(FP7 614100)

D6.4 Implementation of Context Reasoning Engine

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: d6.4_implementation_context_reasoning_engine_v1.docx
Document version: 1.0
Document owner: Carlos Kamienski (UFABC)

Work package: WP6– Software System Engineering and Context Management
Task: Task 6.2, Task 6.3, Task 6.4
Deliverable type: P

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Carlos Kamienski	05/02/2015	Initial Version
0.4	Fabrizio Borelli and Gabriela Biondi	05/03/2015	Most content added in all sections
0.9	Carlos Kamienski	22/03/2015	First version ready for internal review
1.0	Carlos Kamienski	24/03/2015	Final version after internal reviews

Internal review history:

Reviewed by	Date	Summary of comments
Thiago Rocha (UFAM)	23/03/2015	Approved with minor comments and some corrections
Marc Jentsch (FIT)	24/03/2015	Approved with minor corrections and comments

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. 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. The architecture of the IMPReSS Context Management framework is based on an object-oriented context modelling and a rule-based context reasoning.

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. 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. 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. The specification of the Context Manager has been presented in Deliverable D6.3 and this deliverable presents implementation choices and details.

The implementation of the Context Manager uses Drools as the rule-based engine and Esper as a complex event processing solution that provides data fusion features. The context modelling approach is object-oriented but since object-oriented database did not come true, the Context Manager stores entity templates into a relational database and the mapping between objects in the programming language and tables in the database is made by EclipseLink, an Object Relational Mapping (ORM) system. Also, the Context API has been developed using RESTful Web Services.

A case study has been designed, implemented and deployed covering a typical scenario involving a university classroom, where lighting and temperature can be controlled automatically, in order to evaluate the architecture of the Context Manager and its implementation. The main idea is to demonstrate that we are able to manage the energy used in a place using the IMPReSS platform. Even though this scenario is simple and straightforward with a small number of sensors and actuators, the Context Manager is able to manage larger and more complex scenarios based on different types of sensors, actuators and devices, which makes it different from current solutions, which usually are carefully designed for controlling specific scenarios. In this scenario, the Context Manager automatically controls lighting and temperature and displays power consumption. For both situations, there are rules for adapting behavior and fusion criteria for preprocessing sensor data.

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 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 an output of Task 6.3 (Context Modelling Templates), whose main goal is to define and implement a context modelling technique and associated context reasoning approach to be used in the IMPReSS project. Deliverable D6.3 presented the Context Management Framework Architecture and Design of Context Templates. Now, this document describes the technical details of the implementation of the Context Manager, responsible for the context templates and context reasoning in the IMPReSS architecture. As the core component of the context management framework, the context reasoning engine (Context Manager) provides 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. The Context Manager is based on an object-oriented approach for context modelling and a rule-base approach for context reasoning. Also, this document presents a prototype case study representing a university classroom deployed using the Context Manager.

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 implementation of the Context Manager is a key achievement for the IMPReSS project, since it allows the IMPReSS platform to provide context-awareness features to the applications generated from it.

This deliverable is aimed at providing a clear understanding the Context Manager focusing on implementation details and choices.

2.3 Document Structure

The reminder of this document is organized in four chapters.

- Chapter 3 introduces the key technological platforms used in the implementation of the Context Manager, such as Drools and Esper.
- Chapter 4 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. This architecture has been presented initially in Deliverable D6.3.

- Chapter 5 presents the main components, choices and details of the implementation of the Context Manager.
- Chapter 6 depicts a case study based on a university classroom, which has been prototyped and is used for demonstration purposes.
- Chapter 7 presents some final remarks and the next steps.

3. Background

This section presents technologies and tools used for the development of the Context Manager.

3.1 Drools

Drools[1] is a Business Rules Management System (BRMS) provided by JBoss[2], i.e., a Rule Engine that employs a rule-based approach to implement an expert system. Drools has been chosen for the core context reasoning function of the Context Manager due to its large community, robustness and its seamless integration with object-oriented programming languages, such as Java. Drools is a suite composed of different modules:

- Drools Workbench: a web user interface for rule authoring and management, previously called Guvnor.
- Drools Expert: the actual business rules engine
- Drools Fusion: a complex event processing tool
- jBPM: a Business Process Management (BPM) suite which provides a dual focus, making the bridge between business analysts and developers
- OptaPlanner: a constraint satisfaction solver that optimizes business resource planning (i.e. automated planning)

For the implementation of IMPReSS Context Manager, Drools Workbench (actually Guvnor) and Expert has been used. Instead of Drools Fusion, Esper has been used as a Complex Event Processing tool, because the former is newer and the latter is more stable and well-known. jBPM and OptaPlanner have not been considered for the Context Manager because their key functions are not required.

3.2 Esper

Esper[3] is a component for enabling complex event processing (CEP) and event series analysis, available for Java as Esper (used in the Context Manager) and for .NET as NEsper. 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.

In other words, Esper is a specialized tool for performing sensor data fusion for the Context Manager. A different approach for fusion processing would be to build your own program. However, since Esper is a stable and very powerful tool, it is the ideal solution for a large environment with hundreds or thousands of sensors constantly flooding the Context Manager with huge amounts of raw data. In such a scenario, developing your own programs to perform fusion would be like reinventing the wheel.

3.3 REST and RESTEasy

A Service Oriented Architecture (SOA) employs "services" as the basic unit for the separation of concerns in the development of computing systems (OASIS 2006). Services can be seen as the means whereby consumers access providers' capabilities. Among other interesting features, services provide loosely coupled interaction between partners in a business process (or any other computing activity). Services that implement SOA using web technologies are usually called Web Services.

[1] http://www.drools.org
[2] http://www.jboss.org
[3] http://esper.codehaus.org

There are two main approaches for implementing Web Services (Pautasso et. al 2008). SOAP (Simple Object Access Protocol) is the most traditional approach, but due to its complexity (it is based on XML) a lighter and modern approach has been increasingly adopted, called REST (REpresentational State Transfer). REST was originally introduced as an architectural style for building large-scale distributed hypermedia systems (Pautasso et. al 2008).

REST is a collection of architectural principles and constraints for the development of distributed applications in the Web. It adopts the client/server paradigm, seeking simplicity and low coupling. REST has gained widespread acceptance across the Web as a simpler alternative to SOAP Web services. Applications conforming to the REST constraint style are usually called RESTful. RESTful systems typically, but not always, communicate over the Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE) used by web browsers to retrieve web pages and send data to remote servers.

RESTEasy[4] is a JBoss project that provides various frameworks to help you build RESTful Web Services and RESTful Java applications. It is a fully certified and portable implementation of the JAX-RS specification. JAX-RS specification (Java API for RESTful Web Services) defines a set of APIs for the development of Web Services, using the Java programming language based on the REST architectural principles of (ORACLE 2012).

3.4 JBoss Application Server

JBoss Application Server 7[5], or simply AS 7, is a version of the JBoss Java open source Application Server, a server widely used by application developers based on the Java EE platform. JBoss is around since 1999, initially known as EJBoss or EJB Open Source Server, but due to legal reasons it has been renamed to JBoss. Today it is one of the main Java application servers, competing with Apache Tomcat. JBoss is distributed under GNU license and it is completely free and implemented 100% in Java.

For the Context Manager, the choice for JBoss AS 7 was straightforward since Drools runs over it.

3.5 EclipseLink (ORM)

An Object-Relational Mapping (ORM) system is a programming technique for converting data between two different paradigms, namely an object-oriented language and a relational database. An ORM may be considered a "virtual object database" that can be used within a programming language, so that programmers may have the illusion that objects are stored directly into the database as it was a true object database, whereas in practice a relational database is in use. EclipseLink[6] and Hibernate[7] are two highly well-known ORMs, based on the Java Persistence API (JPA). Via JPA the developer can map, store, update and retrieve data from relational databases to Java objects and vice versa, i.e., it allows developers to work with object instead of with SQL statements. JPA typically defines mapping metadata through annotations in the Java class. Actually, the reference implementation of JPA is EclipseLink.

EclipseLink is the open source Eclipse Persistence Services Project from the Eclipse Foundation. The software provides an extensible framework that allows Java developers to interact with various data services, including databases and web services.

3.6 MQTT

MQTT[8] is a protocol used in Internet of Things (IoT) scenarios, ideal for use in different situations, especially in constrained environments with devices with limited resources such as sensors and

[4] resteasy.jboss.org
[5] http://jbossas.jboss.org
[6] http://eclipse.org/eclipselink
[7] http://hibernate.org
[8] mqtt.org

actuators. These scenarios are usually called Machine-to-Machine (M2M). MQTT was originally developed by IBM, and its name stands for MQ Telemetry Transport, but now it is a standard in charge of OASIS (OASIS 2014). MQTT has a client/server model, where every sensor is a client and connects to a server, known as a broker, over TCP. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery.

The key features of MQTT may be summarized as:

- MQTT is message oriented and a broker intermediates the communication of constrained devices with the software applications. Every message is a discrete chunk of data, opaque to the broker.
- MQTT is based on a publish/subscribe approach, which provides one-to-many message distribution and decoupling of applications. Every message is published to an address, known as a topic and clients may subscribe to multiple topics. Every client subscribed to a topic receives every message published to that topic.
- MQTT allows three levels of quality of service with increasing levels of delivery guarantees: "at most once", "at least once" and "exactly once"

The broker is a highly demand component of MQTT architecture and therefore it plays an important role in the communication in any M2M or IoT environment. Mosquitto[9] is an open source (BSD licensed) message broker that implements MQTT. The Context Manager uses Mosquitto.

3.7 Arduino

Arduino[10] is a tool for making resource constrained computer for sensing and controlling the physical world. It is an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on a local server of in the cloud. Arduino boards can be assembled by hand and multiple expansions (known as shields) may be coupled to a single Arduino board.

Even though Arduino is a simple, easy and cheap platform for prototyping applications that require communication with a variety of physical resources, skills are required for making it possible to implement the intended features into a highly constrained platform. The first use case prototype implementation of the Context Manager, running a university classroom use case uses Arduino for interfacing with sensors and actuators. Since it is limited, next version may be based on a more powerful platform, such as Raspberry Pi[11].

[9] mosquitto.org
[10] www.arduino.cc
[11] http://www.raspberrypi.org

4. Context Management Framework Architecture

According the IMPReSS Software Architecture, introduced in IMPReSS Deliverable D2.2.1 (Kamienski 2014b), the Context Manager is a module of the IMPReSS Middleware 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 section introduces the Context Management Framework Architecture, also known as Context Manager, introduced in IMPReSS Deliverable D6.3 (Kamienski 2014c). The Context Manager is based on object-oriented context modeling and rule-based context reasoning.

Figure 1 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.

