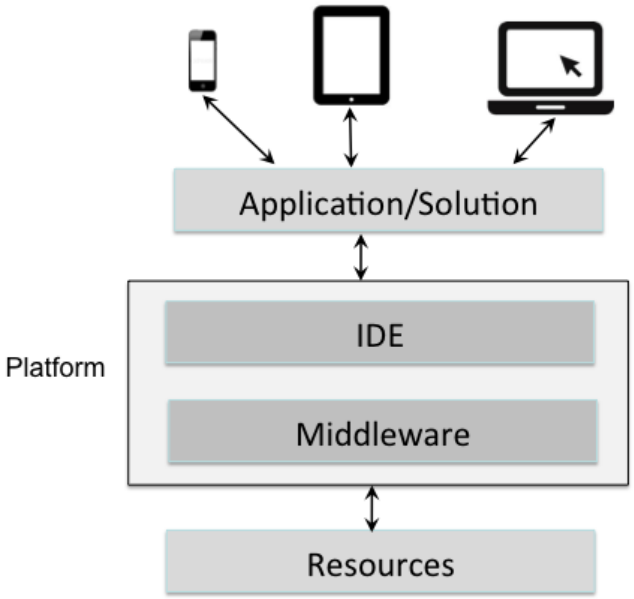

Figure 4 - IMPReSS Architectural Layers

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 IDE contains a series of GUI modules and the middleware contains modules with background management responsibilities. 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 IDE component of the IMPReSS Platform have counterparts in the Middleware component and they communicate through the Middleware API.

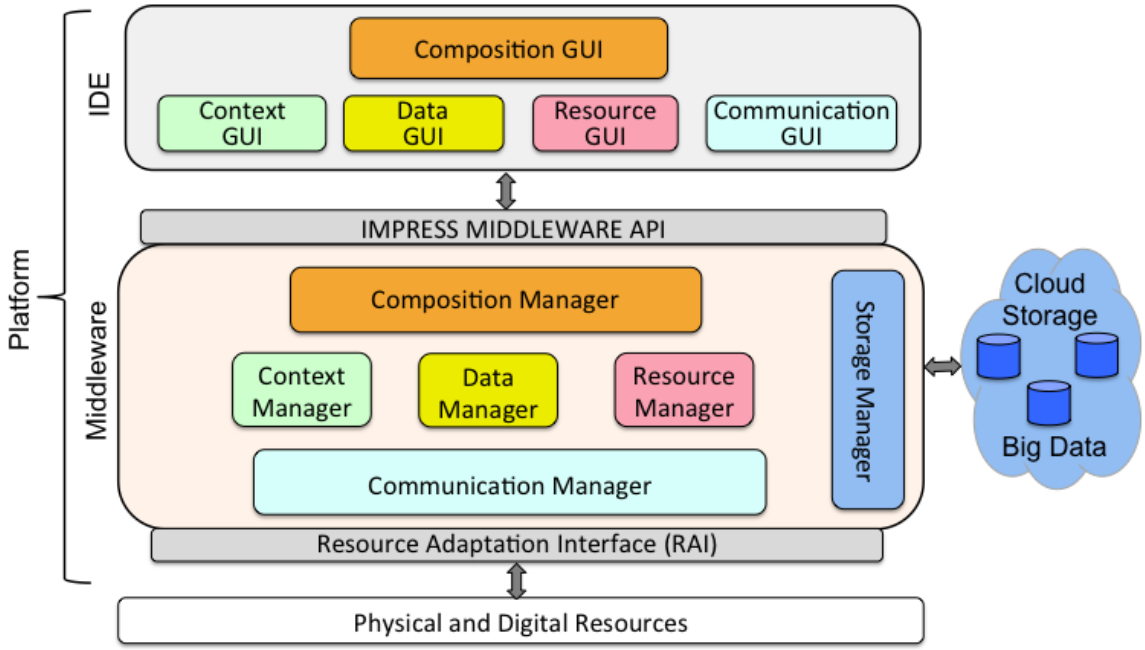

Figure 5 - IMPReSS Partner's View

Both IDE and Middleware are comprised of five main modules, which are related to each other. In addition, Middleware has a module to establish communication with the remote storage. The IMPReSS Platform IDE modules are:

