(FP7 614100)

D6.3 Context Management Framework Architecture and Design of Context Templates

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 version: 1.0 Submission date: 28/11/2014

Document control page

Document file:	d6.3_context_management_framework_architecture_v1.0.docx
Document version:	1.0
Document owner:	Carlos Kamienski (UFABC)
Work package:	WP6 – WP6 Software System Engineering and Context Management
Task:	Task 6.2, Task 6.3, Task 6.4
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	01/11/2014	Initial Version
0.4	Gabriela de Oliveira	14/11/2014	Most content added in all sections
0.7	Carlos Kamienski	25/11/2014	First complete version ready
1.0	Carlos Kamienski	28/11/2014	Final version ready

Internal review history:

Reviewed by	Date	Summary of comments
Jussi Kiljander	27/11/2014	Minor grammar corrections. Few comments about the context model and architecture.
Ferry Pramudianto	27/11/2014	Approved with corrections

Index:

1. Executive summary

IMPReSS aims at providing a Systems Development Platform (SDP) for enabling rapid development of mixed critical complex systems involving Internet of Things and Services (IoTS). 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. Application developers will develop applications using the SDP for a variety of purposes, including energy efficiency management.

In order to provide an efficient use of energy in buildings, the IMPReSS SDP will need to be context aware, which means that it must know what happens inside the buildings so that opportunities to save energy can be identified and effectively fulfilled. Context-aware systems are able to adapt their operations according to the current conditions without any explicit user intervention. Based on our research, we identify that the context life cycle composed of four phases including context acquisition, context modelling, context reasoning, and context dissemination. The key components of any context-aware system are the context model and the context reasoning approach, used in a particular context-aware management system. In addition, context modelling and reasoning are intrinsically related to each other and must be defined together. The choice of a context model is a very important step in any context-aware system because it influences the whole lifecycle of the system (e.g., productivity of the developers, the expressiveness of the system, performance). Moreover, it also limits the choice of a context reasoning technique (e.g., logic based, statistical inferences).

The IMPReSS SDP is divided into IDE and Middleware, where the IDE contains a series of GUI modules and the middleware contains modules with background management responsibilities. The Context IDE is a graphical tool for managing context information, 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 modelling and rule authoring. The Context Manager encompasses all background software components that a typical context-aware middleware offers to its users, such as context templates, context models, context reasoning engine, and algorithms for sensor and data fusion. This document updates the Deliverable D2.2.1 (Kamienski et. al 2014a) based on the most recent discussions and general understanding among IMPReSS partners.

The work package 6 provides entities and templates for energy efficiency applications, which is aimed at simplifying the developer's tasks for modelling and programing the context-awareness features in their applications. This document provides an update of deliverable D6.1 (Kamienski et al. 2014b), based on the most recent developments of context model and reasoning. The design of context templates is characterized by context entities, their relationships and their attributes, which play a key role in the architecture of the Context Manager. Eight entities of typical scenarios for context-aware management for making efficient use of energy in buildings have being identified and each one generated a variety of templates: Subject, Resource, Place, Fusion, Rule, Action, Activity and Notification. The last three ones are being introduced in this documented where the first five ones have been identified in Deliverable D6.1 (Kamienski 2014b).

After introducing some background on the context-aware computing and up-to-date versions of the IMPReSS Software Architecture and Entities/Templates, this document focuses on the Architecture of the Context Manager, also known as Context Management Framework Architecture. The context modelling in IMPReSS will be based on an object-oriented approach, and a rule-based approach will be used for the context reasoning. The use of an object-oriented context modelling is due to its high penetration and widespread use by the software development community, which makes its adoption easier. Moreover, there exists already very mature underlying technology such as persistence storage and rule engines that are designed based on object-oriented approach. This will reduce the development efforts, as well as ensuring the maturity of the context management framework.

Given that object-oriented context modelling was chosen, the natural choice is to use a rule-based reasoning, which is commonly used, and offers different existing tools that integrate well with object-oriented programming languages. This architecture can be divided into two main planes inside the IMPReSS Middleware, namely the control plane and the event plane. It also includes modules that are in the IMPReSS Middleware Interface and in the Resource Adaptation Interface. Some implementation guidelines are also presented, based on some literature research and hands-on experience with existing Complex Event Processing, Rule Engines, protocols for the Internet of Things (IoT), and related technology. Finally, in order to simplify the interworking of the Context Manager components this documents sheds some light into the data flow and processing steps involved in the process of analyzing sensor data and sending commands to actuators.

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 present document is the output of Task 6.3 (Context Modelling Templates), whose main goal is to define a context modelling technique and associated context reasoning approach to be used in the IMPReSS project. It contains the specification for the architecture of the Context Manager whose reference implementation will be presented in D6.4 (Implementation of Context Reasoning Engine). As the core component of the context management framework, the context reasoning engine will provide the context awareness services to the application. It processes the context model provided by the application (and created through the tools included in the framework) and constantly monitors the state of the smart entities. It utilises the sensor and data fusion services in order to obtain the required information, and detects the occurrence of situations (i.e., specified states of a given set of smart entities) defined within the context model.

2.2 Scope of this deliverable

In order to allow applications to make efficient use of energy in buildings, the IMPReSS Platform must provide context-aware management features, so that automatic decisions can be made based on existing context information coming from a variety of sources, including physical sensors, calendars and business rules. The specification of the architecture for Context Management is a key achievement for the IMPReSS project, since it will guide the further developments of the software that will effectively provide context-aware features in the Platform.

This deliverable is aimed at providing a clear understanding the Context Management architecture and puts it into the correct framework of the IMPReSS Software Architecture. Also, it is aimed at contextualizing according to the state of the art in context modelling and reasoning techniques.

2.3 Document Structure

The reminder of this document is organized in four chapters.

- Chapter 3 presents the IMPReSS System Development Platform (SDP) as specified by a former work within IMPReSS, which is based on two main modules, IDE and middleware.
- Chapter 4 introduces the main concepts about context-aware computing, focusing on the main techniques for context modelling and context reasoning.
- Chapter 5 define eight entities and their templates for context modeling of energy efficiency scenarios, aimed at making it easier to understand, model and program the context-awareness features of the IMPReSS project.
- Chapter 6 introduces the architecture for the Context Manager module of the IMPReSS architecture, focusing on its main internal components and interactions with other modules and with external actors, such as IDE modules. Also, guidelines for implementation used in a preliminary implementation are provided, along with examples of executing the event plane of the architecture.
- Chapter 7 presents some final remarks and the next steps.

3. The IMPReSS System Development Platform

IMPReSS Software Architecture adopts four views: Partner, Developer, Integrator and Recipient. Figure 1 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 1 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.

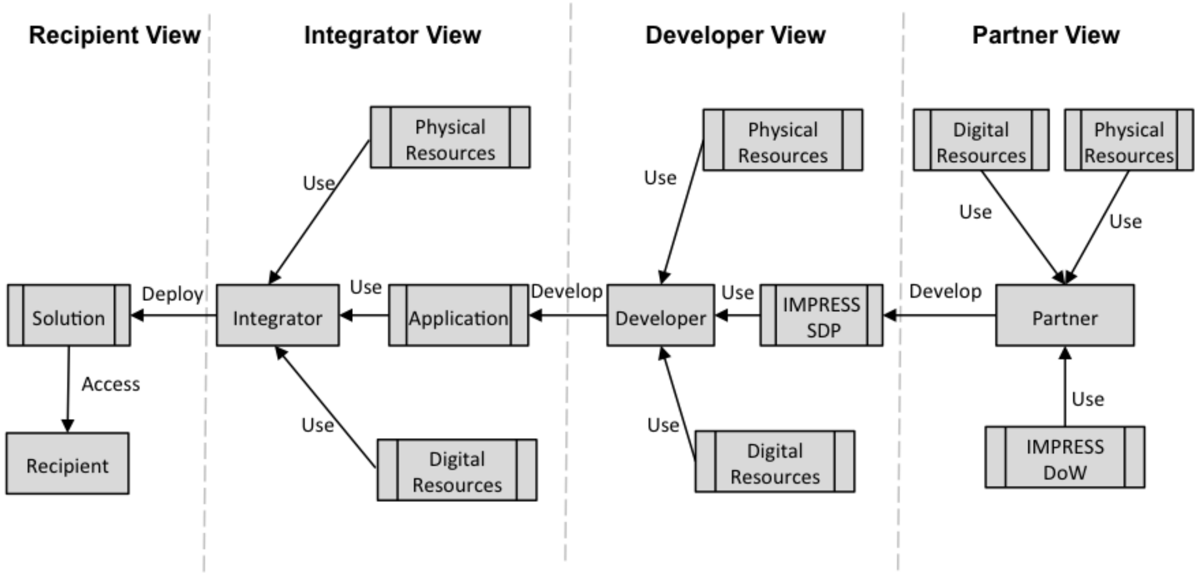


Figure 1 - IMPReSS Architecture Views

The IMPReSS software architecture, which is presented in Figure 2, has been introduced in IMPReSS Deliverable D2.2.1 (Kamienski 2014a) from February 2014 and has been improved to reflect the collaborative view of partners better. It is divided up into four views, where the Partner's view is the most complete one and therefore it is considered to represent the IMPReSS Architecture. The other views are for the Application Developer, Solution Integrator and Final Recipient. Figure 2 shows that the SDP is divided into IDE and Middleware, where 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 relational as well as NoSQL databases. Modules in the IDE component of the IMPReSS Platform might have counterparts in the Middleware component and they communicate through the Middleware API. The whole set comprised of Middleware and IDE modules is called the IMPReSS Platform or IMPReSS SDP.