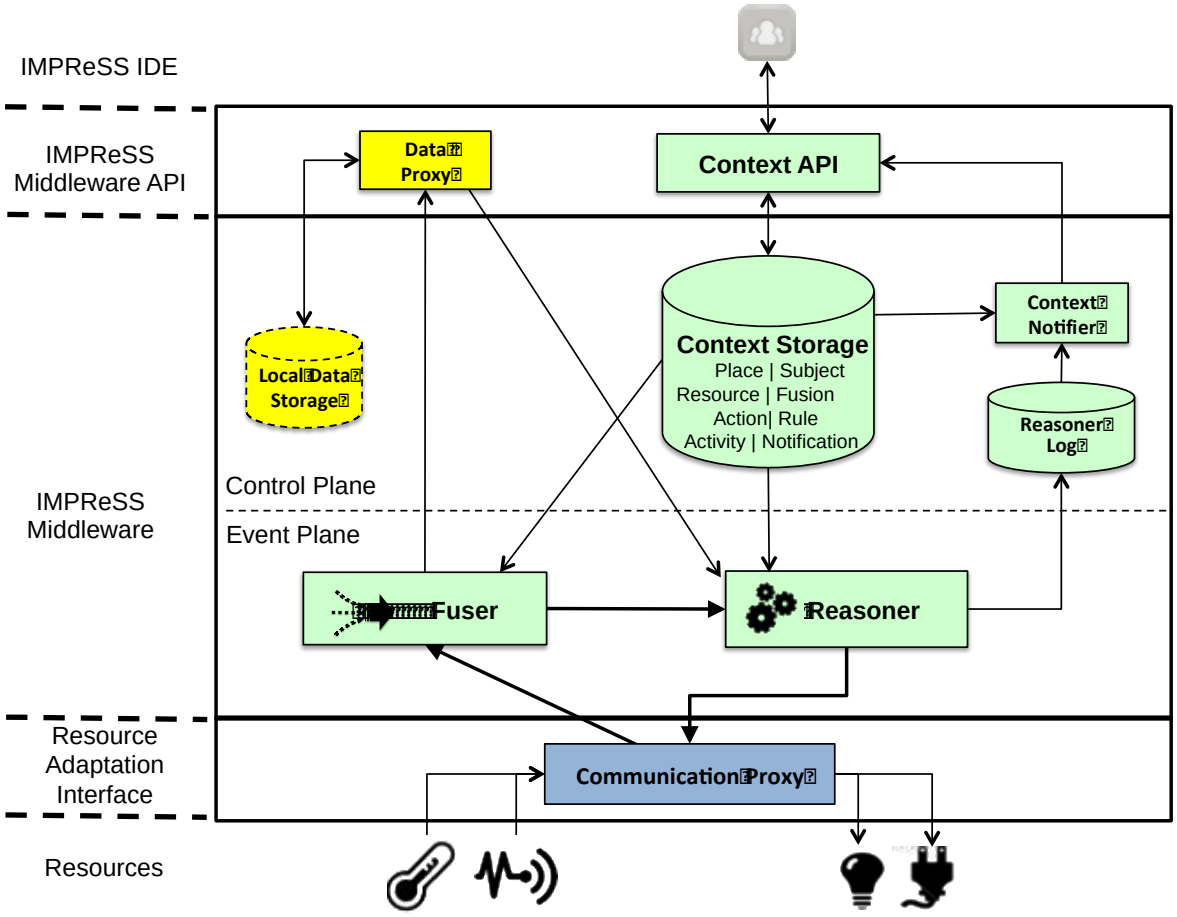

Figure 1 – IMPReSS Context Manager Architecture

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.

5. Context Manager Implementation

This section describes the implementation choices, details and modelling used in the development of the Context Manager.

5.1 Technologies

The implementation of the Context Manager Architecture described in section 4, based on an object-oriented context modeling and rule-based context reasoning, is aimed at developing a preliminary prototype for evaluating the adequateness of using different existing technologies. The implementation guidelines shown in Figure 2 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.

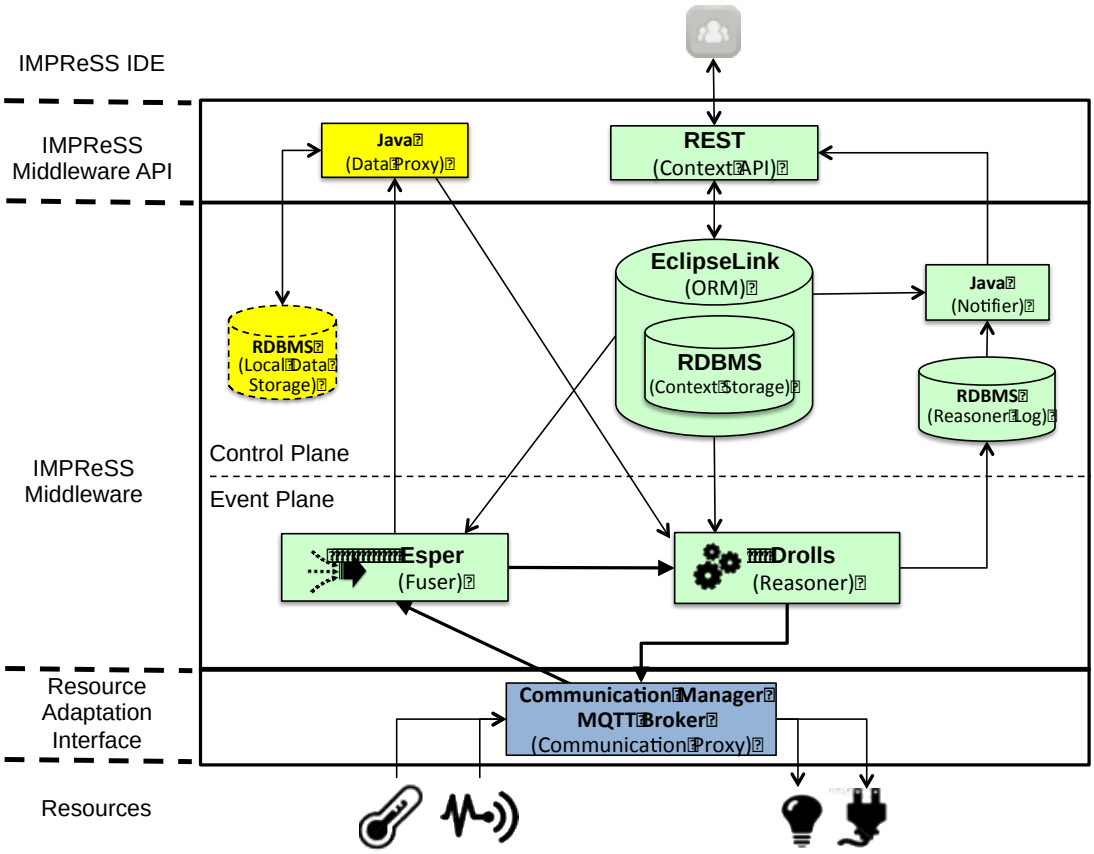

Figure 2 –Implementation guidelines for IMPReSS Context Manager Architecture

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 (for accessing the Data Manager) or the Local Data Storage.
- Local Data Storage: the current version uses PostgreSQL[12], a well-known Relational Database Management System (RDBMS).

[12] http://www.postgresql.org

- Context Storage: any RDBMS with an Object-Relational Mapping (ORM) system, such as EclipseLink[13] or Hibernate[14]. The use of an ORM is needed because on the one hand data is structured 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. We used the ORM EclipseLink together with PostgreSQL. EclipseLink is widely used and supported by a community of developers and since the Eclipse IDE has been chosen, it makes the configuration of EclipseLink simple and straightforward.
- Reasoner: Drools[15] is a Rule Engine, which is classified as a Business Rules Management System (BRMS)[16]. 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.
- Reasoner Log: any RDBMS can be used, and this version is based on PostgreSQL
- 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[17] 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[18] broker.

5.2 Context Storage – Relational Database Model

The Context Manager is based on an object-oriented modelling, since it easily integrates with current programming languages, such as Java. However, when it comes to storing the objects in main memory, the most efficient way currently in use is by converting them into a relational database. This solution requires a relational database, PostgreSQL in the current version, a mapping between objects represented in the programming language and tables in the database and an efficient modeling of the database for representing the context entities templates.

We developed a relational modelling for storing entity templates, which operates in two categories: 1 - How to Operate; 2 – Operation Log. How-to and log are represented in blue and orange, respectively, as shown in Figure 3.

Figure 3 – Layers where the relational data modelling operates

Tables belonging to the "how to" category are aimed at storing the entity templates, such as places, sensors, actuators and other preferences. On the other hand, tables belonging to the "operation log"

[13] http://eclipse.org/eclipselink
[14] http://hibernate.org
[15] http://www.drools.org
[16] http://www.drools.org
[17] http://esper.codehaus.org
[18] http://mqtt.org

category store log information of all actions executed by the Context Manager, such as actions executed by actuators, fusion outcomes, etc.

Figure 4 depicts the diagram that represents a relational model developed for the Context Manager. In a relational model, each entity (table) is represented as a rectangle. Relationships between entities are represented as continuous lines with symbols in their ends representing the type of relationship, which in our case may be one-to-one/many or one-to-zero/one/many. Please notice that entity PLACE, which stores all places of importance to the system, has self-relationship, because a place may be located inside another place. For example, a "classroom" place is within a "corridor" place, which in turn is within a "floor" place, which finally is within a "building" place. A self-relationship is implemented by using a foreign key filled in with a value coming from a primary key from the same table. Some tables may be play a temporary and internal role and may be substituted when all modules of the IMPReSS architecture are integrated into the platform, because some data types (entities) will be stored inside other modules of the IMPReSS platform. This relational model has been implemented in PostgreSQL and the creation script can be found in Annex A.

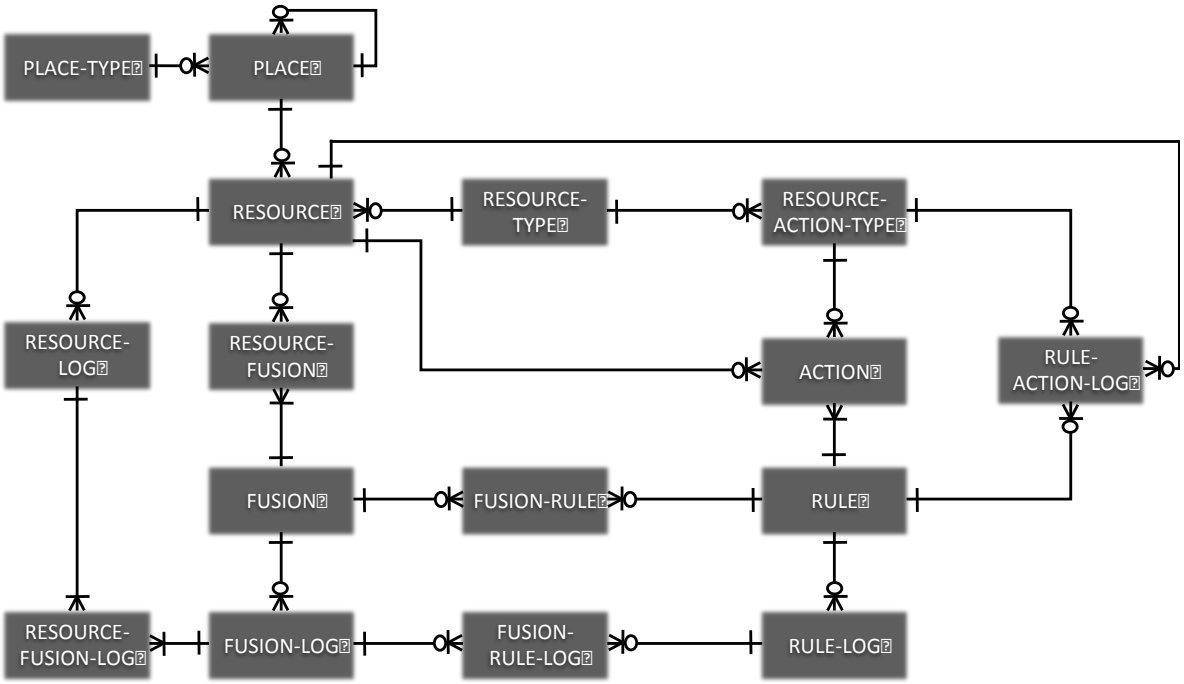

Figure 4 – Context Manager Relational Model

The tables belonging to the Context Manager Contextual Model that store entities are presented below.