- Composition GUI: A graphical tool for allowing Developers to interconnect the various modules of the platform in a way that better fits the purpose and the needs of their particular applications. This module is the GUI part of the Composition Tool, a commissioning tool presented in the original IMPReSS platform (**Error! Reference source not found.**). It is based on Model-driven development (MDD), a software engineering approach where developers create technology-agnostic models using high levels of abstraction, aiming at simplifying and formalizing the various activities related to the software life cycle management (Hailpern 2006). The Composition GUI runs in foreground and communicates with its twin module Composition Manager in the Middleware, which runs in background.
- Context GUI: 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 GUI 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.
- Data GUI: 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 GUI: 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 GUI: 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 IDE counterparts:

- Composition Manager: This module is an engine that runs in background and supports the Composition GUI. For example, it may be implemented as an Web Services Engine that supports the MDD approach disclosed by the Composition GUI to Developers.
- 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.
- Data Manager: This module provides all software components needed to implement data analysis and historic context information that will be used by IMPReSS applications. The Data Manager also stores and retrieves its raw and processed data using the Storage Manager. The machine learning algorithms used to process context-aware information for

energy efficiency systems are within the Data Manager. As for the Context Manager, resources can be accessed directly or through the Resource and Communication Managers.

- 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. It provides functionalities to the IMPReSS middleware that in Figure 1 are identified as Monitoring and Control.
- 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). Figure 1 identifies it as the Service Proxy module and all modules related to the LinkSmart middleware.
- Storage Manager: This module is logically represented as a single and centralized software component in Figure 5, though its implementation can be as decentralized and distributed as the other modules need to. It provides an interface to different storage approaches, ranging from traditional relational databases stored in the cloud to big data and NoSQL databases.

All IDE 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 IDE GUI modules (Composition, 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.

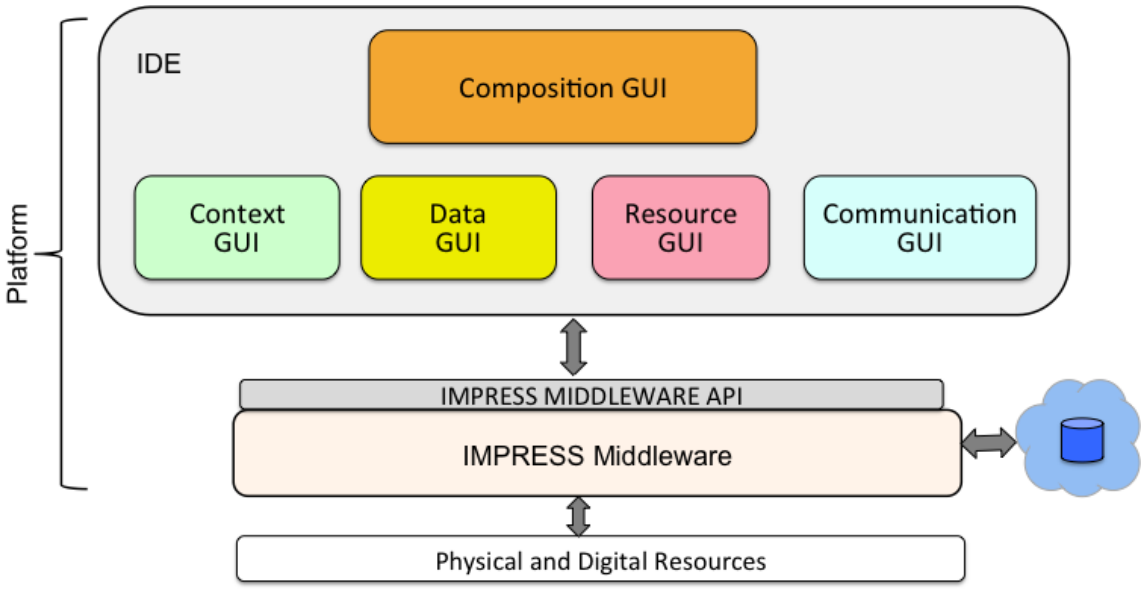

Figure 6 - IMPReSS Developer's View

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 IDE. 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-enabled platforms and they can connect them to the application through the Middleware API. Examples of non-IMPReSS-enabled 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-enable applications they access through their own development tools.