Regarding the need for both IDE and Middleware the current view is that whereas the Middleware components are always needed in order to characterize the use of the IMPReSS Platform, the use of the IDE components might be considered optional. In other words, since the Middleware components offer their services through a well-defined API, other third party or tailor made GUI modules may use IMPReSS services without necessarily using the IDE. Also, applications may choose not to use all Middleware modules depending on the developer's preferences. Middleware is comprised of four main modules, which have counterparts in IDE that are related to each other. In addition, IDE has a specific module for allowing the design of interfaces using Model-Driven Development. The IMPReSS Middleware API is comprised of the APIs of the other four middleware modules, namely Context Manager API, Data Manager API, Resource Manager API and Communication Manager API.

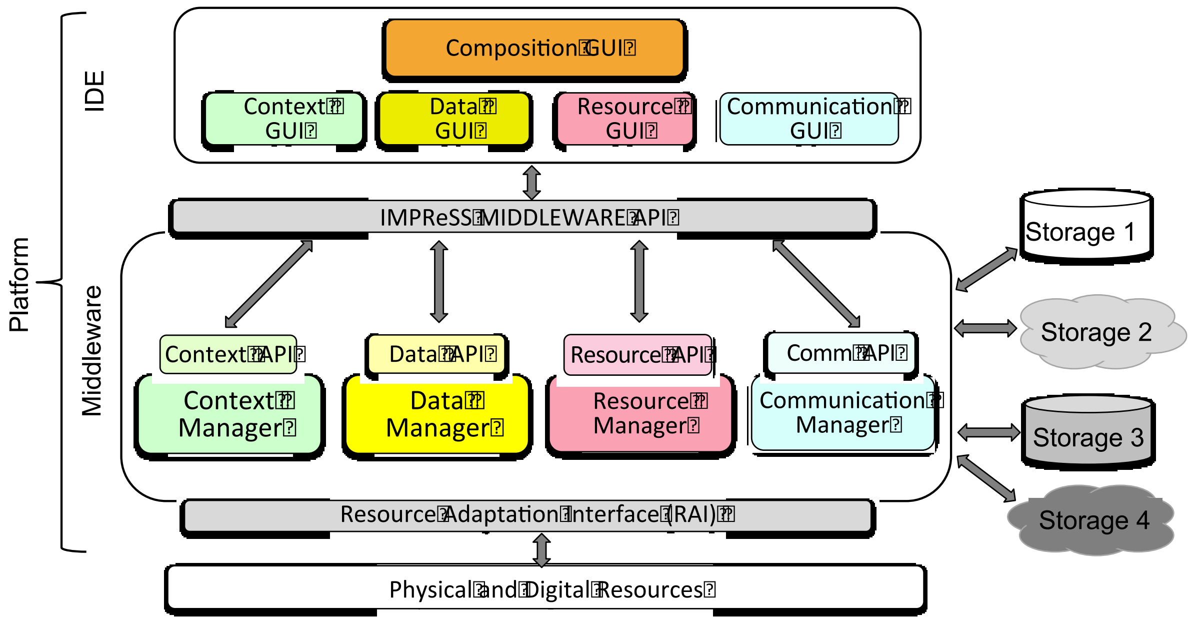


Figure 2 - IMPReSS SDP Architecture – Partner's View – Version 2

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 based on Model-Driven Development (MDD), a software engineering approach where developers create technology-agnostic models using high levels of abstraction.
- Context GUI: A graphical tool for 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 discloses 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 for 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 for 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:

- Context Manager: This module encompasses all background software components that a typical context-aware middleware offers to its users (Perera 2014), such as context templates, context models, context reasoning engine, and algorithms for sensor and data fusion. It also interacts with storage modules to be able to store and retrieve context data. 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.
- 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).

4. Background

This section introduces that concept of context-aware computing focusing on the two key points, which are context model and context reasoning.

4.1 Context-aware Computing

Context awareness is a core feature of ubiquitous and pervasive computing systems and has been around for more than 20 years (Perera 2014). In the last decade, context-aware computing evolved from typical pre-Internet platforms such as desktops to web-based applications and mobile computing, surfing on the pervasive/ubiquitous computing wave and finally is now considered as an important component for the Internet of Things (IoT).

Context-aware systems can be defined as systems that are able to adapt their behavior to the current context conditions without explicit user intervention (Baldauf 2007). Makris et. al also define context awareness as the ability of computing systems to acquire and reason about the context information and adapt the corresponding applications accordingly" (Makris 2013). The term context usually refers to locations, but actually can comprise different information used to characterize the situation of entities that play an important role in the interaction of user and application (Dey 2007). Entities are usually classified in three categories, which are: a) places, e.g. rooms, buildings, etc.; b) people, both individuals and groups, and; c) things, which are physical objects, computer components, etc. (Dey 2001).

The context life cycle might be defined as composed of four phases, as shown in Figure 3, which are context acquisition, context modelling, context reasoning and context dissemination (Perera 2014). Context needs to be acquired from a variety of sources that typically are physical sensors (temperature, lighting, presence, etc.), but also can come from virtual sensors such as social networks and calendars or logical sensors such as weather services. This context information acquired from sensors must be modeled so that it can be used meaningfully. Further, modeled context data must be processed to derive high-level context information from the raw data. This process is called reasoning. Finally, context data (both low-level and high-level) must the distributed to all applications that need it. The life cycle inks context dissemination back to context acquisition because new context acquisition will be influenced by the adaptation of system behavior resulted from the new context.

Different techniques and mechanism have been proposed for context acquisition and dissemination. However, the key components of any context-aware system are the context model and the context reasoning approach. These components are also intrinsically related to each other and must be therefore defined together. The choice of a context model is a very important step in any context-aware system because it influences the whole system and also limits the choice of a context reasoning technique.

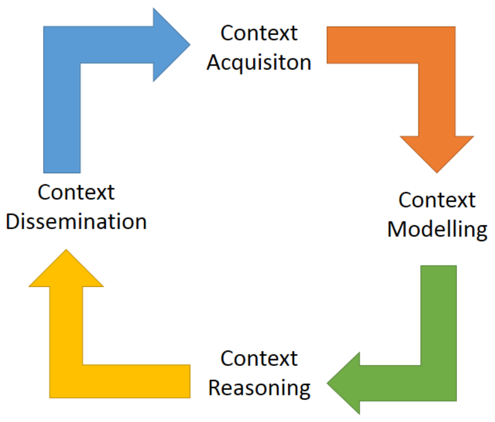


Figure 3 – Context life cycle

The bird's eye view of the macro-components presented by Figure 3 plays an important role in making it easier to understand the general activities that must be undertaken in order to build any context-aware system. Figure 4 presents a layered framework for context-aware systems (Baldauf et. al 2007), where the separation of detecting and using context is highlighted to improve extensibility and reusability of systems. The first layer is made or sensors (physical, virtual and logical) and the second layer contains mechanisms for acquiring the context data from the sensors. The raw data coming from sensors must be preprocessed to be useful for the higher levels where it is stored accordingly and managed. Context management is a very important component, since it includes the idea of controlling and taking decisions based on context information. Finally, the application is notified of the context information and uses it as it sees fit.

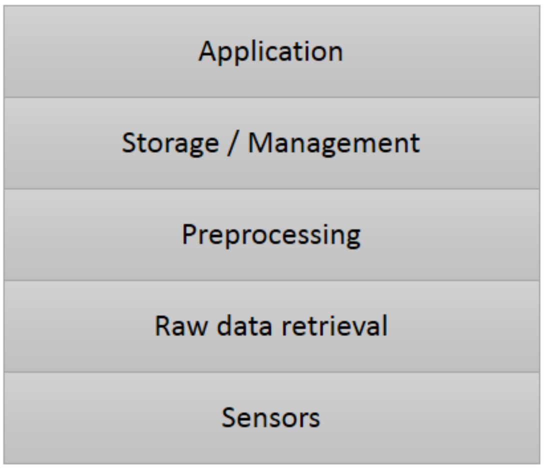


Figure 4 - Layered conceptual framework for context-aware systems

4.2 Context Modelling

A context model is needed for defining, handling, storing and distributing context data in a machine-processable form (Baudalf 2007). Different context models have been proposed and tested, such as key-value, markup, graphical, object oriented, logical and ontology-based models (Makris 2013). Key-value techniques are the simplest ones and model context information as key-value pairs in different formats such as text and binary files. Markup Scheme Modelling improves the basic key-value techniques with tag-based modeling methods. Graphical Modelling adds relationships to context modelling using graphical representations such as Unified Modeling Language (UML). Object Oriented Modelling extends the object-orientation paradigm for context modeling, using class hierarchies and relationships, allowing data encapsulation and code re-usability. Logic Based Modelling uses facts, expressions, and rules to represent context information, providing more expressive context modeling methods compared to the previous modeling approaches but the lack of standardization reduces its applicability. Finally, Ontology Based Modelling organizes context information into ontologies using semantic technologies, such as Resource Description Framework (RDF) and Web Ontology Language (OWL).

Table 1 compares the different techniques for context modelling, presenting advantages, disadvantages and application guidelines, adapted from (Makris 2013). We highlight the object-oriented and ontology-based modeling techniques, as they are clearly more suitable to our needs than the other ones.