- Place: stores Place entities
- Place-Type: stores types of places
- Resource: stores Resource entities
- Resource-Type: stores types of resources
- Resource-Log: stores all values gathered from sensors (a temporary table that will be changed by a query to the Data Manager, responsible for storing sensor data)
- Resource-Fusion: relates resources to fusion criteria
- Fusion: stores Fusion entities
- Fusion-Log: logs fusion criteria applied and its outcomes
- Resource-Fusion-Log: log of activities for resource-fusion
- Action: stores Action entities

- Resource-Action-Type: stores types of actions
- Rule: stores Rule entities
- Rule-Log: logs rules that have been selected and executed
- Fusion-Rule-Log: relates fusion and rule logs, because when a fusion outcome is available it fires the rule engine, which may (and is very likely to) execute a rule
- Rule-Action-Log: logs actions performed as a result of the application of a rule

In order to provide a clearer understanding about the way this data model works and the relationship among entities, Figure 5 depicts examples of sample data for some tables. Examples of Log Tables are depicted in Appendix C. Some observations:

- In Figure 5(b), place R808 is a classroom located in the 8th Floor
- In Figure 5(d), resource T1310 is a thermometer located in classroom R808
- In Figure 5(f), the same rule can be invoked by more than one fusion criteria

TABLE PLACE-TYPE	
PLACE_TYPE_ID	**DESCRIPTION**
001	Build
002	Floor
003	Yard
004	Classroom
005	Professor room
006	Auditory

a)

TABLE PLACE			
PLACE_ID	**DESCRIPTION**	**PLACE_TYPE_FK_ID**	**PLACE_FK_ID**
001	B	001	NULL
002	8º	002	001
003	3º	002	001
004	R808	004	002
005	A	001	NULL
006	R300	004	003

b)

TABLE RESOURCE-TYPE		
RESOURCE_TYPE_ID	**DESCRIPTION**	**SENSOR-0-ACTUATOR-1**
001	Air conditioning	1
002	Turnstile	0
003	Thermometer	0
004	Curtains	1
005	Water pump	1
006	Elevator	1

c)

TABLE RESOURCE			
RESOURCE_ID	DESCRIPTION	RESOURCE_TYPE_FK_ID	PLACE_FK_ID
001	AC1320	001	004
002	AC1321	001	004
003	T1310	003	004
004	AC8001	001	006
005	C0077	004	006
006	T0088	003	006

d)

TABLE FUSION	
FUSION_ID	TEXT
001	Calculate the average of the last 10 measurements
002	Calculate the standard deviation of the last 10 measurements
003	Discover maximum point
004	Calculate the standard deviation of the last 5 measurements
005	Discover curve slope
006	Calculate average of the last 5 measurements

e)

TABLE RESOURCE-FUSION		
RESOURCE_FUSION_ID	FUSION_FK_ID	RESOURCE_FK_ID
001	001	001
002	001	003
003	001	005
004	002	002
005	002	004
006	002	006

f)

TABLE RESOURCE-ACTION-TYPE		
RESOURCE_ACTION_TYPE_ID	RESOURCE_TYPE_FK_ID	TEXT
001	001	Turn on
002	001	Turn off
003	001	Increase
004	001	Decrease
005	004	Open
006	004	Close

g)

TABELA RULES	
ID_RULES	TEXT
001	Rule: Value > 22,0 Action
002	Rule: Value > 0,8 Action

h)

TABLE FUSION-RULE		
FUSION_RULE_ID	FUSION_FK_ID	RULES_FK_ID
001	001	001
002	006	001
003	002	002
004	004	002

i)

TABLE ACTION			
ACTION_ID	RULES_FK_ID	RESOURCE_FK_ID	RESOURCE_ACTION_TYPE_FK_ID
001	001	001	001
002	001	001	003
003	002	002	001
004	002	002	003

j)

Figure 5 – Sample data for selected tables

5.3 Drools: Guvnor and Expert

Drools is a product provided by Red Hat composed of different modules, among which for the IMPReSS Context Manager the Guvnor and Expert modules were used. Guvnor is executed by a JBoss Application Server. Drools Guvnor is in charge of storing rules in a repository, which for the Context Manager can be updated at any time without recompiling any source code or stopping the application. This feature is in line with the concept of context-awareness, which aims at allowing application to change behavior according to changes in the environment without human intervention. Drools Expert is responsible for the reasoning of the rules. It runs within a Java application and can be updated by Guvnor on the fly.

5.4 Esper: Complex Event Processing (Fusion)

Sensors usually send measured data to a server or broker, depending of the system architecture under consideration. The amount of data that arrives at the broker/server at a given type can be enormous, depending on the number of sensors and the frequency by which they are configure to report the measured data. In such a scenario, it is common that repeated or noisy data is received. The idea behind data fusion is to analyze sensor data and filter or pre-process them in order to provide refined and summarized data to the rule reasoning engine. Thus, any application created by the IMPReSS platform will fire a lower number of actions and consequently send a lower number of messages to the broker, which in turn send them to the actuators. Also, fusion criteria can mix different types of data coming from different sensors, thus yielding some high-level context information out of low-level context data.

The IMPReSS Context Manager uses Esper for Fusion purposes. In Esper, Fusion criteria, called streaming, look like SQL statements. Esper allows a variety of fusion criteria to be executed at the same time. Also, Esper provides object-oriented features, which allows fusion criteria to be integrated to the classes and objects written in Java for our Context Manager. As presented in Figure 6, the Context Manager architecture has a pre-processing module that subscribes topics in the MQTT broker and receives data coming from sensors. These data are treated and converted into events before they are forwarded to Esper. When Esper receives these events, it analyzes them according to fusion criteria, which are read from the entity repository, stored in a relational database.

5.5 Context API

In order to provide a uniform and standard interface for external access as well as to provide data integrity to the Context Manager, all Create-Read-Update-Delete (CRUD) operations are performed through a REST API. Basically, a REST API uses four methods: GET, DELETE, POST and PUT. Appendix B contains a detailed list of all methods belonging to the Context Manager API.

Table 1 – Mapping between CRUD operation and REST Method

Operation	**Method**
Create	POST
Read	GET
Update	PUT
Delete	DELETE

The Swagger framework is used to generate automatic documentation of the REST API developed for the Context Manager of the IMPReSS platform. This framework is based on a Graphical User Interface and provides searching operations and REST URLs.

5.6 Context Manager Processing Steps

In order to provide a better understanding of the interworking of the Context Manager components (Figure 6), 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 6 introduces the processing steps and data flow for the Context Manager, based on the implementation guidelines with software modules presented in Figure 2. 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.

In section 6.4 we exemplify the processing steps and data flow for the Context Manager with two examples, of controlling temperature and lighting.

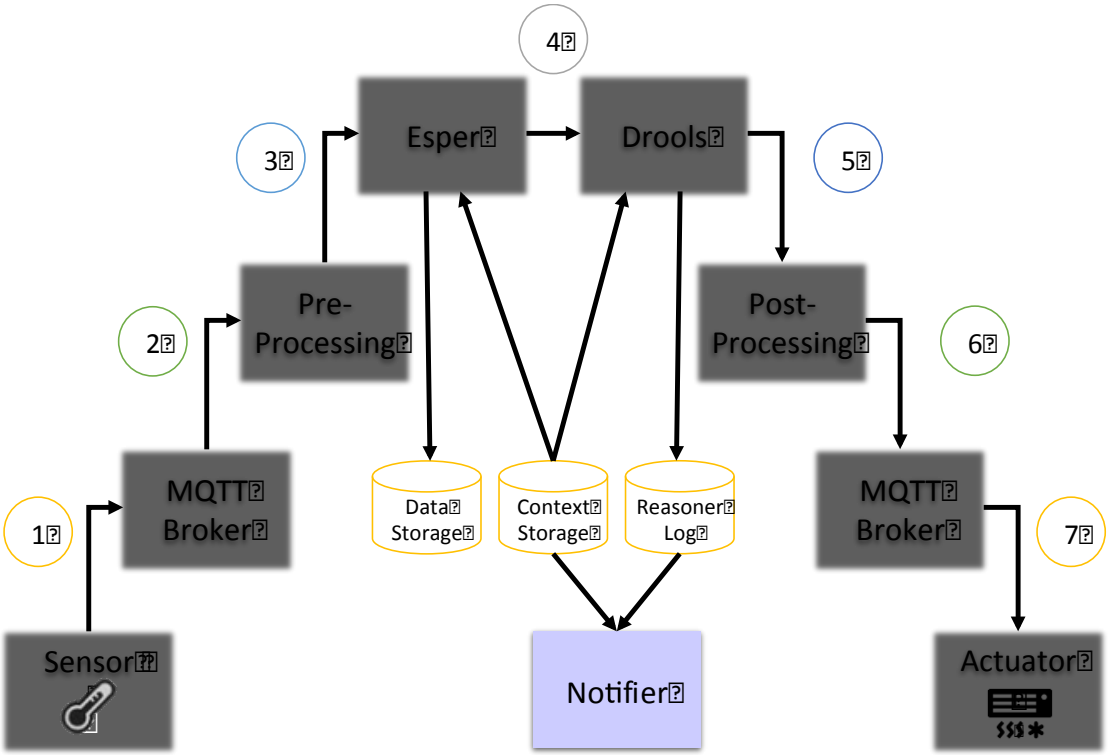

Figure 6 –Processing Steps and Data Flow for the Context Manager

6. Case Study: University Classroom Prototype

A case study has been designed, implemented and deployed covering a typical scenario involving a university classroom, where lighting and temperature can be controlled automatically, in order to evaluate the architecture of the Context Manager and its implementation. The main idea is to demonstrate that we are able to manage the energy used in a place using the IMPReSS platform. Even though this scenario is simple and straightforward with a small number of sensors and actuators, the Context Manager is able to manage larger and more complex scenarios based on different types of sensors, actuators and devices, which makes it different from current solutions, which usually are carefully designed for controlling specific scenarios.