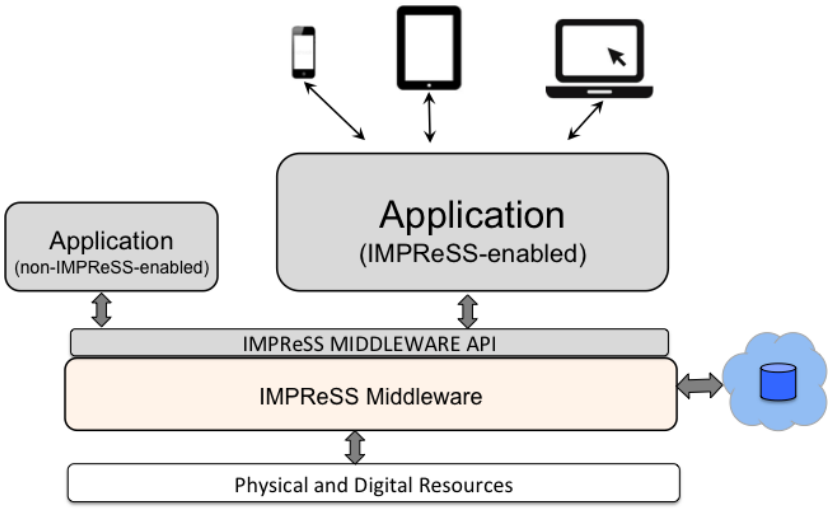

Figure 7 - IMPReSS Integrator's View

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).

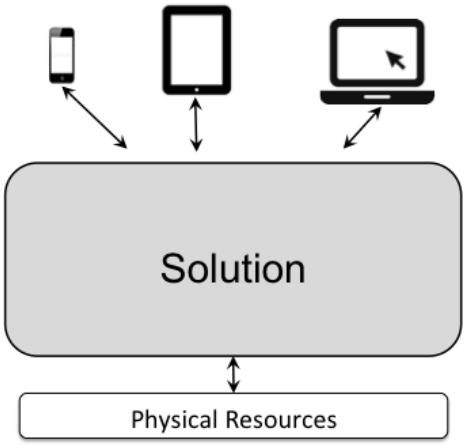

Figure 8 - IMPReSS Recipient's View

5.7 IMPReSS Architecture vs. Effort Distribution

Figure 9 depicts the IMPReSS Partner's View assigning to all modules a Workpackage according to the effort distribution specified in the DoW.

- Workpackage 2: IMPReSS Middleware API
- Workpackage 3: Communication GUI and Manager; Resource Adaptation Interface
- Workpackage 4: Resource GUI and Manager
- Workpackage 5: Data GUI and Manager
- Workpackage 6: Context GUI and Manager
- Workpackage 7: Composition GUI and Manager
- Workpackage 3, 5, 5, 6, 7: Storage Manager

As emphasized in section 5.3 the Storage Manager is logical centralized but physically distributed, so that all workpackages will be responsible for it, according to their needs.

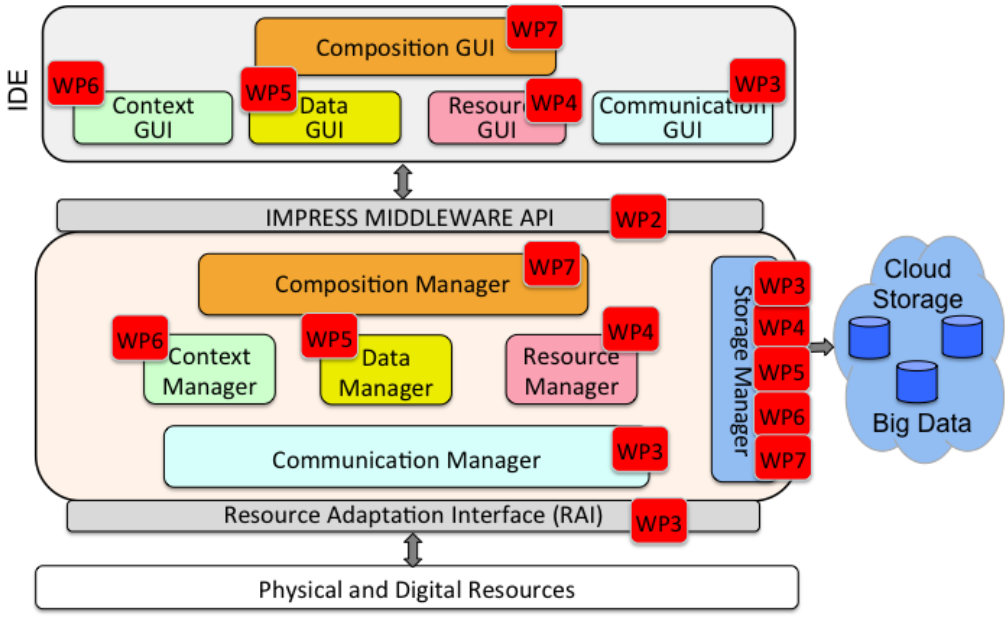

Figure 9 - Partner's View vs. Effort Distribution

Since Workpackages 1 and 9 are focused at non-technical activities - management and dissemination/exploitation - they are not related to the architecture.