Table 1 – Context Modelling and Representation Techniques (Makris 2013)

Technique	**Pros**	**Cons**	**Applicability**
Key- Value	• Simple • Flexible • Easy to manage when small in size	• Strongly coupled with applications • Not scalable • No structure or schema • Hard to retrieve information • No way to represent relationships	Can be used to model limited amount of data such as user preferences and application configurations

		• No validation support • No standard processing tools are available	
Markup Scheme	• Flexible • More structured • Validation possible through schemas • Processing tools are available	• Application depended as there are no standards for structures • Can be complex when many levels of information are involved • Moderately difficult to retrieve information	Can be used as intermediate data organization format as well as mode of data transfer over network
Graphical	• Allow relationships modelling • Information retrieval is moderately easier • Different standards and implementations are available • Validation possible through constraints	• Querying can be complex • Configuration may be required • Interoperability among different implementations is difficult • No standards but governed by design principles	Can be used for long term and large volume of permanent data archival
Object Based	• Allows relationships modelling • Can be well integrated using programming languages • Processing tools are available	• Hard to retrieve information • Semantic reasoning through reflection is quite complex, and some languages even do not support reflection • No standard but governed by design principles • Lack of validation	Can be used to represent context in programming code level Allows context runtime manipulation Very short term, temporary, and mostly stored in computer memory
Logic Based	• Allows to generate high-level context using low-level context • Simple to model and use • Support logical reasoning • Processing tools are available	• No standards • Lack of validation • Strongly coupled with applications	Can be used to generate high-level context model events and actions, and define constrains and restrictions
Ontology Based	• Supports semantic reasoning • Allows more expressive representation of context • Strong validation • Application independent and allows sharing • Strong support by standardizations • Sophisticated tools available	• Representations can be complex • Information retrieval can be complex and resource intensive • Ontology-based reasoning is slow	Can be used to model domain knowledge and structure context based on the relationships defined by the ontology

4.2.1 Object-Oriented Context Modelling

Modeling context information using object-oriented techniques offers all advantages of this well-known and well-accepted programming and modelling paradigm. The existing approaches use objects to represent different types of context (e.g. location, temperature) making it possible to encapsulate, process and represent context data. Context information is represented as a set of entities that in turn describe physical or conceptual objects such as a person or a communication channel. The properties of the entities are represented by attributes.

One of the main advantages of the object-oriented approach for context modelling is the widespread use of object-oriented languages, such as C++, Java and Python. Programmers have been learning object-oriented techniques in the last 20 or 30 years and they master software development using this approach. Therefore, using it to model context information is a very natural choice, because it inherits all advantages of the object-oriented paradigm, such as encapsulation, inheritance, abstraction and re-utilization. The object-oriented approach allows to better organize knowledge and to simplify the creation of rules for context manipulation. Additional advantages and disadvantages are shown in Table 1.

4.2.2 Ontology-based Context Modelling

An efficient model for manipulation, sharing and storage of context data is essential for context-aware systems. Generally speaking, ontologies can be used for making communication possible or easier among different people, applications, systems, which are part of the same knowledge domain, but not always share the same concepts related to the components of that domain. The lack of shared understanding may impose difficulties for knowledge interoperability, reuse and sharing, which is important given the large variety of computing methods, paradigms, languages and tools. Ontologies have been extensively used for context modeling and representation because they allow knowledge sharing among humans and software agents, in addition to make it possible to reuse knowledge and to easily integrate with reasoning engines. The most common language for expressing Ontologies is OWL (Dean 2004) and rules are specified in SWRL (Horrocks 2004).

Many authors consider ontologies the most adequate approach for representing context because of its expressiveness that allows modelling very complex environments, and reasoning engines can build very complex reasoning sequences. However, ontology-based reasoning has some problems that make its use in practical systems a challenging endeavor. Ontology-based languages, such as OWL, are difficult to work with, with does not contributed for its acceptance among software developers. Also, most of the ontology-based reasoning has a very poor performance. Additional advantages and disadvantages are shown in Table 1.

4.3 Context Reasoning

Context reasoning refers to information or knowledge that can be inferred or deducted from analyzing data and combining different context information, so that low-level context can be used to generate high-level context information (Makris 2013). The need for reasoning can also be explained by two characteristics of raw context: imperfection and uncertainty. The process of context reasoning may be divided into three broad phases, which are context preprocessing, sensor data fusion and context inference. Preprocessing is needed for adapting low-level context information into some format that may be correctly processed by the other phases. Data fusion coming from different sensors at different times frequently need to be combined with other to form a more elaborate or to compute some statistics that make it more useful (Pramudianto 2014). For example, instead of analyzing individual context information coming from a single sensor, some context management system may prefer to average values of various sensors spread in a place such as a temperature of a room for a given time can be calculated from an average value of several thermometers in that room over the last one minute. Lastly, context inference is the process of using existing context data to create knew knowledge, based on physical, virtual, and logical sensors, as well as using rules entered by system administrators. For example, a rule may state that air conditioners must be turned off and lights must be switched off when there is no one in a room.

There are a large number of different context reasoning decision models, such as machine learning techniques, evidence or Dempster-Shafer theory, ontology-based and rule-based. These models are not specific to context reasoning but commonly used in different fields in computing and engineering. Two or the more common techniques are rule-based and ontological reasoning.

4.3.1 Rule-based Context Reasoning

Rules are a very simple and straightforward method of reasoning and therefore it is very popular and there are many tools available. Rules are usually structured in a format similar to a selection command (IF-THEN-ELSE) of a programming language. Expressing context reasoning as rules makes it simpler for humans to better understand the behavior of the system and the outcomes and the nature of actions that may be taken when a rule is successfully evaluated and executed. Rules can be generated manually or even automatically depending on the occurrence of some important and recurrent events. Whenever reasoning is needed, the system invokes the reasoning engine that evaluates all stored rules searching for matches. When a rule is selected, the commands inside their THEN or ELSE clauses are executed.

Rule-based reasoning can be used with object-oriented or ontology-based modeling.

4.3.2 Ontology-based Context Reasoning

Ontological reasoning is based on description logic, which is a family of logic based knowledge representations of formalisms, and it supported by two common representations of semantic web languages: RDF and OWL. The main advantage of ontological reasoning is that it can be easily integrated with ontology modelling. On the other hand, a well-known disadvantage is that it is not able to deal with imperfect context data (missing or ambiguous data, for instance). Missing values happen due to the inherent process of data acquisition, where data is lost very frequently for a variety of reasons. The use of rules can enhance ontological reason by providing default data to be used in place of imperfect ones. For example, rules can replace missing values by predefined values defined by the application.

5. Context Entities and Templates

The use of entities and templates for energy efficiency context management is aimed at making it easier to understand, model and program the context-awareness features of the IMPReSS project. This section provides an update of deliverable D6.1 (Kamienski et al. 2014), based on the most recent developments of context model and reasoner.

The design of context templates is characterized by context entities, their relationships and their attributes, which play a key role in the architecture of the Context Manager presented in section 6.

5.1 Entities, Relationships and Templates

Eight entities of typical scenarios for context-aware management for making efficient use of energy in buildings have been identified and each one generated a variety of templates: Subject, Resource, Place, Fusion, Rule, Action, Activity and Notification. Entities have templates that are involved in the process of managing energy efficiency context. In the following sections, the templates (models) for entities are presented, which are used for designing the context manager module.

5.2 Subject Template

Subjects, i.e., people are of paramount importance for energy efficiency in buildings, since the former inhabits the latter. The use of the entity Subject might be important in a giving IMPReSS application for making it clear the role played by people. However, Subject can be modeled as a Resource in some cases, because people will be always be identified through a resource, either a sensor or a device (e.g. a smartphone that identifies someone, which in this case might be also considered a sensor).

A subject template for IMPReSS is specified by the following information:

- Id: a unique identifier for that particular participant in a given scenario;
- Role: the role played by someone in a particular scenario;
- Function: an explanation of the function associated to the role someone is playing; one subject (person) can play different roles in different situations. For example, in a university someone can be both a student and an employee at the same time, but depending on the context they will interact in a different way with certain context features.

5.3 Resource Template

Resources are central for context management, because they interact directly with subjects and are influenced by them.

A device template for IMPReSS is specified by the following information:

- Id: a unique identifier for all resources in a particular scenario;
- Name: resource name;
- Class: the class of the resource, which defines its role in the scenario for context management, such as devices, equipments, sensors and actuators;
- Function: a description of the resource and its role in the scenario;
- Measurement: Additional information specifying exactly what is being measured in case the resource is a sensor.

5.4 Place Template

Places are the locus where subjects and resources interact and where context situations occur and must be manage by IMPReSS. Places may include offices, rooms, halls, corridors, atria, etc.

A place template for IMPReSS is specified by the following information:

- Id: a unique identifier for a place in a particular scenario; places may host different scenarios, and in each one they may be used in a different way;
- Name: the name of a place;
- Type: an attribute used to better specify the place according to the conditions people find and the way they interact with the place.
- Function: the function of the place in a particular scenario;
- Volume/area: an attributed used to better characterize the place according to its capacity of holding people, equipments and its needs of lightening and temperature.
- Condition: typical operation conditions encountered by people and managed by the IMPReSS-enabled context-aware application.

5.5 Fusion Template

Sensor data Fusion is highly needed by any context management system, since a high volume of context information may be produced by sensors and must be deal by the system at any time. Fusion is needed for reducing the amount of data available to the system according to some criteria, falling into a class of applications called Complex Event Processing (CEP) (Wu et al. 2006).