6.1 Scenario

In this scenario, the Context Manager automatically controls lighting and temperature and displays power consumption. It may be considered a mix of scenarios 2 (efficient use of the air conditioning system) and 3 (efficient use of lighting), introduced in Deliverable D6.1 (Kamienski et. a. 2014a). For both situations, there are rules for adapting behavior and fusion criteria for preprocessing sensor data. This scenario contains eight sensors and six actuators. It is depicted in Figure 7 and the communication of these resources with the Context Manager follows the sequence presented in Figure 6. The main components of the scenario shown in Figure 7 are:

- Presence sensors (4): the classroom has four rows of desks and presence sensors identify the presence of students in each row.
- Temperature sensors (2): the classroom has two thermometers, whose measures are averaged by the Fusion component (Esper) for eventually changing the behavior of the system.
- Lighting control sensor (1): this sensor is represented by an app running in a smartphone, where the teacher can turn the lighting system into three modes: ON, OFF and AUTOMATIC. The ON and OFF modes have priority over the AUTOMATIC mode, switching all lights on or off. When in AUTOMATIC mode lights are turned on or off according to the presence sensors.
- Consumption sensor (1): a consumption sensor sums up the power consumption of all devices used in this scenario, i.e., lights and a fan.
- Lighting actuators (4): these actuators turn on and off individual lights for each row.
- Fan actuator (1): turns on and off the fan.
- Lighting control actuator (1): the same smartphone app emulating sensors also emulates an actuator, used to switch the modes of the device.
- Fan (1): a cooler playing the role of an air conditioning system.
- LEDs (4): represent the lights for each row.
- Display (1): a tablet running an app displays information related to all sensors and the state of the system, located in a place where should be an electronic whiteboard. The app is a UI that collects information from the Context Manager through the Context API. The displays present the values measured from of all sensors, percentage of power consumption (related to the maximum) and average temperature. The display is important and should be present in different places in smart buildings, for making available for everyone information about how much energy has been saved by the use of an application generated by the IMPReSS platform.
- Arduino and related circuits (not represented in in Figure 7): an Arduino board together with other related circuits play the role of a customized controller for sensors and actuators. Arduino connects to a MQTT Broker in order to send data measured by the sensors and receive commands for the actuators.

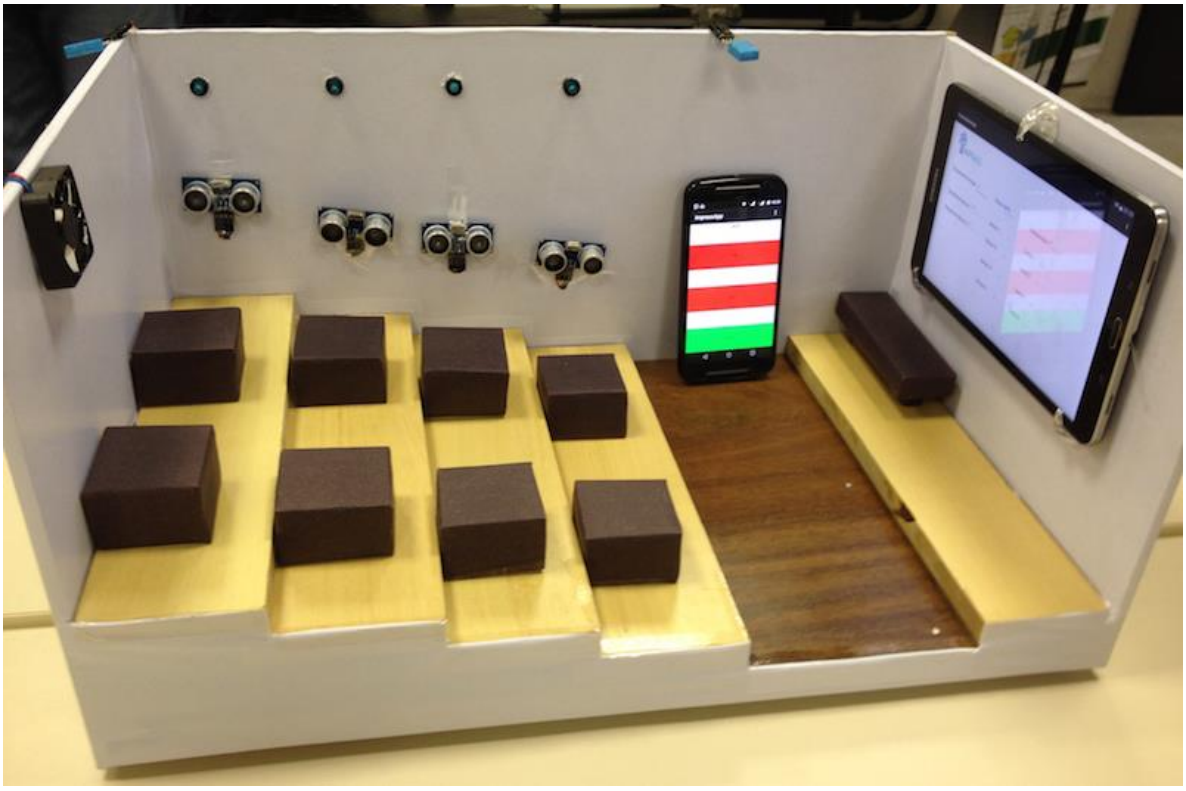

Figure 7 – University Classroom Prototype

6.2 Sensors and Actuators

Arduino works as a hub, connecting all physical sensors and actuators to a Mosquitto[19] MQTT Broker, which in turn connects to the Context Manager. Arduino is connected to sensors and with a frequency that can be configured (say each 1 second) it sends data measured from each sensor to the broker. Also, it receives commands from the Context Manager and performs the due actions in the actuators. The Android (smartphone) app is not connected to the Arduino, because it is able to run a MQTT agent itself. A specific protocol for sending and receiving messages through MQTT has been developed. The data values measured by the sensors are sent to the Context Manager through the MQTT broker and stored in the relational database according to section 5.2.

Table 2 shows all sensors and actuators used in this case study. It is important to highlight that virtual sensors are also created. For example, the average temperature is considered to be a virtual sensor.

Table 2 – Resources used in the prototype (sensors and actuators)

ID	Description	Description
1	PS1	Presence Sensor 1
2	PS2	Presence Sensor 2
3	PS3	Presence Sensor 3
4	PS4	Presence Sensor 4
5	TS1	Temperature Sensor 1
6	TS2	Temperature Sensor 2
7	US1	Usage Sensor 1

[19] http://mosquitto.org

8	LI1	Light Actuator 1
9	LI2	Light Actuator 2
10	LI3	Light Actuator 3
11	LI4	Light Actuator 4
12	AI4	Fan
13	AN1	Android Sensor 1
14	AN2	Android Actuator1
15	AVT T	Average Temperature (virtual)
16	PRESENCE SUM	Bit Sum Presence sensor (virtual)

For this case study three MQTT topics were created, namely impress/action, impress/demo and impress/android, due to the low processing power of the Arduino. Topic impress/demo is used to send sensor data to the Context Manager. Topic impress/action is used by the Context Manager to send actions to the actuators and topic impress/android is used to send notifications to the lighting control app executed in the smartphone.

6.3 Fusion and Rules

A typical scenario involving a university campus with various buildings will be based on hundreds or thousands of sensors, which generate a huge amount of data that must be processed by the Context Manager. A great deal of these data is repeated, noisy or they are not needed in its raw form (like individual temperature data). This leads to the need to use specialized software for preprocessing data before they are sent to the rule engine. This operation is known as data fusion, which in our implementation is played by the Esper Complex Even Processing (CEP) software.

This scenario uses three fusion criteria:

- Fusion 1: compute the average of temperature data coming from the temperature sensors every 10 seconds
- Fusion 2: capture the sum of presence sensor bits every 3 seconds, according to Table 3. The fusion is a sum of the values (0 or 1) coming from the presence sensors, where sensor 1 (row 1) is the least significant and sensor 4 (row 4) is the most significant bit. According to the sum of the bits coming from the sensors, we are able to pinpoint the rows where there are students and those that are empty.
- Fusion 3: Filter Lighting Control Sensor when it is in AUTOMATIC mode, i.e., the output will be the measured value only if it is ON or OFF

Table 3 – States of the sensors presence and corresponding actions

Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sum	Actions
0	0	0	0	0	Turn off all lights
0	0	0	1	1	Turn on light 1; Turn off lights 2, 3 e 4
0	0	1	0	2	Turn on lights 1 e 2; Turn off lights 3 e 4
0	0	1	1	3	Turn on lights 1 e 2; Turn off lights 3 e 4
0	1	0	0	4	Turn on lights 1, 2 e 3; Turn off light 4
0	1	0	1	5	Turn on lights 1, 2 e 3; Turn off light 4
0	1	1	0	6	Turn on lights 1, 2 e 3; Turn off light 4

Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sum	Actions
0	1	1	1	7	Turn on lights 1, 2 e 3; Turn off light 4
1	0	0	0	8	Turn on lights 1, 2, 3 e 4;
1	0	0	1	9	Turn on lights 1, 2, 3 e 4;
1	0	1	0	10	Turn on lights 1, 2, 3 e 4;
1	0	1	1	11	Turn on lights 1, 2, 3 e 4;
1	1	0	0	12	Turn on lights 1, 2, 3 e 4;
1	1	0	1	13	Turn on lights 1, 2, 3 e 4;
1	1	1	0	14	Turn on lights 1, 2, 3 e 4;
1	1	1	1	15	Turn on lights 1, 2, 3 e 4;

However, fusion is not enough for changing behavior of the application on the fly upon changes on the context. The key player for implementing this vision is the rule engine that receives data coming from the fusion engine (Esper), analyzes a set of rules and decides whether actions should be taken. Rules are stored in Drools Guvnor and processed by Drools Expert, which is the engine running over JBoss AS 7. Whenever Esper produces a new result from applying a fusion criterion, the rule engine is invoked and the whole set of rules is analyzed. For our scenario, 10 rules were created, which are presented in Table 4.

Table 4 – Rules used in the Case Study

Rule	Description	Actions
1	Average temperature is higher than a threshold defined by the administrator	Turn on fan
2	Average temperature is lower than a threshold defined by the administrator	Turn off fan
3	Lighting control sensor is set to OFF mode	Turn off all lights
4	Lighting control sensor is set to ON mode	Turn on all lights
5	Sum of presence sensors = 1	Turn on light 1; Turn off lights 2,3 e 4
6	Sum of presence sensors = 2 or 3	Turn on lights 1 e 2; Turn off lights 3 e 4
7	Sum of presence sensors >= 4 and <= 7	Turn on lights 1, 2 e 3; Turn off light 4
8	Sum of presence sensors >= 8	Turn on lights 1, 2, 3 e 4;
9	Sum of presence sensors = 0 and lighting control sensor is in AUTOMATIC mode	Turn off all lights
10	Sum of presence sensors = 0 and lighting control sensor is in ON or OFF modes	Switch to AUTOMATIC mode

6.4 Processing Steps: Lighting and Temperature

This section presents instantiations of the processing steps introduced in section 5.6 for lighting and temperature control. 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.