Also, as depicted by Figure 10, Workpackage 8 will develop pilot Applications, thus playing the role of a Developer in using IMPReSS IDE modules.

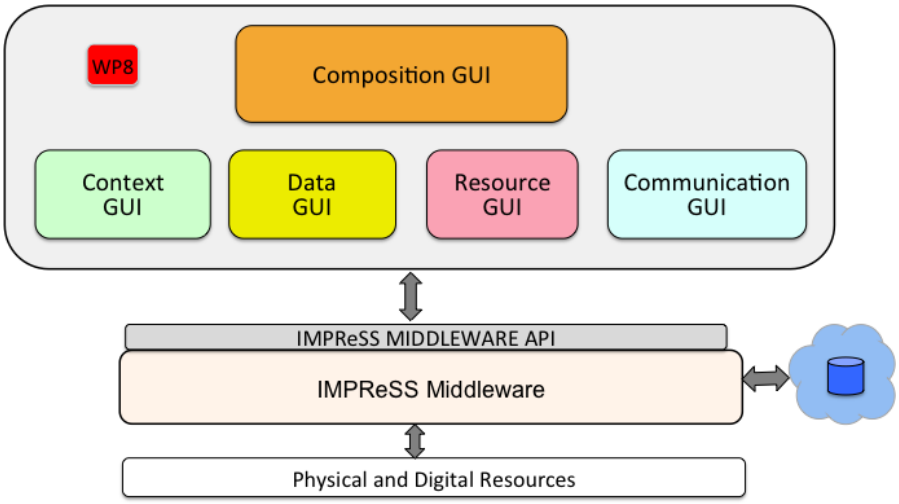

Figure 10 - Developer's View vs. Effort Distribution

6. Requirements vs. Architecture

IMPReSS Deliverable D2.1.1 Initial Requirement Report present the application- and technical-based requirements that together with the Description of Work (DoW) influenced the IMPReSS Architecture. As such, the architecture presented in this document fulfills the requirements gathered from a extensive and collaborative process. Conversely, requirements are also influenced by the Architecture, as described in section 12, because for each software module of the architecture, requirements should be listed.

6.1 Functional Requirements to Architecture Mapping

In Table 1 each functional requirement identified in D2.1.1 is mapped to some architecture component and a short explanation is also presented.

Table 1 - Functional Requirement to Architecture Mapping

Functional Requirement	Architecture	Comment
IMP-3 Devices should be allocated to a logical area	Communication Manager	Communication details are encapsulated by the Communication Manager
IMP-4 Devices should be allocated to one or more groups	Communication Manager	Communication details are encapsulated by the Communication Manager
IMP-5 The data should be persisted in NoSQL database	Storage Manager	The Storage Manager is able to access NoSQL as well as traditional databases
IMP-6 The data should be analysed using data mining and machine learning techniques to find relevant information and make predictions.	Data Manager	Data analysis is performed inside the Data Manager
IMP-7 SDP will have a communication layer that allows storing data in the IMPReSS Cloud	Storage Manager	The Storage Manager is able to access cloud storage as well as other storage approaches
IMP-8 The application should provide historical energy consumption and use of electrical devices.	Data GUI and Manager	Developers use the Data GUI to configure appropriate historical information that is dealt with by the Data Manager
IMP-12 Access prioritization to resources	Resource Manager	Based on application classification (e.g. critical and non-critical), the Resource Manager prioritizes resource access
IMP-13 Annotate application with the level of criticality	Resource GUI	Resource GUI allows the classification of applications, to be interpreted by the Resource Manager
IMP-16 Reusable components for trend analysis and forecasting of energy and occupancy data	Data Manager	This is pertaining to the implementation details of the Data Manager
IMP-17 Dynamically adjustable security level for resource constrained devices	Resource Manager	The Resource Manage should be able to deploy such level of dynamicity in enforcing security
IMP-21 Graphical model-driven commissioning tool	Composition GUI	The Composition GUI is a MDD based commissioning tool