A fusion template for IMPReSS is specified by the following information:

- Id: a unique identifier for a particular fusion criterion;
- Name: the name of the fusion criterion;
- Sensor: a sensor, a set of sensors of the same type in different places or a set of sensors of different types that provide data for a particular fusion criterion.
- Algorithm: algorithm, or formula, used by the system to fuse sensor data.

5.6 Rule Template

Rules describe how the context management system deals with situations that occur frequently, according to pre-specified parameters and conditions. A rule-processing engine receives a set of parameters and matches them to the rules previously stored by the administrator (developer, integrator or recipient, using the IMPReSS terminology) for selecting one or more rules for processing.

A rule template for IMPReSS is specified by the following information:

- Id: a unique identifier for a particular context rule;
- Name: the name of a rule;
- Subject: the subject, or people, who request a rule application or who are affected by a rule;
- Resource: a resource of a set of resources that are considered by a particular rule;
- Action: action or set of actions to be performance in case a given rule is processed
- Fusion: fusion criteria to be used by a rule
- Condition: a specific condition that must occur in order for a rule to be processed;

- Result: the result of the rule processing after the action has been executed.

5.7 Action Template

An Action, or set of actions, is executed as a result of firing a Rule. Actions may perform different tasks for dynamically adapting the behavior of some digital of physical resources for changing some system configuration or sending an instruction for an actuator, e.g. changing the temperature of the air conditioner or heater.

An action template for IMPReSS is specified by the following information:

- Id: a unique identifier for a particular action;
- Name: the name of the action;
- Description: textual description of the action;
- Algorithm: the program or configuration to be made;
- Resource: actuators or other systems to be affected by the action.

5.8 Activity Template

An Activity is a scheduled activity that will happen in a given Place according to a timetable and that will affect one or more Resources. For example, an activity in a Theater may be a play, a concert or a rehearsal and in a university it might be a lecture or a meeting.

An activity template for IMPReSS is specified by the following information:

- Id: a unique identifier for a particular activity;
- Name: the name of the activity;
- Data/Time: Date and time of the scheduled activity;
- Place: place where the activity will happen;

5.9 Notification Template

Notification is the feedback sent to the application of any Action performed by the reasoner as a result of firing a rule. Applications must register notifications they want to receive from the context manager.

A notification template for IMPReSS is specified by the following information:

- Id: a unique identifier for a particular notification;
- Name: the name of the notification;
- Action/Rule: Action and rule that generated that notification.

5.10 Entity Relationships

Figure 5 depicts the relationships among the eight entities, which are:

- Place has Resource: A Resource is always located in a Place;
- Resource fires Rule: Rules are fired by Resource usage;
- Fusion uses Resource: Fusion criteria combines data coming from sensors, which are classified as Resources;
- Fusion influences Rule: Rules are influenced by sensor data that are combined by a Fusion criteria;

- Rule relates to Place: Rules affect and are affected by Places;
- Rule affects Subject: Subjects are directly affected by Rules in many different ways;
- Subject interacts with Resource: Subjects use Resources and are affected by them. Also, resources as sensors monitor Subjects;
- Rule performs Action: when a Rule is fired, its processing results in one or more Actions to be performed;
- Activity fires Rule: the schedule of Activities may fire Rules regardless of data coming from sensors;
- Action sends Notification: the execution of an Action may result in a Notification being sent back to the application.

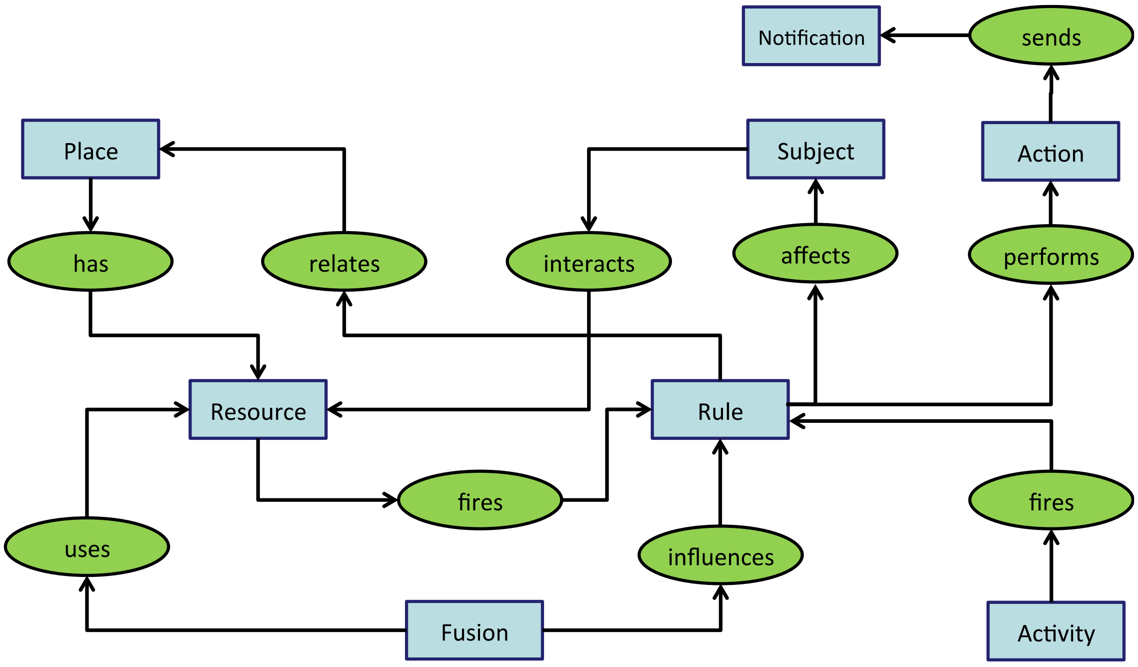


Figure 5 – IMPReSS Context Entities and Relationships

6. Context Management Framework Architecture

According the IMPReSS Software Architecture presented in section 3, the Context Manager is a module of the IMPReSS Middleware, which is in charge of providing background software components that a typical context-aware middleware offers to its users, such as context templates, context models, context reasoning engine, and algorithms for sensor and data fusion. This session focuses explicitly in the architecture of the Context Manager, based on the Context Templates presented in section 5 and its interactions with the other IDE and Middleware modules.

6.1 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. This effort resulted in the content presented in a summarized form in section 4. 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.

6.2 Context Manager Architecture

Figure 6 depicts the Context Manager Architecture and its relationships with other components of the IMPReSS Architecture. It can be roughly divided into two main planes inside the IMPReSS Middleware, namely control plane and event plane. This architecture also includes modules that are in the IMPReSS Middleware Interface and in the Resource Adaptation Interface, according to Figure 2, where the Context API and Communication Proxy reside, respectively.

- Event Plane: it comprises the two main components that operate in real time, i.e. the fusion and the reasoner module, receiving and processing data coming from sensors and sending commands to actuators.
- Control Plane: it comprises modules for context template configuration, storage and notification, which are needed for the features of the event plane to work properly.
- IMPReSS Middleware API: the key module in the IMPReSS Middleware API as far as context-awareness is concerned is the Context API, but the interface to the Data Manager is also represented, as Data Proxy and the Context Notification feature.
- Resource Adaptation Interface (RAI): this module encapsulates the communication with physical and digital resources via the Communication Proxy.