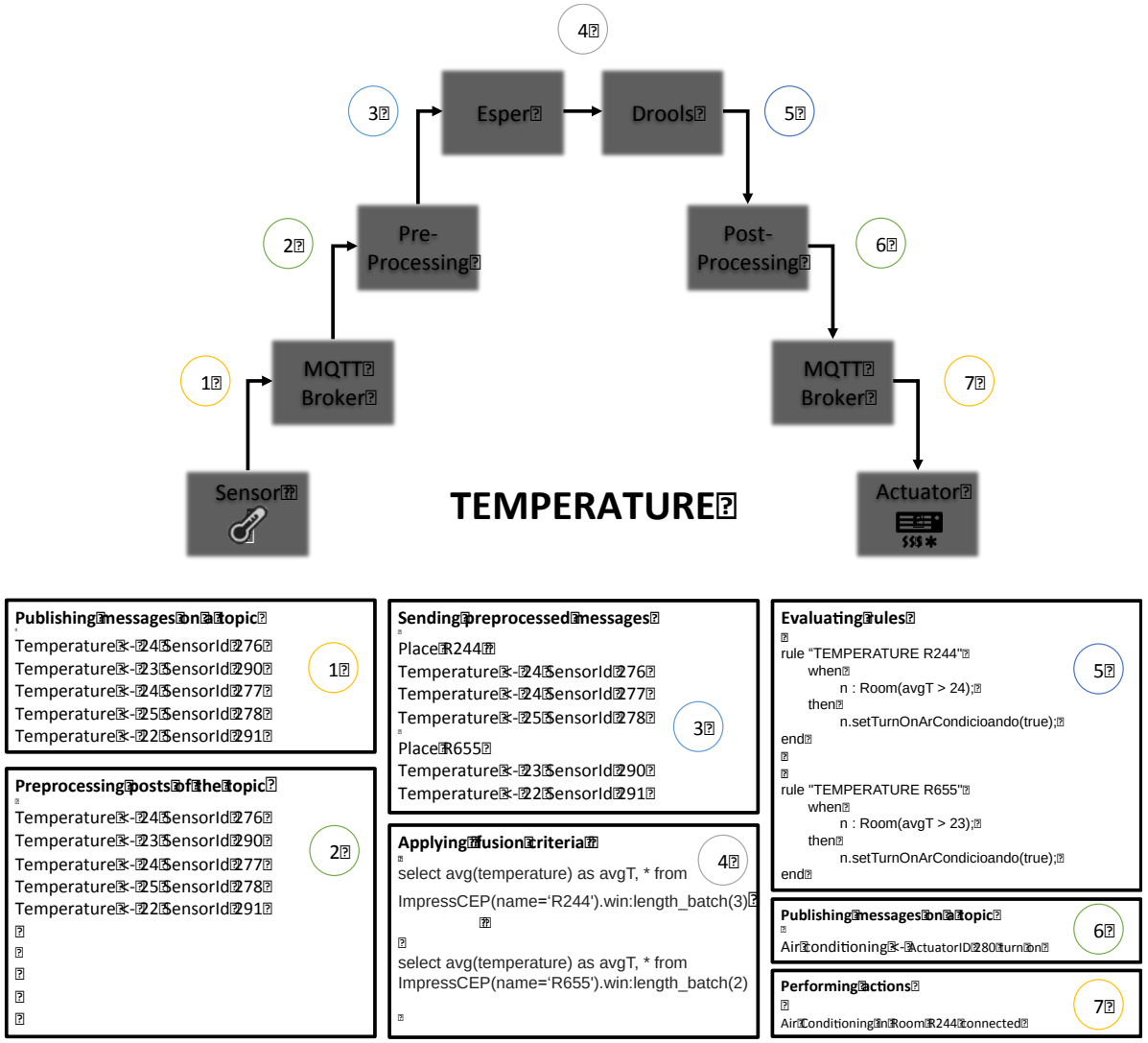

Figure 9 – Processing steps and data flow – Temperature Control

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.

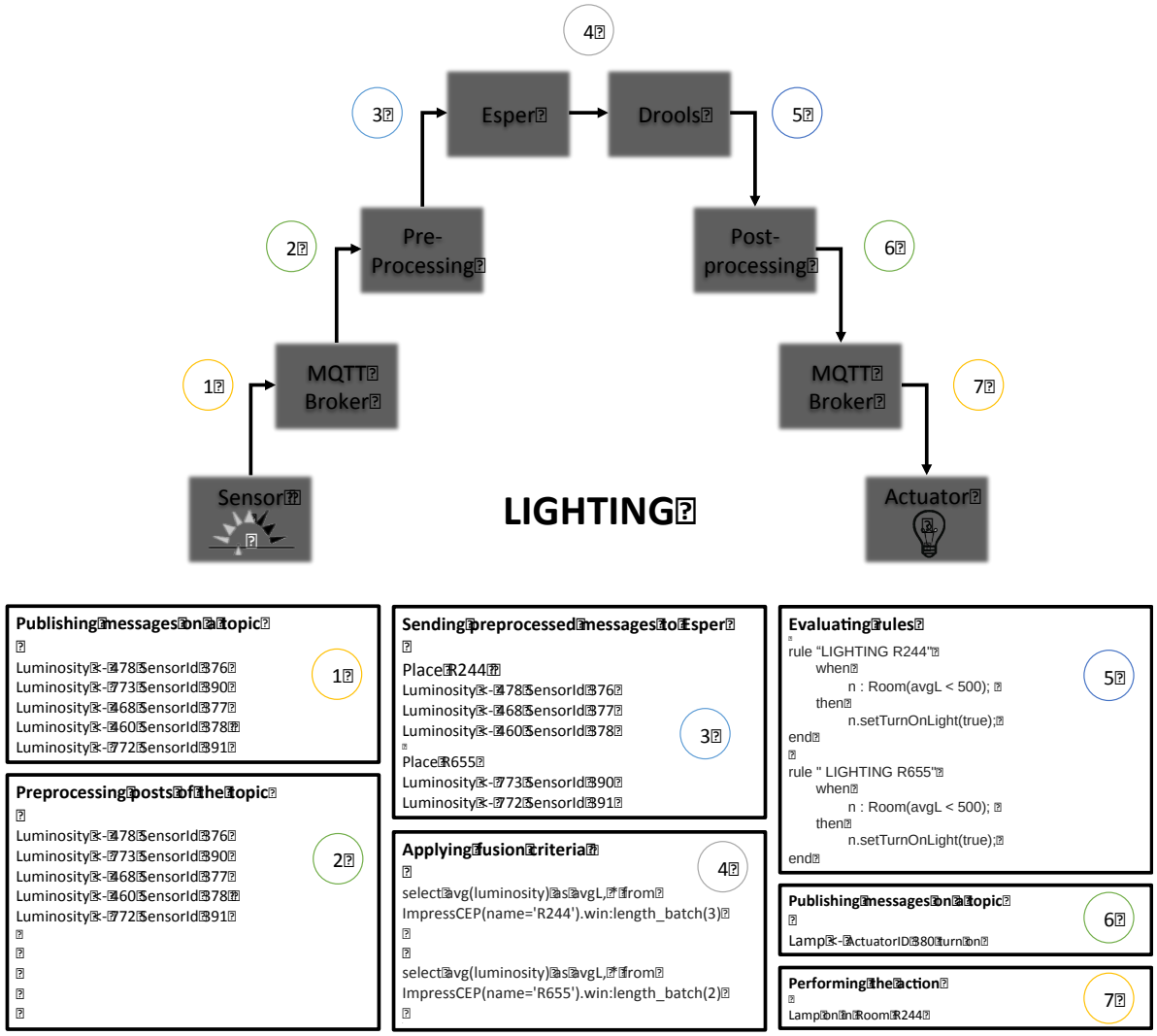

Figure 10 – Processing steps and data flow – Lighting Control

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 object-oriented and rule-based respectively. The architecture of the Context Manager was presented in Deliverable D6.3, which is 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.

This deliverable presents the implementation of the Context Manager, which uses Drools as the rule-based engine and Esper as a Complex Event Processing solution that provides data fusion features. The context modelling approach is object-oriented but since an object-oriented database did not come true, the Context Manager stores entity templates into a relational database and the mapping between objects in the programming language and tables in the database is made by EclipseLink, an Object Relational Mapping (ORM) system. Also, the Context API has been developed using RESTful Web Services.

A case study has been designed, implemented and deployed covering a typical scenario involving a university classroom, where lighting and temperature can be controlled automatically, in order to evaluate the architecture of the Context Manager and its implementation. This case study was successfully developed and resulted in a prototype that can be used for evaluation and demonstration purposes. Different challenges have been identified with the implementation of the Context Manager and the case study, which are not in the mainstream of activities planned for IMPReSS. Therefore, solutions for those challenges will be pursued by MSc and PhD students, since applying context-awareness features to save energy in buildings has become a hot topic in the modern world, and its importance will increasingly grow with the implementation of smart cities concepts.

This deliverable is an important output of Task 6.3 (Context Modelling Templates), following another previous output of the same task, which was the specification of the Context Management Architecture. The next steps are a general improvement in the Context Manager for making it more robust and scalable, which will be subject to an extensive performance analysis. Also, the Context Manager and the Context API will influence the development of the context IDE module, or management UI, that will be reported in Deliverable D6.5 (Implementation of Context Modelling Tool and Templates).

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.

(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.

(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.

(Kamienski et. al 2014c) Kamienski, C., Borelli, F., Oliveira, G., Moretti, W., Pinheiro, I., Belati, E. (2014), Context Management Framework Architecture and Design of Context Templates, IMPReSS Consortium, Deliverable D6.4, November 2014.

(OASIS 2006) OASIS, "Reference Model for Service Oriented Architecture 1.0", October 2006

(OASIS 2014) MQTT Version 3.1.1, OASIS Standard, October 2014.

(ORACLE 2012) ORACLE. JAX-RS: Java API for RESTful Web Services - Version 2.0 Public Review (Second Edition).

(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.

(Pautasso et. al 2008) Pautasso, C., Zimmermann, O., Leymann, F., Restful web services vs. "big"' web services: making the right architectural decision, WWW 2008.