Functional Requirement	Architecture	Comment
IMP-22 Runtime services/devices discovery and commissioning	Communication Manager	Services and devices are discovered by particular sub-modules inside the Communication Manager
IMP-23 Development toolkit for resources integration	Resource GUI	The Resource GUI should be able to provide features to resource integration
IMP-24 APIs definition	IMPReSS Middleware API an RAI	APIs are to be defined for making it possible to isolate the middleware from the IDE and from the resources
IMP-25 IMPReSS architecture views	Partner's View Developer's View Integrator's View Recipient's View	The IMPReSS Software Architecture is composed of four views
IMP-26 Templates for smart entities	Context GUI and Manager	Templates for smart entities are specified by the Context GUI and processed by the Context Manager

6.2 Non-Functional Requirements to Architecture Mapping

Table 2 presents non-functional requirements taken from Deliverable D2.1.1 and map them to some architecture component.

Table 2 - Non-Functional Requirement to Architecture Mapping

Functional Requirement	Architecture	Comment
IMP-1 Sensors must be unobtrusive	-	This requirement is not directly related to the architecture
IMP-9 The SDP should encapsulate the complexity of different technologies, developing a single logic to devices manipulation.	Communication Manager	The Communication Manager encapsulate details of a variety of different technologies used by the physical resources
IMP-10 The SDP shall support multiple communication protocols	Communication Manager	Different resources may require different protocols so the Communication Manager must provide seamless and transparent communication to them
IMP-11 The software components of the middleware should be modularized	Partner's View	The main software components are defined in the Partner's View and will be further refined
IMP-14 The impress core runs on a Gateway that cost below USD50	-	This requirement is a warning for the development of the IMPReSS platform, that should be kept as light-weighted as possible to be able to run on resource constrained devices
IMP-18 The IMPReSS platform should support development of IoT systems that are extendable for future needs	Partner's View	IMPReSS Software Architecture is aimed at being extended for different needs, both IDE and Middleware.

Functional Requirement	Architecture	Comment
IMP-19 The IMPReSS platform should be agnostic to the application domain	Developer's View	IMPReSS Software Architecture is in principle orthogonal to application domains and Developers can develop different Applications for different purposes using the IMPReSS Platform. The IMPReSS project will focus on two pilot applications for saving energy, in Teatro Amazonas and the UFABC campus.
IMP-20 The IMPReSS SDP should be easy to use	IMPReSS IDE	The GUI Modules inside the IDE should be seamlessly integrated and should provide adequate quality of experience levels for Developers
IMP-27 Data in the IMPReSS network is classified to different categories based on the criticality	Resource Manager Communication Manager	Resource Manager and Communication Manager should be integrated in such a way to allow data flowing from applications classified to different criticality levels to be treated likewise
IMP-28 Confidentiality of the messages between IMPReSS platform devices can be guaranteed	Communication Manager	Communication Security is encapsulated by the Communication Manager
IMP-29 Integrity of the messages between IMPReSS devices can be guaranteed	Communication Manager	Communication Security is encapsulated by the Communication Manager
IMP-30 Availability of the critical IMPReSS devices must be guaranteed	Resource Manager Communication Manager	Monitoring should be performed for the software to be able to make decisions related to fault tolerance. Hardware availability can be only guaranteed by physical redundancy
IMP-31 Data transmitted in the IMPReSS network is classified to different classes based on the confidentiality.	Resource Manager Communication Manager	Resource Manager and Communication Manager should be integrated in such a way to allow data flowing from applications classified to different criticality levels to be treated likewise

7. Conclusion

This reports described the initial thoughts and views that lead to the design of the first version of the IMPReSS Software Architecture. This initial architecture, serving as a comprehensive and unique view of the big picture, plays a key role in maintaining partners aware of the IMPReSS platform so that they can always have that in mind when designing and implementing particular modules. Integration is a key concern when adopting a highly distributed software development process.

During the upcoming activities, partners will design their own modules' architecture by refining this initial architecture. Also, APIs will be defined in order to keep a controlled and ordered communication between software modules. This process will provide useful insights and feedback for making it possible to come up with a more complete and up-to-date architecture that will be documented in the future.

The design of this initial 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 out of the box thinking, in order for making it possible to understand the functionality and role of each module and to simplify the architectural design, at least in the starting phase of the project. The role of stakeholders has been defined and they influenced the architecture views defined in this document. Last, but not least, requirements should not be forgotten during the development process and to this end they have been mapped to architecture components.

8. 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.

(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.

(IMPRESS 2014) IMPReSS, "Initial Requirement Report", IMPReSS Deliverable D2.1.1, January 2014.

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

(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.