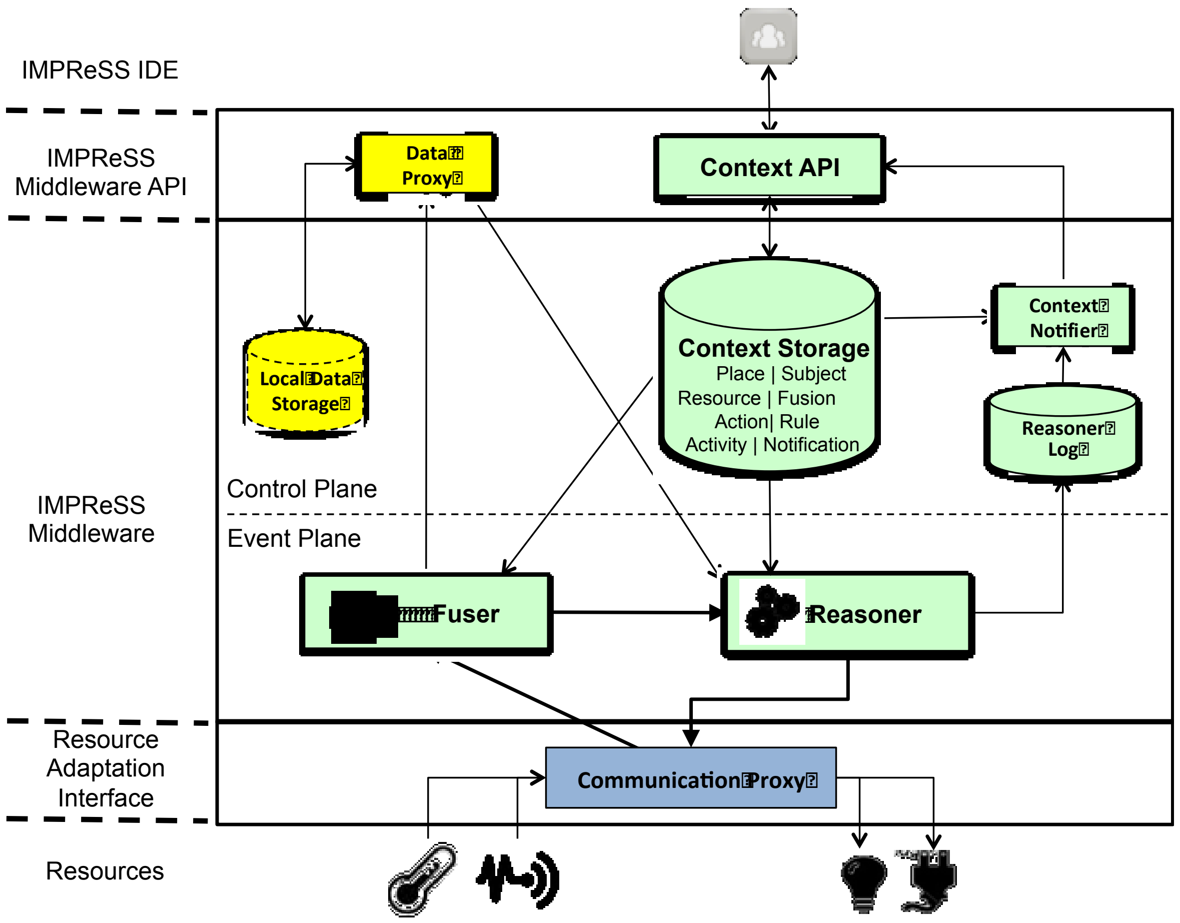


Figure 6 – IMPReSS Context Manager Architecture

The IMPReSS Context Manager modules are:

- Context API: The Context API is part of the IMPReSS Middleware API and exposes an interface, allowing other modules, both belonging to the IDE and Middleware, to interact with the Context Manager. For instance, for CRUD (create, read, update, delete) operations related to context templates. Through the Context API the entity templates are configured in the Context Storage. Please notice that the Context Manager assumes it will be able to successfully find and establish communication with Resources, i.e., sensors and actuators. The Context Manager learns about Resources through the Context API used by the application in the IMPReSS IDE. In turn, the application will learn about Resources through the Resource Manager (section 3), which discovers them from the environment.
- Data Proxy: This module encapsulates the communication with a data storage and retrieval module, either raw data coming from sensors or processed data produced by a fusion operation. It is defined as a small stub module that hides the details of using the Data API to the Context Manager, or alternatively it provides access to an internal Local Data Storage in case communication with the Data Manager is not available.
- Local Data Storage: This module implements an internal data storage feature for situations where using the Data Manager (section 3) is not possible or even desirable. It stores all data coming from sensors and also data fused by the Context Manager. It is a local database for making it possible to have prompt access to historical data.
- Context Storage: This module is responsible for storage and retrieval of context entity templates, via the Context API. According to section 5, eight entities have been identified for the Context Manager and are dealt with by the Context Storage, namely Subject, Resource, Place, Fusion, Rule, Action, Activity and Notification.

- Reasoner: The Context Reasoner is the piece of software able to infer logical consequences from a set of asserted facts, as introduced in section 4.3. Also according to section 6.1, the IMPReSS Context Manager is based on an object-oriented model and a rule-based reasoning approach. The Reasoner performs its function by reading entities from the Context Storage, i.e. Entity Templates such as Rule, Place, Resource and Action. Having all entities, whenever it is invoked with a set or parameters it searches the entire set of rules for a match, i.e., a particular rule that matches the parameters and as a consequence will be executed. In some situations the Reasoner may find two or more rules that match the parameters, i.e. there may be a rule conflict. Whenever a conflict happens, the Reasoner must select only one rule to be executed based on some conflict resolution mechanism. The Reasoner is invoked by the Fuser whenever Fusion criteria are met. 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. The Reasoner can also receive historical data from the Data Proxy that may be needed by some rules.
- Reasoner Log: This module stores all actions taken by the Reasoner, for notification and auditing purposes.
- Context Notifier: Whenever an application requires a notification of certain actions (e.g. turning on or off certain devices) taken as a result of the successful processing of a rule by the Reasoner, it can configure the Notification entity template. The Context Notifier will monitor all actions configured to be notified in the Notification template and send notifications back to the application through the Context Notification feature inside the Context API. Context Notification may be implemented using a mechanism based on callback functions that are registered by the application when configuring the Notification Entity Template.
- Fuser: This module is responsible for data fusion, i.e. a set of techniques that combine data from multiple sources such as sensors and gather that information in order to achieve inferences, which will be more efficient and potentially more accurate than if they were achieved by means of a single source. 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 in a data storage using the Data Proxy. Multiple fusion criteria may be active concurrently and therefore this module plays a key role for the performance of the Context Manager, because in a real scenario hundreds or thousands of sensors may send data values with a high frequency. The Fuser reads the fusion criteria from the Context Storage and that is how it finds out which data must be requested from the Communication Proxy. Whenever a new fusion criteria is configured the Fuser registers the corresponding resources to be monitored, e.g. sensors, in the Communication Proxy and the latter starts sending data to the former.
- Communication Proxy: This module encapsulates the communication with resources, i.e. sensors and actuators. It can be implemented as a small stub module for hiding the details of both the Communication Manager and Resource Manager (Figure 2). Alternatively, it can directly implement a communication protocol that interacts with the resources, for instance using a machine-to-machine communication protocol typically used in the Internet of Things (Borgia 2014). The Communication Proxy may also be represented as part of the IMPReSS Middleware API but from the Context Manager point of view it is an internal middleware interface. Also, it intermediates communication with lower level resources and thus it makes it easier to understand the architecture and the way the modules interact with each other.

6.3 Implementation Guidelines

The implementation of the Context Manager Architecture described in section 6.2, based on an object-oriented context modeling and rule-based context reasoning, has already started aiming at developing a preliminary prototype for evaluating the adequateness of using different existing technologies. The implementation guidelines shown in Figure 7 is a result of extensive research about existing Complex Event Processing (Wu et al. 2006), Rule Engines (Sun et. al 2014), protocols for the Internet of Things (Aztoria e. al 2010) and related technology.

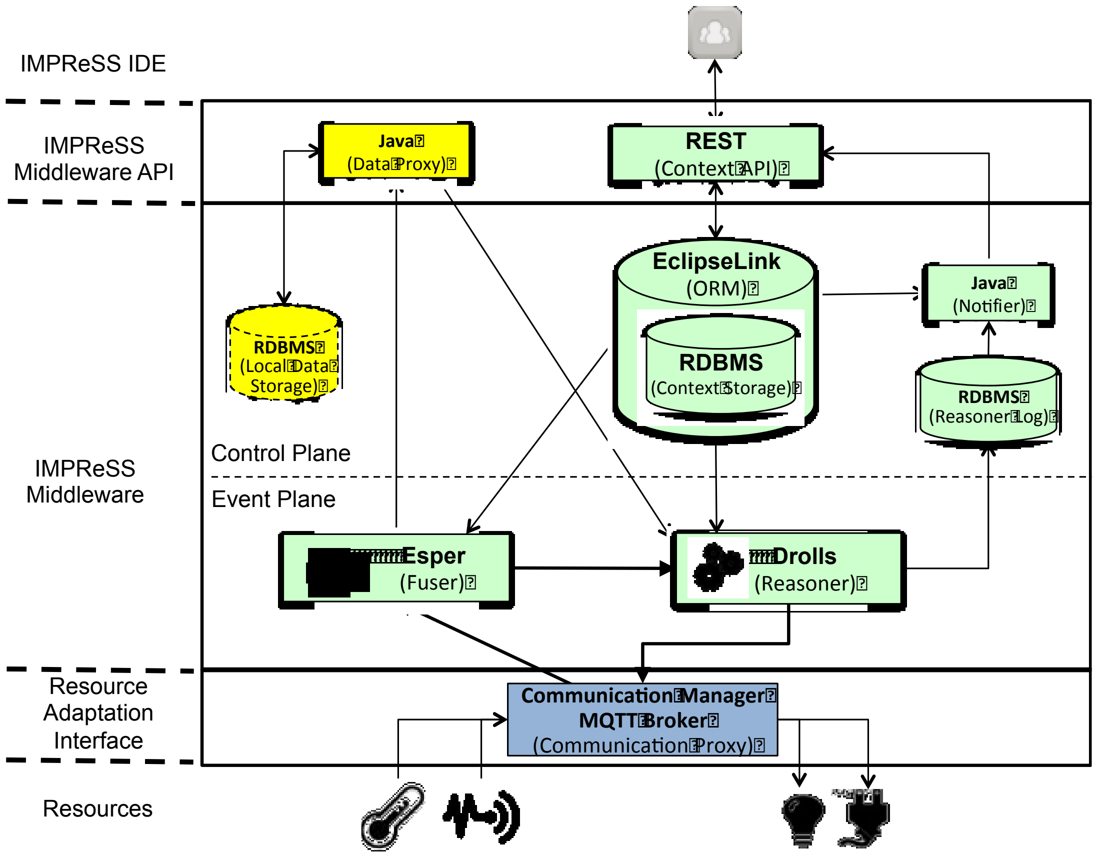


Figure 7 – Implementation guidelines for IMPReSS Context Manager Architecture

The implementation of the Context Manager Architecture will be based on free or open-source software systems. Some strong candidates for implementing the components of the Context Manager are:

- Context API: RESTful Web Services (Pautasso et. al 2008), like the other middleware APIs.
- Data Proxy: a specially developed Java program that encapsulates communication with the Data API or Local Data Storage.
- Local Data Storage: any Relational Database Management System (RDBMS), such as PostgreSQL[1] or MySQL[2].
- Context Storage: any RDBMS with an Object-Relational Mapping (ORM) system, such as EclipseLink[3] or Hibernate[4]. The use of an ORM is needed because on the one hand data is structure and therefore suitable for a RDBMS. On the other hand, the context modeling is object-oriented, so that a mapping between both models is required.
- Reasoner: Drools[5] is a Rule Engine, which is classified as a Business Rules Management System (BRMS)[6]. Two components have been evaluated for the Context Reasoner, Drools Expert, a business rules engine and Drools Workbench (formerly known as Guvnor) a web graphical interface for rule authoring and management.