(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

Appendix A – Script for the Context Relational Model

```
CREATE TABLE type_place(
        id_type_place bigserial NOT NULL PRIMARY KEY,
        description text NOT NULL
);

CREATE TABLE place(
        id_place bigserial NOT NULL PRIMARY KEY,
        description text NOT NULL,
        id_type_place_fk bigint,
        CONSTRAINT id_type_place_fk FOREIGN KEY (id_type_place_fk) REFERENCES type_place(id_type_place),
        id_place_fk bigint,
        CONSTRAINT id_place_fk FOREIGN KEY (id_place_fk) REFERENCES place(id_place)
);

CREATE TABLE type_resource(
        id_type_resource bigserial NOT NULL PRIMARY KEY,
        description text NOT NULL,
        sensor_0_actuator_1 int NOT NULL
);

CREATE TABLE resource(
        id_resource bigserial NOT NULL PRIMARY KEY,
        description text NOT NULL,
        id_type_resource_fk bigint,
        CONSTRAINT id_type_resource_fk FOREIGN KEY (id_type_resource_fk) REFERENCES type_resource(id_type_resource),
        id_place_fk bigint,
        CONSTRAINT id_place_fk FOREIGN KEY (id_place_fk) REFERENCES place(id_place)
);

CREATE TABLE log_resource(
        id_log_resource bigserial NOT NULL PRIMARY KEY,
        id_resource_fk bigint,
        CONSTRAINT id_resource_fk FOREIGN KEY (id_resource_fk) REFERENCES resource(id_resource),
        log_resource_value text NOT NULL, /* value para log_resource_value */
```

```
	creation_date timestamp NOT NULL /* timestamp para creation_date */
);

CREATE TABLE fusion(
	id_fusion bigserial NOT NULL PRIMARY KEY,
	fusion_text text NOT NULL /* text para fusion_text */
);

CREATE TABLE log_fusion(
	id_log_fusion bigserial NOT NULL PRIMARY KEY,
	id_fusion_fk bigint,
	CONSTRAINT id_fusion_fk FOREIGN KEY (id_fusion_fk) REFERENCES fusion(id_fusion),
	log_fusion_value text NOT NULL, /* value para log_fusion_value */
	creation_date timestamp NOT NULL /* timestamp para creation_date */
);

CREATE TABLE resource_fusion(
	id_resource_fusion bigserial NOT NULL PRIMARY KEY,
	id_fusion_fk bigint,
	CONSTRAINT id_fusion_fk FOREIGN KEY (id_fusion_fk) REFERENCES fusion(id_fusion),
	id_resource_fk bigint,
	CONSTRAINT id_resource_fk FOREIGN KEY (id_resource_fk) REFERENCES
resource(id_resource)
);

CREATE TABLE type_rsc_actions(
	id_type_rsc_actions bigserial NOT NULL PRIMARY KEY,
	id_type_resource_fk bigint,
		CONSTRAINT id_type_resource_fk FOREIGN KEY (id_type_resource_fk)
REFERENCES type_resource(id_type_resource),
	type_rsc_actions_text text NOT NULL /* text para type_rsc_actions */
);

CREATE TABLE log_rsc_fusion(
	id_log_rsc_fusion bigserial NOT NULL PRIMARY KEY,
	id_log_fusion_fk bigint,
	CONSTRAINT id_log_fusion_fk FOREIGN KEY (id_log_fusion_fk) REFERENCES
log_fusion(id_log_fusion),
	id_log_resource_fk bigint,
```

```
        CONSTRAINT id_log_resource_fk FOREIGN KEY (id_log_resource_fk) REFERENCES
log_resource(id_log_resource)
);

CREATE TABLE rules(
        id_rules bigserial NOT NULL PRIMARY KEY,
        rules_text text NOT NULL /* text para rules_text */
);

CREATE TABLE log_rules(
        id_log_rules bigserial NOT NULL PRIMARY KEY,
        id_rules_fk bigint,
        CONSTRAINT id_rules_fk FOREIGN KEY (id_rules_fk) REFERENCES rules(id_rules),
        creation_date timestamp NOT NULL /* timestamp para creation_date */
);

CREATE TABLE fusion_rules(
        id_fusion_rules bigserial NOT NULL PRIMARY KEY,
        id_fusion_fk bigint,
        CONSTRAINT id_fusion_fk FOREIGN KEY (id_fusion_fk) REFERENCES fusion(id_fusion),
        id_rules_fk bigint,
        CONSTRAINT id_rules_fk FOREIGN KEY (id_rules_fk) REFERENCES rules(id_rules)
);

CREATE TABLE log_fusion_rules(
        id_log_fusion_rules bigserial NOT NULL PRIMARY KEY,
        id_fusion_rules_fk bigint,
        CONSTRAINT id_fusion_rules_fk FOREIGN KEY (id_fusion_rules_fk) REFERENCES
fusion_rules(id_fusion_rules),
        creation_date timestamp NOT NULL /* timestamp para creation_date */
);

CREATE TABLE rules_actions(
        id_rules_actions bigserial NOT NULL PRIMARY KEY,
        id_rules_fk bigint,
        CONSTRAINT id_rules_fk FOREIGN KEY (id_rules_fk) REFERENCES rules(id_rules),
        id_resource_fk bigint,
        CONSTRAINT id_resource_fk FOREIGN KEY (id_resource_fk) REFERENCES
resource(id_resource),
```

```
    id_type_rsc_actions_fk bigint,
    CONSTRAINT id_type_rsc_actions_fk FOREIGN KEY (id_type_rsc_actions_fk) REFERENCES type_rsc_actions(id_type_rsc_actions)
);

CREATE TABLE log_rules_actions(
    id_log_rules_actions bigserial NOT NULL PRIMARY KEY,
    id_rules_fk bigint,
  CONSTRAINT id_rules_fk FOREIGN KEY (id_rules_fk) REFERENCES rules(id_rules),
id_type_rsc_actions_fk bigint,
  CONSTRAINT id_type_rsc_actions_fk FOREIGN KEY (id_type_rsc_actions_fk) REFERENCES type_rsc_actions(id_type_rsc_actions),
    id_resource_fk bigint,
  CONSTRAINT id_resource_fk FOREIGN KEY (id_resource_fk) REFERENCES resource(id_resource),
    creation_date timestamp NOT NULL /* timestamp para creation_date */
);
```

Appendix B – Context Manager API

URLID	Table	Method	Description
http://localhost:8080/restapi/type place-api/get/id=all	place-type	GET	Returns place_type_id and description of all registers
http://localhost:8080/restapi/type place-api/get/id=n	place-type	GET	Returns description of register place_type_id = n;
http://localhost:8080/restapi/type place-api/get/description=x	place-type	GET	Returns place_type_id of register description = x;
http://localhost:8080/restapi/type place-api/delete/id=all	place-type	DELETE	Delete all registers;
http://localhost:8080/restapi/type place-api/delete/id=n	place-type	DELETE	Delete register with place_type_id = n;
http://localhost:8080/restapi/type place-api/delete/description=x	place-type	DELETE	Delete all registers with description = x;
http://localhost:8080/restapi/type place-api/post/description=x	place-type	POST	Insert a new register with description = x;
http://localhost:8080/restapi/type place-api/put/id=n/description=x	place-type	PUT	Update register place_type_id = n with description = x
http://localhost:8080/restapi/type resource-api/get/id=all	resource-type	GET	Returns resource_type_id, description and sensor_0_actuator_1 of registers;
http://localhost:8080/restapi/type resource-api/get/id=n	resource-type	GET	Returns description and sensor_0_actuator_1 of register with resource_type_id = n;
http://localhost:8080/restapi/type resource-api/get/description=x	resource-type	GET	Returns resource_type_id and sensor_0_actuator_1 of register with description = x;
http://localhost:8080/restapi/type resource-api/get/sensor=x	resource-type	GET	Returns resource_type_id and description of all registers with sensor_0_actuator_1 = x;
http://localhost:8080/restapi/type resource-api/delete/id=all	resource-type	DELETE	Delete all registers;
http://localhost:8080/restapi/type resource-api/delete/id=n	resource-type	DELETE	Delete register with resource_type_id = n;
http://localhost:8080/restapi/type resource-api/delete/description=x	resource-type	DELETE	Delete all registers with description = x;
http://localhost:8080/restapi/type resource-api/delete/sensor=x	resource-type	DELETE	Delete all registers with sensor_0_actuator_1 = x;
http://localhost:8080/restapi/type resource-api/post/description=x/sensor=y	resource-type	POST	Insert a new register with description = x and sensor_0_actuador_1 = y;
http://localhost:8080/restapi/type resource-api/put/id=n/description=x/sensor=y	resource-type	PUT	Update register resource_type_id = n with description = x and sensor_0_actuador_1 = y;
URLID	Table	Method	Description
http://localhost:8080/restapi/type rscactions-api/get/id=all	resource-action-type	GET	Returns resource_action_type_id, resource_type_id and resource-action-type_text of all registers;
http://localhost:8080/restapi/type rscactions-api/get/id=n	resource-action-type	GET	Returns resource_type_id and text of register with resource_action_type_id = n;

http://localhost:8080/restapi/typersactions-api/get/text=x	resource-action-type	GET	Returns resource_action_type_id and resource_type_id of register with resource_action_type_id = x;
http://localhost:8080/restapi/typersactions-api/get/type=x	resource-action-type	GET	Returns resource_action_type_id and text of register with resource_type_id = x;
http://localhost:8080/restapi/typersactions-api/delete/id=all	resource-action-type	DELETE	Delete all registers;
http://localhost:8080/restapi/typersactions-api/delete/id=n	resource-action-type	DELETE	Delete register with resource_action_type_id = n;
http://localhost:8080/restapi/typersactions-api/delete/text=x	resource-action-type	DELETE	Delete all registers with resource_action_type_id = x;
http://localhost:8080/restapi/typersactions-api/delete/type=x	resource-action-type	DELETE	Delete all registers with resource_type_id = x;
http://localhost:8080/restapi/typersactions-api/post/type=x/action=y	resource-action-type	POST	Insert a new register with resource_type_id = x and text = y;
http://localhost:8080/restapi/typersactions-api/put/id=n/type=x/action=y	resource-action-type	PUT	Update register resource_action_type_id = n with resource_type_id = x and text = y;
URLID	Table	Method	Description
http://localhost:8080/restapi/fusion-api/get/id=all	fusion	GET	Returns fusion_id and text of all registers;
http://localhost:8080/restapi/fusion-api/get/id=n	fusion	GET	Returns fusion_text of register with fusion_id = n;
http://localhost:8080/restapi/fusion-api/get/text=x	fusion	GET	Returns fusion_id of register with text = x;
http://localhost:8080/restapi/fusion-api/delete/id=all	fusion	DELETE	Delete all registers;
http://localhost:8080/restapi/fusion-api/delete/id=n	fusion	DELETE	Delete register with fusion_id = n;
http://localhost:8080/restapi/fusion-api/delete/text=x	fusion	DELETE	Delete all registers with fusion_text = x;
http://localhost:8080/restapi/fusion-api/post/text=x	fusion	POST	Insert a new register with fusion_text = x;
http://localhost:8080/restapi/fusion-api/put/id=n/text=x	fusion	PUT	Update register fusion_id = n with de fusion_text = x;
URLID	Table	Method	Description
http://localhost:8080/restapi/rules-api/get/id=all	rules	GET	Returns rules_id and text of all registers;
http://localhost:8080/restapi/rules-api/get/id=n	rules	GET	Returns rules_text of register with rules_id = n;
http://localhost:8080/restapi/rules-api/get/text=x	rules	GET	Returns rules_id of register with rules_text = x;
http://localhost:8080/restapi/rules-api/delete/id=all	rules	DELETE	Delete all registers;