[1] http://www.postgresql.org
[2] http://www.mysql.com
[3] http://eclipse.org/eclipselink
[4] http://hibernate.org
[5] http://www.drools.org
[6] http://www.drools.org

- Reasoner Log: any RDBMS.
- Context Notifier: a specially developed Java program that must work together with the Reasoner Log, either configuring triggers or polling the database for actions that must be reported to the application.
- Fuser: Esper[7] is a component for enabling complex event processing (CEP) and event series analysis. It enables rapid developments of applications that process large volumes of incoming real-time and historical messages or events. Esper can filter, analyze, and fuse events in various ways, configurable through an SQL-like Event Processing Language (EPL). In Esper, Fusion criteria are called streams.
- Communication Proxy: a specially developed Java program that encapsulates the communication with resources, either the Communication Manager or a MQTT[8] broker.

6.4 Context Manager Processing Steps

In order to understand the interworking of the Context Manager components (Figure 6) easier, this section sheds some light into the data flow and processing steps involved in the process of analyzing sensor data and sending commands to actuators. Figure 8 introduces the processing steps and data flow for the Context Manager, based on the implementation guidelines with software modules presented in Figure 7. Not all components are detailed here, but only those that are in the critical path in the event plane. The sequence of steps is:

1. Sensors send measured data through the MQTT broker;
2. A preprocessor receives data values from the MQTT broker and adapts them for being processed by Esper. Please notice that the preprocessor is part of the Fuser.
3. The preprocessed data is delivered to Esper;
4. Esper applies fusion criteria, using its Complex Event Processing engine.
5. Whenever an Esper triggers a result, it is delivered to Drools for searching rules to decide actions to be taken;
6. Actions resulting of Drools rules go through a postprocessor and sent to MQTT;
7. Actions commanded by Drools are delivered to actuators to be enforced.

Fused data values produced by Esper are stored in the Data Storage and actions taken by Drools are stored in the Reasoner Log. In addition to the data coming from sensors, Esper and Drools are configured by data coming from entities stored in the Context Storage. All actions registered for notifications are sent back to the application.

Next we exemplify the processing steps and data flow for the Context Manager with three examples, of controlling temperature, lighting and a water pump.

[7] http://esper.codehaus.org
[8] http://mqtt.org

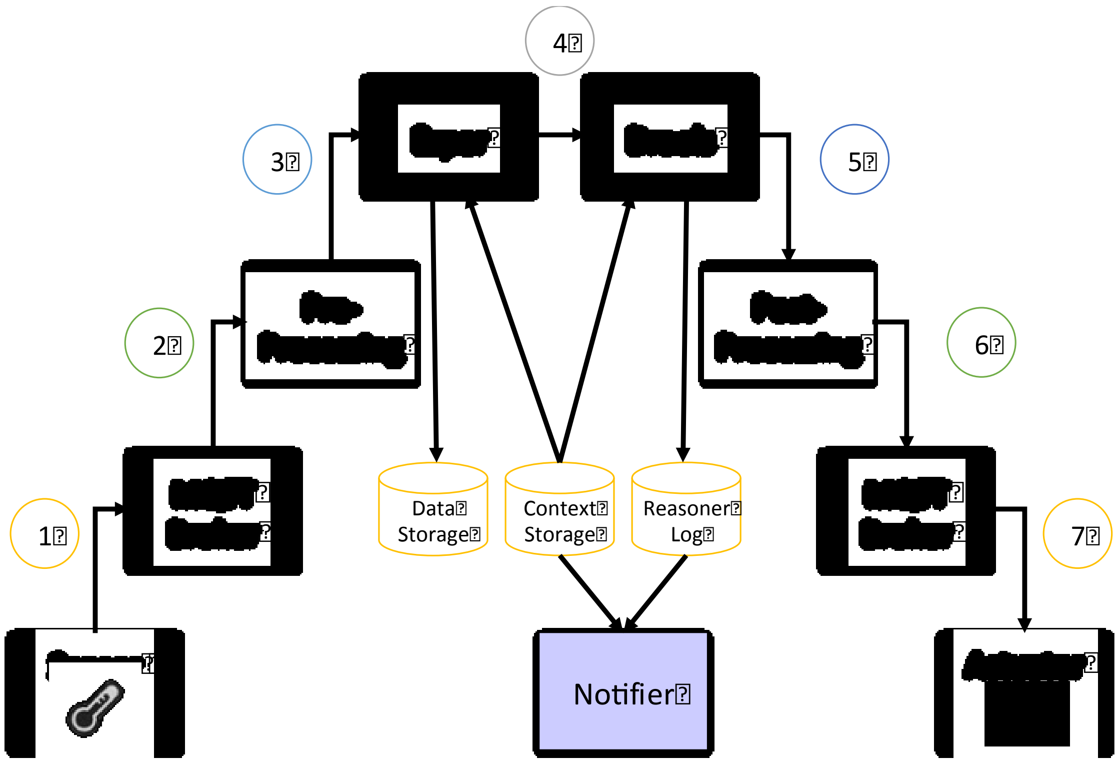
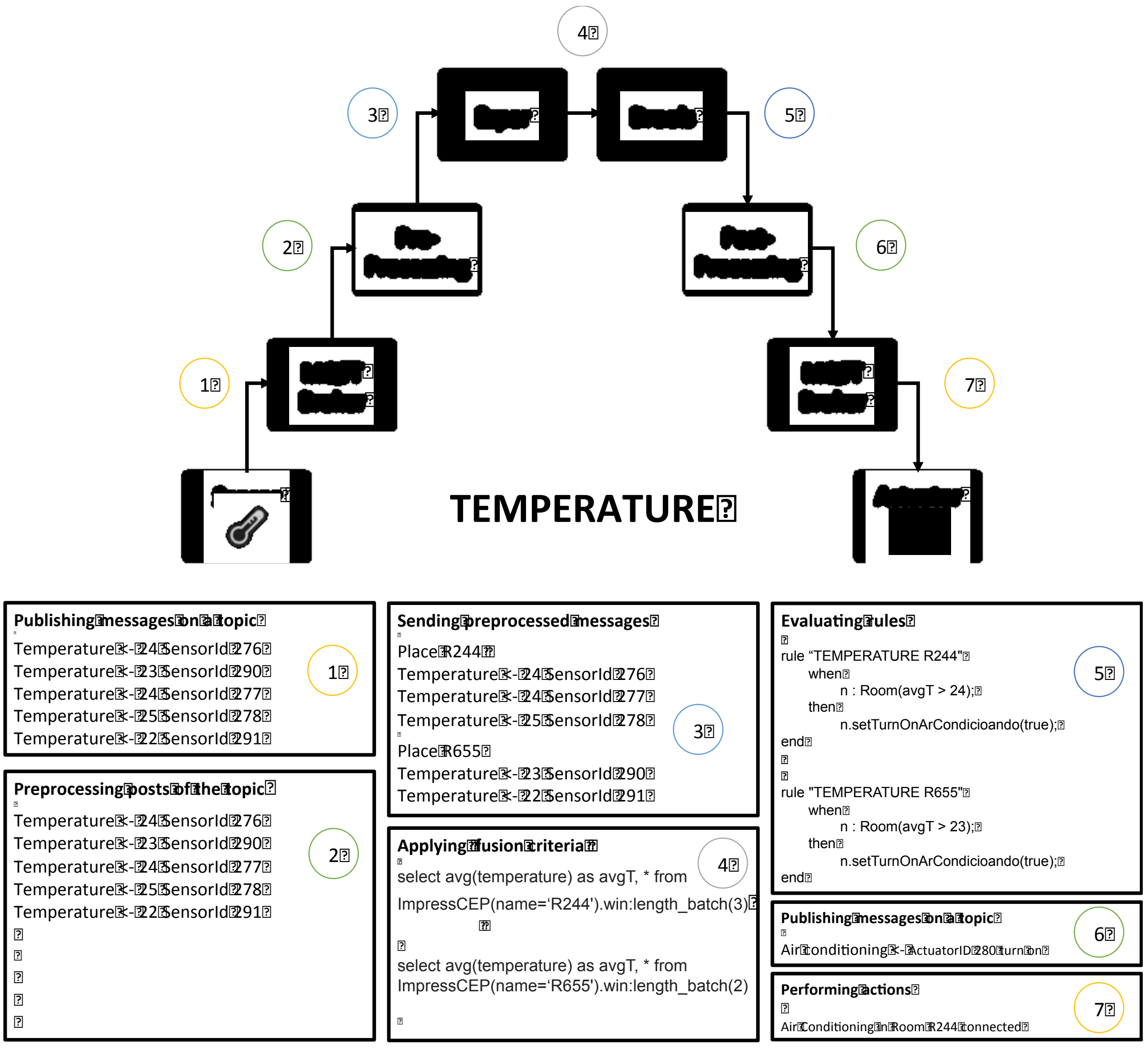


Figure 8 – Processing Steps and Data Flow for the Context Manager

Figure 9 depicts a scenario of automatic control of temperature. The data values measured by a variety of sensors are sent, via MQTT, to the preprocessor that groups the measurements by Place and forward them to Esper, which in turn applies a Fusion criterion that computes an average. Whenever a new average is available, Esper invokes Drools, that now searches all Rules stored in its database that match the fused data received as a parameter, i.e. the average temperature. Drools finds a rule for turning on the air conditioner in that Place. Finally, that action is interpreted by the postprocessor and sent via MQTT as a message commanding an actuator to turn on the air conditioner. Please notice that Esper is required in this scenario because hundreds or thousands of fusion criteria may be processed simultaneously, as long as there are different Places and resources eligible to be controlled by the IMPReSS Application.