http://localhost:8080/restapi/rules-api/delete/id=n	rules	DELETE	Delete register with rules_id = n;
http://localhost:8080/restapi/rules-api/delete/text=x	rules	DELETE	Delete all registers with rules_text = x;
http://localhost:8080/restapi/rules-api/post/text=x	rules	POST	Insert a new register with rules_text = x;
http://localhost:8080/restapi/rules-api/put/id=n/text=x	rules	PUT	Update register rules_id = n with de rules_text = x;
URLID	Table	Method	Description
http://localhost:8080/restapi/place-api/get/id=all	place	GET	Returns place_id, description, place_type_id and place_id(fk) of all registers;
http://localhost:8080/restapi/place-api/get/id=n	place	GET	Returns description, place_type_id and place_id(fk) of register with resource_type_id = n;
http://localhost:8080/restapi/place-api/get/description=x	place	GET	Returns place_id, place_type_id and place_id(fk) of register with description = x;
http://localhost:8080/restapi/place-api/get/type=x	place	GET	Returns place_id, description and place_id(fk) of all registers with place_type_id = x;
http://localhost:8080/restapi/place-api/get/idfk=n	place	GET	Returns place_id, place_type_id and description of all registers with place_id (fk) = n;
http://localhost:8080/restapi/place-api/delete/id=all	place	DELETE	Delete all registers;
http://localhost:8080/restapi/place-api/delete/id=n	place	DELETE	Delete register with place_id = n;
http://localhost:8080/restapi/place-api/delete/description=x	place	DELETE	Delete all registers with description = x;
http://localhost:8080/restapi/place-api/delete/type=x	place	DELETE	Delete all registers with place_type_id = x;
http://localhost:8080/restapi/place-api/delete/idfk=n	place	DELETE	Delete all registers with place_id (fk) = n;
http://localhost:8080/restapi/place-api/post/description=x/type=y/idfk=z	place	POST	Insert a new register with description = x, place_type_id = y and place_id (fk) = z;
http://localhost:8080/restapi/place-api/put/id=n/description=x/type=y/idfk=z	place	PUT	Update register place_id = n with description = x, place_type_id and place_id (fk) = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/resource-api/get/id=all	resource	GET	Returns resource_id, description, resource_type_id and place_id(fk) of all registers;
http://localhost:8080/restapi/resource-api/get/id=n	resource	GET	Returns description, place_type_id and place_id(fk) of register with resource_id = n;
http://localhost:8080/restapi/resource-api/get/description=x	resource	GET	Returns resource_id, place_type_id and place_id(fk) of register with description = x;

http://localhost:8080/restapi/resource-api/get/type=x	resource	GET	Returns resource_id, description and place_id(fk) of all registers with resource_type_id = x;
http://localhost:8080/restapi/resource-api/get/place=p	resource	GET	Returns resource_id, resource_type_id and description of all registers with place_id (fk) = n;
http://localhost:8080/restapi/resource-api/delete/id=all	resource	DELETE	Delete all registers;
http://localhost:8080/restapi/resource-api/delete/id=n	resource	DELETE	Delete register with resource_id = n;
http://localhost:8080/restapi/resource-api/delete/description=x	resource	DELETE	Delete all registers with description = x;
http://localhost:8080/restapi/resource-api/delete/type=x	resource	DELETE	Delete all registers with resource_type_id = x;
http://localhost:8080/restapi/resource-api/delete/idfk=n	resource	DELETE	Delete all registers with place_id (fk) = n;
http://localhost:8080/restapi/resource-api/post/description=x/type=y/idfk=z	resource	POST	Insert a new register with description = x, resource_type_id = y and place_id (fk) = z;
http://localhost:8080/restapi/resource-api/put/id=n/description=x/type=y/idfk=z	resource	PUT	Update register place_id = n with description = x, resource_type_id and place_id (fk) = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/resource-fusion-api/get/id=all	resource-fusion	GET	Returns resource_fusion_id, fusion_id and resource_id of all registers;
http://localhost:8080/restapi/resource-fusion-api/get/id=n	resource-fusion	GET	Returns fusion_id and resource_id of register with resource_fusion_id = n;
http://localhost:8080/restapi/resource-fusion-api/get/fusion=x	resource-fusion	GET	Returns resource_fusion_id and resource_id of all registers with fusion_id = x;
http://localhost:8080/restapi/resource-fusion-api/get/resource=x	resource-fusion	GET	Returns resource_fusion_id and fusion_id of all registers with resource_id = x;
http://localhost:8080/restapi/resource-fusion-api/delete/id=all	resource-fusion	DELETE	Delete all registers;
http://localhost:8080/restapi/resource-fusion-api/delete/id=n	resource-fusion	DELETE	Delete register with resource_fusion_id = n;
http://localhost:8080/restapi/resource-fusion-api/delete/fusion=x	resource-fusion	DELETE	Delete all registers with fusion_id = x;
http://localhost:8080/restapi/resource-fusion-api/delete/resource=x	resource-fusion	DELETE	Delete all registers with resource_id = x;
http://localhost:8080/restapi/resource-fusion-api/post/fusion=x/resource=y	resource-fusion	POST	Insert a new register with fusion_id = x and resource_id = y;
http://localhost:8080/restapi/resource-fusion-api/put/id=n/fusion=y/resource=z	resource-fusion	PUT	Update register resource_fusion_id = n with de fusion_id = x and resource_id = z;

URLID	Table	Method	Description
http://localhost:8080/restapi/fusion-rules-api/get/id=all	fusion-rule	GET	Returns fusion_rule_id, fusion_id and rules_id of all registers;
http://localhost:8080/restapi/fusion-rules-api/get/id=n	fusion-rule	GET	Returns fusion_id and rules_id of register with fusion_rule_id = n;
http://localhost:8080/restapi/fusion-rules-api/get/fusion=x	fusion-rule	GET	Returns fusion_rule_id and rules_id of all registers with fusion_id = x;
http://localhost:8080/restapi/fusion-rules-api/get/rules=x	fusion-rule	GET	Returns fusion_rule_id and fusion_id of all registers with rules_id = x;
http://localhost:8080/restapi/fusion-rules-api/delete/id=all	fusion-rule	DELETE	Delete all registers;
http://localhost:8080/restapi/fusion-rules-api/delete/id=n	fusion-rule	DELETE	Delete register with fusion_rule_id = n;
http://localhost:8080/restapi/fusion-rules-api/delete/fusion=x	fusion-rule	DELETE	Delete all registers with fusion_id = x;
http://localhost:8080/restapi/fusion-rules-api/delete/rules=x	fusion-rule	DELETE	Delete all registers with rules_id = x;
http://localhost:8080/restapi/fusion-rules-api/post/fusion=x/rules=y	fusion-rule	POST	Insert a new register with fusion_id = x and rules_id = y;
http://localhost:8080/restapi/fusion-rules-api/put/id=n/fusion=y/rules=z	fusion-rule	PUT	Update register fusion_rule_id = n with de fusion_id = x and rules_id = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/rules-actions-api/get/id=all	action	GET	Returns action_id, rules_id, resource_id and resource_action_type_id of all registers;
http://localhost:8080/restapi/rules-actions-api/get/id=n	action	GET	Returns rules_id, resource_id and resource_action_type_id of register with action_id = n;
http://localhost:8080/restapi/rules-actions-api/get/rules=x	action	GET	Returns action_id, resource_id and resource_action_type_id of register with rules_id = x;
http://localhost:8080/restapi/rules-actions-api/get/resource=x	action	GET	Returns action_id, rules_id and resource_action_type_id of all registers with resource_id = x;
http://localhost:8080/restapi/rules-actions-api/get/type=x	action	GET	Returns action_id, rules_id and resource_id of all registers with resource_action_type_id = x;
http://localhost:8080/restapi/rules-actions-api/delete/id=all	action	DELETE	Delete all registers;
http://localhost:8080/restapi/rules-actions-api/delete/id=n	action	DELETE	Delete register with action_id = n;
http://localhost:8080/restapi/rules-actions-api/delete/description=x	action	DELETE	Delete all registers with rules_id = x;
http://localhost:8080/restapi/rules-actions-api/delete/type=x	action	DELETE	Delete all registers with resource_id = x;
http://localhost:8080/restapi/rules-actions-api/delete/idfk=n	action	DELETE	Delete all registers with resource_action_type_id = n;
http://localhost:8080/restapi/rules-actions-	action	POST	Insert a new register with rules_id = x, resource_id = y and resource_action_type_id =

api/post/rules=x/resource=y/type=z			z;
http://localhost:8080/restapi/resource-api/put/id=n/rules=x/resource=y/type=z	action	PUT	Update register action_id = n with rules_id = x, resource_id and resource_action_type_id = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/log-resource-api/get/id=all	log-resource	GET	Returns resource_log_id, resource_id and value of all registers;
http://localhost:8080/restapi/log-resource-api/get/id=n	log-resource	GET	Returns resource_id and resource_log_value of register with resource_log_id = n;
http://localhost:8080/restapi/log-resource-api/get/resource=x	log-resource	GET	Returns resource_log_id and resource_log_value of the most recently register with resource_id = x;
http://localhost:8080/restapi/log-resource-api/get/value=x	log-resource	GET	Returns resource_log_id and resource_id of all registers with resource_log_value = x;
http://localhost:8080/restapi/log-resource-api/delete/id=all	log-resource	DELETE	Delete all registers;
http://localhost:8080/restapi/log-resource-api/delete/id=n	log-resource	DELETE	Delete register with resource_log_id = n;
http://localhost:8080/restapi/log-resource-api/delete/resource=x	log-resource	DELETE	Delete all registers with resource_id = x;
http://localhost:8080/restapi/log-resource-api/delete/value=x	log-resource	DELETE	Delete all registers with resource_log_value = x;
http://localhost:8080/restapi/log-resource-api/post/resource=x/value=y	log-resource	POST	Insert a new register with resource_id = x and resource_log_value = y;
http://localhost:8080/restapi/log-resource-api/put/id=n/resource=y/value=z	log-resource	PUT	Update register resource_log_id = n with de resource_id = x and resource_log_value = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/log-fusion-api/get/id=all	fusion-log	GET	Returns fusion_log_id, fusion_id and value of all registers;
http://localhost:8080/restapi/log-fusion-api/get/id=n	fusion-log	GET	Returns fusion_id and fusion_log_value of register with fusion_log_id = n;
http://localhost:8080/restapi/log-fusion-api/get/fusion=x	fusion-log	GET	Returns fusion_log_id and fusion_log_value of all registers with fusion_id = x;
http://localhost:8080/restapi/log-fusion-api/get/value=x	fusion-log	GET	Returns fusion_log_id and fusion_id of all registers with fusion_log_value = x;
http://localhost:8080/restapi/log-fusion-api/delete/id=all	fusion-log	DELETE	Delete all registers;
http://localhost:8080/restapi/log-fusion-api/delete/id=n	fusion-log	DELETE	Delete register with fusion_log_id = n;
http://localhost:8080/restapi/log-fusion-api/delete/fusion=x	fusion-log	DELETE	Delete all registers with fusion_id = x;
http://localhost:8080/restapi/log-fusion-api/delete/value=x	fusion-log	DELETE	Delete all registers with fusion_log_value = x;