Publishing messages on a topic 1

Temperature - 24 SensorId 276
Temperature - 23 SensorId 290
Temperature - 24 SensorId 277
Temperature - 25 SensorId 278
Temperature - 22 SensorId 291

Preprocessing posts of the topic 2

Temperature - 24 SensorId 276
Temperature - 23 SensorId 290
Temperature - 24 SensorId 277
Temperature - 25 SensorId 278
Temperature - 22 SensorId 291

Sending preprocessed messages 3

Place R244
Temperature - 24 SensorId 276
Temperature - 24 SensorId 277
Temperature - 25 SensorId 278

Place R655
Temperature - 23 SensorId 290
Temperature - 22 SensorId 291

Applying fusion criteria 4

```
select avg(temperature) as avgT, * from
ImpressCEP(name='R244').win:length_batch(3)

select avg(temperature) as avgT, * from
ImpressCEP(name='R655').win:length_batch(2)
```

Evaluating rules 5

```
rule "TEMPERATURE R244"
    when
        n : Room(avgT > 24);
    then
        n.setTurnOnArCondicioando(true);
end

rule "TEMPERATURE R655"
    when
        n : Room(avgT > 23);
    then
        n.setTurnOnArCondicioando(true);
end
```

Publishing messages on a topic 6

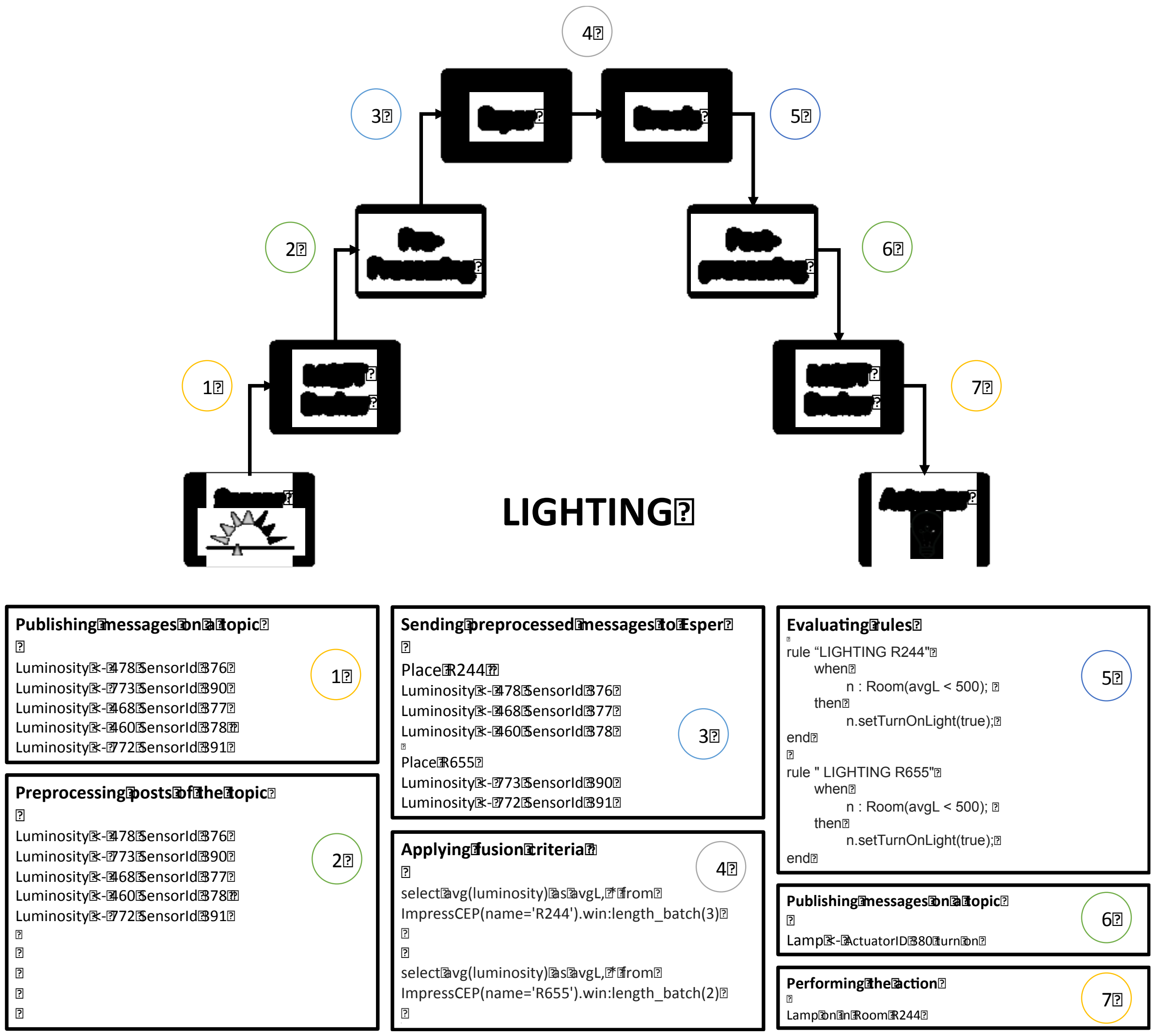
Air conditioning - ActuatorID 280 turn on

Performing actions 7

Air Conditioning in Room R244 connected

Figure 9 – Processing steps and data flow – Temperature Control

Figure 10 depicts a scenario of automatic control of lighting in a given Place. The data flow and processing steps are the same for the temperature scenario. The data values with light intensity are measured by different sensors and send via MQTT to the preprocessor that groups the measurements by Place and forward them to Esper, which in turn applies a Fusion criterion that computes an average. Whenever a new average is available, Esper invokes Drools, that now searches all Rules stored in its database that match the fused data received as a parameter, i.e. the average temperature. Drools finds a rule for switching on the lights in that Place. Finally, that action is interpreted by the postprocessor and sent via MQTT as a message commanding an actuator to switch on the lights.



Publishing messages on a topic 1

```
Luminosity - 478 SensorId 376
Luminosity - 773 SensorId 390
Luminosity - 468 SensorId 377
Luminosity - 460 SensorId 378
Luminosity - 772 SensorId 391
```

Preprocessing posts of the topic 2

```
Luminosity - 478 SensorId 376
Luminosity - 773 SensorId 390
Luminosity - 468 SensorId 377
Luminosity - 460 SensorId 378
Luminosity - 772 SensorId 391
```

Sending preprocessed messages to Esper 3

```
Place R244
Luminosity - 478 SensorId 376
Luminosity - 468 SensorId 377
Luminosity - 460 SensorId 378

Place R655
Luminosity - 773 SensorId 390
Luminosity - 772 SensorId 391
```

Applying fusion criteria 4

```
select avg(luminosity) as avgL, * from
ImpressCEP(name='R244').win:length_batch(3)

select avg(luminosity) as avgL, * from
ImpressCEP(name='R655').win:length_batch(2)
```

Evaluating rules 5

```
rule "LIGHTING R244"
    when
        n : Room(avgL < 500);
    then
        n.setTurnOnLight(true);
end

rule " LIGHTING R655"
    when
        n : Room(avgL < 500);
    then
        n.setTurnOnLight(true);
end
```

Publishing messages on a topic 6

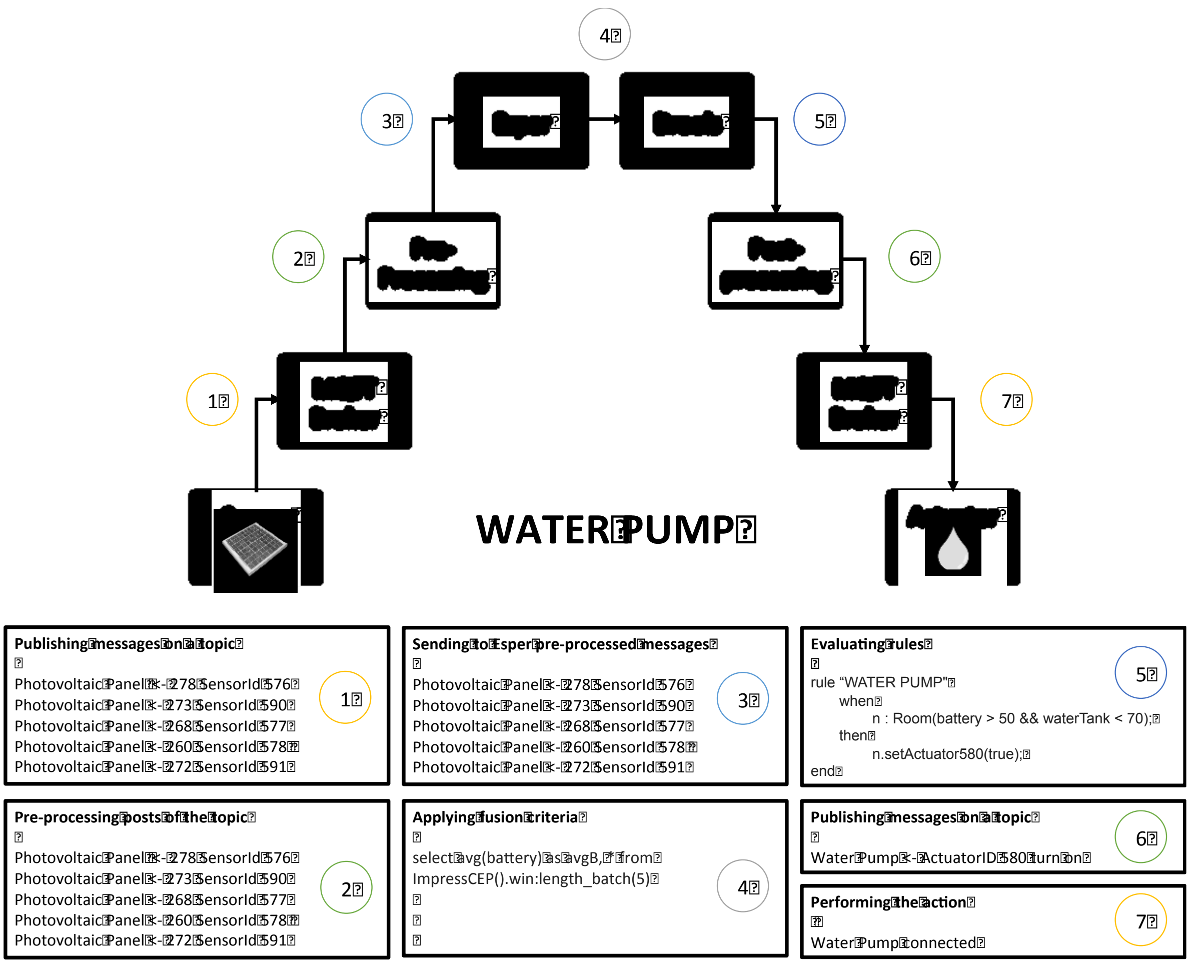
```
Lamp - ActuatorID 380 turn on
```

Performing the action 7

```
Lamp on in Room R244
```

Figure 10 – Processing steps and data flow – Lighting Control

The last example is the control of a water pump based on energy stored on photovoltaic panels, showed by Figure 11. The data values with the amount of energy in the solar energy system are measured by sensors and sent via MQTT to the preprocessor to be sent to Esper. In turn, Esper applies a Fusion criterion involving the amount of energy generated by the photovoltaic panels and the amount of energy stored in the batteries and when this is done, it invokes Drools for searching all its Rules that match criteria, that in this case eventually turn on the water pump. Finally, that action selected in the Rule is interpreted by the postprocessor and sent via MQTT as a message commanding an actuator to switch on the lights.



Publishing messages on a topic 1

Photovoltaic Panel - 278 SensorId 576
Photovoltaic Panel - 273 SensorId 590
Photovoltaic Panel - 268 SensorId 577
Photovoltaic Panel - 260 SensorId 578
Photovoltaic Panel - 272 SensorId 591

Sending to Esper pre-processed messages 3

Photovoltaic Panel - 278 SensorId 576
Photovoltaic Panel - 273 SensorId 590
Photovoltaic Panel - 268 SensorId 577
Photovoltaic Panel - 260 SensorId 578
Photovoltaic Panel - 272 SensorId 591

Evaluating rules 5

```
rule "WATER PUMP"
    when
        n : Room(battery > 50 && waterTank < 70);
    then
        n.setActuator580(true);
end
```

Pre-processing posts of the topic 2

Photovoltaic Panel - 278 SensorId 576
Photovoltaic Panel - 273 SensorId 590
Photovoltaic Panel - 268 SensorId 577
Photovoltaic Panel - 260 SensorId 578
Photovoltaic Panel - 272 SensorId 591

Applying fusion criteria 4

```
select avg(battery) as avgB, * from
ImpressCEP().win:length_batch(5)
```

Publishing messages on a topic 6

Water Pump - ActuatorID 580 turn on

Performing the action 7

Water Pump connected

Figure 11 – Processing steps and data flow – Water Pump Control

7. Conclusion

The IMPReSS System Development Platform (SDP) will need to be context aware in order to provide an efficient use of energy in buildings, in such a way to adapt its operations to the current context conditions without explicit user intervention. The key components of a context-aware system are its context modelling and reasoning approaches, which in IMPReSS will be an object-oriented and rule-based respectively. The use of entities and templates for energy efficiency context management is aimed at making it easier to understand, model and program the context-awareness features of the IMPReSS project. Eight entities of typical scenarios for context-aware management for making efficient use of energy in buildings have being identified and each one generated a variety of templates: Subject, Resource, Place, Fusion, Rule, Action, Activity and Notification.

The IMPReSS SDP is divided into IDE and Middleware, where the IDE contains a series of GUI modules and the middleware contains modules with background management responsibilities. The Context Manager encompasses all background software components that a typical context-aware middleware offers to its users, such as context templates, context models, context reasoning engine, and algorithms for sensor and data fusion. This report introduces the Architecture of the Context Manager, divided into control and event planes inside the IMPReSS Middleware. It also includes modules that are in the IMPReSS Middleware Interface and in the Resource Adaptation Interface. Some implementation guidelines are also presented, based on some literature research and hands-on experience with existing technology.

This deliverable is an important output of Task 6.3 (Context Modelling Templates) since it will influence the further implementation of the Context Manager, including to reasoning engine, which will be reported in Deliverable D6.4 (Implementation of Context Reasoning Engine). Also, it will have an influence in the context IDE module, or management GUI, that will be reported in Deliverable D6.5 (Implementation of Context Modelling Tool and Templates). The IDE is part of the IMPReSS Platform, and thus developers will use it to generate new applications (section 3). Many different interfaces may access the Context Manager as part of the IMPReSS Middleware. The proof-of-concept prototype will be based on the most up-to-date Web development techniques, given that developers will probably develop applications on workstations.

8. References

(Aztoria et. al 2010)	Atzoria, L., Ierab, A., Morabitoc, G., "The Internet of Things: A survey", Computer Networks, 54(15), pp. 2787-2805, October 2010.
(Badauf 2007)	Baldauf, M., Dustdar, S., Rosenberg, F., "A survey on context-aware systems", Intl., Journal of Ad Hoc and Ubiquitous Computing, 2(4), 2007
(Borgia 2014)	Borgia, E. (2014), The Internet of Things vision: Key features, applications and open issues, Computer Communications, 54(1), pp. 1-31, December 2014.
(Dean 2004)	Dean, M., Schreiber, G. (2004), "OWL Web Ontology Language Reference", W3C Recommendation, 2004.
(Dey 2001)	Dey, A., Abowd, G., "A conceptual framework and a toolkit for supporting rapid prototyping of context-aware applications", Human-Computer Interactions (HCI) Journal, 16(2-4), pp.7-166, 2001.
(Dey 2007)	Dey, A., Abowd, G., "Towards a better understanding of context and context-awareness", Workshop on the What, Who, Where, When and How of Context-Awareness, 2007.
(Horrocks 2004)	Horrocks, I., et al. "SWRL: A Semantic Web Rule Language combining OWL and RULEML", W3C member submission, May 2004.
(Kamienski et. al 2014a)	Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Belati, E. (2014), Analysis of Energy Efficiency Context and Sensor Fusion Algorithm, IMPReSS Consortium, Deliverable D6.1, May 2014.
(Kamienski et. al 2014b)	Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Belati, E. (2014), SDP Initial Architecture Report, IMPReSS Consortium, Deliverable D2.2.1, February 2014.
(Makris 2013)	Makris, P. et. al, "A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments' Integration", IEEE Communications Surveys and Tutorials, 15(1), 2013.
(Pautasso et. al 2008)	Pautasso, C., Zimmermann. O., Leymann, F., (2008), Restful Web Services vs. "Big" Web Services: Making the Right Architectural Decision, 17th international conference on World Wide Web (WWW 2008), pp. 805-814, 2008.
(Perera 2014)	Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D., "Context Aware Computing for The Internet of Things: A Survey", IEEE Communications Surveys & Tutorials, 16(1), First Quarter 2014.
(Pramudianto 2014)	Pramudianto, F., (2014), Implementation of Sensor and Data Fusion Module, IMPReSS Consortium, Deliverable D6.3, September 2014.
(Proctor 2012)	Proctor, M. (2012), Drools: a rule engine for complex event processing, 4th international conference on Applications of Graph Transformations with Industrial Relevance, 2012.
(Sun et. al 2014)	Sun, Y., Wu, T.-Y., Zhao, G., Guizani, M. (2014), Efficient Rule Engine for Smart Building Systems, IEEE Transactions on Computer, 99, 2014.
(Wu et al. 2006)	Wu, E., Diao, Y., Rizvi, S. (2006), High-performance complex event processing over streams, ACM SIGMOD 2006, p. 407-418