http://localhost:8080/restapi/log-fusion-api/post/fusion=x/value=y	fusion-log	POST	Insert a new register with fusion_id = x and fusion_log_value = y;
http://localhost:8080/restapi/log-fusion-api/put/id=n/fusion=y/value=z	fusion-log	PUT	Update register fusion_log_id = n with de fusion_id = x and fusion_log_value = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/log-rules-api/get/id=all	rule-log	GET	Returns rule_log_id, rules_id and creation_date of all registers;
http://localhost:8080/restapi/log-rules-api/get/id=n	rule-log	GET	Returns rules_id and creation_date of register with rule_log_id = n;
http://localhost:8080/restapi/log-rules-api/get/rules=x	rule-log	GET	Returns rule_log_id and creation_date of all registers with rules_id = x;
http://localhost:8080/restapi/log-rules-api/get/date=x	rule-log	GET	Returns rule_log_id and rules_id of all registers with creation_date = x;
http://localhost:8080/restapi/log-rules-api/delete/id=all	rule-log	DELETE	Delete all registers;
http://localhost:8080/restapi/log-rules-api/delete/id=n	rule-log	DELETE	Delete register with rule_log_id = n;
http://localhost:8080/restapi/log-rules-api/delete/rules=x	rule-log	DELETE	Delete all registers with rules_id = x;
http://localhost:8080/restapi/log-rules-api/delete/date=x	rule-log	DELETE	Delete all registers with creation_date = x;
http://localhost:8080/restapi/log-rules-api/post/rules=x/date=y	rule-log	POST	Insert a new register with rules_id = x and creation_date = y;
http://localhost:8080/restapi/log-rules-api/put/id=n/rules=y/date=z	rule-log	PUT	Update register rule_log_id = n with de rules_id = x and creation_date = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/log-rsc-fusion-api/get/id=all	resource-fusion-log	GET	Returns resource_fusion_log_id, fusion_log_id and resource_log_id of all registers;
http://localhost:8080/restapi/log-rsc-fusion-api/get/id=n	resource-fusion-log	GET	Returns fusion_log_id and resource_log_id of register with resource_fusion_log_id = n;
http://localhost:8080/restapi/log-rsc-fusion-api/get/fusion=x	resource-fusion-log	GET	Returns resource_fusion_log_id and resource_log_id of all registers with fusion_log_id = x;
http://localhost:8080/restapi/log-rsc-fusion-api/get/resource=x	resource-fusion-log	GET	Returns resource_fusion_log_id and fusion_log_id of all registers with resource_log_id = x;
http://localhost:8080/restapi/log-rsc-fusion-api/delete/id=all	resource-fusion-log	DELETE	Delete all registers;
http://localhost:8080/restapi/log-rsc-fusion-api/delete/id=n	resource-fusion-log	DELETE	Delete register with resource_fusion_log_id = n;
http://localhost:8080/restapi/log-rsc-fusion-api/delete/fusion=x	resource-fusion-log	DELETE	Delete all registers with fusion_log_id = x;
http://localhost:8080/restapi/log-rsc-fusion-api/delete/resource=x	resource-fusion-log	DELETE	Delete all registers with resource_log_id = x;

http://localhost:8080/restapi/log-rsc-fusion-api/post/fusion=x/resource=y	resource-fusion-log	POST	Insert a new register with fusion_log_id = x and resource_log_id = y;
http://localhost:8080/restapi/log-rsc-fusion-api/put/id=n/fusion=y/resource=z	resource-fusion-log	PUT	Update register resource_fusion_log_id = n with fusion_log_id = x and resource_log_id = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/log-fusion-rules-api/get/id=all	fusion-rule-log	GET	Returns fusion_rule_log_id, fusion_rule_id and creation_date of all registers;
http://localhost:8080/restapi/log-fusion-rules-api/get/id=n	fusion-rule-log	GET	Returns fusion_rule_id and creation_date of register with fusion_rule_log_id = n;
http://localhost:8080/restapi/log-fusion-rules-api/get/rules=x	fusion-rule-log	GET	Returns fusion_rule_log_id and creation_date of all registers with fusion_rule_id = x;
http://localhost:8080/restapi/log-fusion-rules-api/get/date=x	fusion-rule-log	GET	Returns fusion_rule_log_id and fusion_rule_id of all registers with creation_date = x;
http://localhost:8080/restapi/log-fusion-rules-api/delete/id=all	fusion-rule-log	DELETE	Delete all registers;
http://localhost:8080/restapi/log-fusion-rules-api/delete/id=n	fusion-rule-log	DELETE	Delete register with fusion_rule_log_id = n;
http://localhost:8080/restapi/log-fusion-rules-api/delete/rules=x	fusion-rule-log	DELETE	Delete all registers with fusion_rule_id = x;
http://localhost:8080/restapi/log-fusion-rules-api/delete/date=x	fusion-rule-log	DELETE	Delete all registers with creation_date = x;
http://localhost:8080/restapi/log-fusion-rules-api/post/rules=x/date=y	fusion-rule-log	POST	Insert a new register with fusion_rule_id = x and creation_date = y;
http://localhost:8080/restapi/log-fusion-rules-api/put/id=n/rules=y/date=z	fusion-rule-log	PUT	Update register fusion_rule_log_id = n with de fusion_rule_id = x and creation_date = z;
URLID	Table	Method	Description
http://localhost:8080/restapi/log-rules-actions-api/get/id=all	rule-action-log	GET	Returns rule_action_log_id, rules_id, resource_action_type_id, resource_id and creation_date of all registers;
http://localhost:8080/restapi/log-rules-actions-api/get/id=n	rule-action-log	GET	Returns rules_id, resource_action_type_id, resource_id and creation_date of register with rule_action_log_id = n;
http://localhost:8080/restapi/log-rules-actions-api/get/rules=x	rule-action-log	GET	Returns rule_action_log_id, resource_action_type_id, resource_id and creation_date of all registers with rules_id = x;
http://localhost:8080/restapi/log-rules-actions-api/get/actions=x	rule-action-log	GET	Returns rule_action_log_id, rules_id, resource_id and creation_date of all registers with resource_action_type_id = x;
http://localhost:8080/restapi/log-rules-actions-api/get/resource=x	rule-action-log	GET	Returns rule_action_log_id, rules_id, resource_action_type_id and creation_date of all registers with resource_id = x;
http://localhost:8080/restapi/log-rules-actions-api/get/date=x	rule-action-log	GET	Returns rule_action_log_id, rules_id, resource_action_type_id and resource_id of all registers with creation_date = x;

http://localhost:8080/restapi/log-rules-actions-api/delete/id=all	rule-action-log	DELETE	Delete all registers;
http://localhost:8080/restapi/log-rules-actions-api/delete/id=n	rule-action-log	DELETE	Delete register with rule_action_log_id = n;
http://localhost:8080/restapi/log-rules-actions-api/delete/rules=x	rule-action-log	DELETE	Delete all registers with rules_id = x;
http://localhost:8080/restapi/log-rules-actions-api/delete/actions=x	rule-action-log	DELETE	Delete all registers with resource_action_type_id = x;
http://localhost:8080/restapi/log-rules-actions-api/delete/resource=x	rule-action-log	DELETE	Delete all registers with resource_id = x;
http://localhost:8080/restapi/log-rules-actions-api/delete/date=x	rule-action-log	DELETE	Delete all registers with creation_date = x;
http://localhost:8080/restapi/log-rules-actions-api/post/rules=x/actions=y/resource=z/date=w	rule-action-log	POST	Insert a new register with rules_id = x, resource_action_type_id = y, resource_id = z and creation_date = w;
http://localhost:8080/restapi/log-rules-actions-api/put/id=n/rules=x/actions=y/resource=z/date=w	rule-action-log	PUT	Update register rule_action_log_id = n with de rules_id = x, resource_action_type_id = y, resource_id = z and creation_date = w;

Appendix C –Log Tables

TABLE RESOURCE-LOG			
RESOURCE_LOG_ID	RESOURCE_FK_ID	VALUE	TIMESTAMP
001	003	23,4	14/10/29 12:35:47
002	006	19,8	14/10/29 12:35:54
003	003	23,7	14/10/29 12:36:47
004	006	19,2	14/10/29 12:36:54
005	003	23,9	14/10/29 12:37:47
006	006	18,9	14/10/29 12:37:54

a)

TABLE FUSION-LOG			
FUSION_LOG_ID	FUSION_FK_ID	VALUE	TIMESTAMP
001	001	23,2	14/10/29 12:45:47
002	002	0,8	14/10/29 12:46:54
003	001	19,7	14/10/29 12:56:47
004	001	19,4	14/10/29 13:06:54
005	002	0,9	14/10/29 13:07:47
006	002	1,1	14/10/29 13:17:54

b)

TABLE RESOURCE-FUSION-LOG		
RESOURCE_FUSION_LOG_ID	FUSION_LOG_FK_ID	RESOURCE_LOG_FK_ID
001	001	001
002	001	003
003	001	005
004	002	002
005	002	004
006	002	006

c)

TABLE RULE-LOG		
RULE_LOG_ID	RULES_FK_ID	TIMESTAMP
001	001	14/10/29 12:35:47
002	001	14/10/29 12:45:54
003	001	14/10/29 12:56:47
004	002	14/10/29 13:26:54
005	002	14/10/29 13:37:47
006	002	14/10/29 13:47:54

d)

TABLE FUSION-RULE-LOG		
FUSION_RULE_LOG_ID	FUSION_RULE_FK_ID	TIMESTAMP
001	001	14/10/29 12:35:47
002	002	14/10/29 12:45:54
003	001	14/10/29 12:56:47
004	002	14/10/29 13:26:54
005	003	14/10/29 13:37:47
006	004	14/10/29 13:47:54

e)

TABLE RULE-ACTION-LOG				
RULE_ACTION_LOG_ID	RULES_FK_ID	RESOURCE_ACTION_TYPE_FK_ID	RESOURCE_FK_ID	TIMESTAMP
001	001	001	001	14/10/29 12:35:47
002	001	001	002	14/10/29 12:45:54
003	001	003	001	14/10/29 12:56:47
004	002	001	004	14/10/29 13:26:54
005	002	003	002	14/10/29 13:37:47
006	002	003	004	14/10/29 13:47:54

f)