Universidade de Aveiro Departamento de Eletrónica,
Telecomunicações e Informática
2016

Helder Moreira

Integração de Dados de Sensores e Gestão de Ambientes Inteligentes

Sensor Data Integration and Management of Smart Environments

Universidade de Aveiro Departamento de Eletrónica,
Telecomunicações e Informática
2016

Helder Moreira

Integração de Dados de Sensores e Gestão de Ambientes Inteligentes

Sensor Data Integration and Management of Smart Environments

"*Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time*"

— Thomas A. Edison

Universidade de Aveiro
2016

Departamento de Eletrónica,
Telecomunicações e Informática

Helder
Moreira

Integração de Dados de Sensores e Gestão de Ambientes Inteligentes

Sensor Data Integration and Management of Smart Environments

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor João Paulo Barraca, Professor auxiliar convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Mestre Nuno Lourenço, gerente da empresa Think Control.

Esta dissertação foi financiada por fundos nacionais da FCT/MCTES através do projeto UID/EEA/50008/2013.

Dedico este trabalho à minha familia e à Cris pelo incansável apoio.

o júri / the jury

presidente / president	Prof. Doutor Joaquim Arnaldo Carvalho Martins Professor Catedrático da Universidade de Aveiro
vogais / examiners committee	Prof. Pedro Miguel Alves Brandão Professor Auxiliar da Universidade de Porto
	Prof. Doutor João Paulo Silva Barraca Professor Auxiliar Convidado da Universidade de Aveiro

agradecimentos / acknowledgements

Gostaria de deixar aqui um sincero agradecimento, a todos aqueles que contribuíram para realização desta dissertação, que marca o fim de mais uma importante etapa da minha vida.

À Cris, pelo apoio incondicional e por sempre acreditar em mim. Gratidão!

Aos meus pais por me proporcionarem a possibilidade de frequentar o ensino superior e pelos sacrifícios que fizeram.

Ao meu orientador, Professor Doutor João Paulo Barraca, e coorientador, Nuno Lourenço, não só pela disponibilidade e pelos conselhos na realização desta dissertação, mas também por todo o conhecimento transmitido.

A todos um muito obrigado!

Palavras Chave

Processamento de Eventos Complexos, Redes de Sensores, Internet das Coisas, Ambientes Inteligentes, Automação

Resumo

Num mundo de constante desenvolvimento tecnológico e acelerado crescimento populacional, observa-se um aumento da utilização de recursos energéticos. Sendo os edifícios responsáveis por uma grande parte deste consumo energético, desencadeiam-se vários esforços de investigações de forma a criarem-se edifícios energeticamente eficientes e espaços inteligentes. Esta dissertação visa, numa primeira fase, apresentar uma revisão das atuais soluções que combinam sistemas de automação de edifícios e a Internet das Coisas. Posteriormente, é apresentada uma solução de automação para edifícios, com base em princípios da Internet das Coisas e explorando as vantagens de sistemas de processamento complexo de eventos, de forma a fornecer uma maior integração dos múltiplos sistemas existentes num edifício. Esta solução é depois validada através de uma implementação, baseada em protocolos leves desenhados para a Internet das Coisas, plataformas de alto desempenho, e métodos complexos para análise de grandes fluxos de dados. Esta implementação é ainda aplicada num cenário real, e será usada como a solução padrão para gestão e automação num edifício existente.

Keywords

Complex Event Processing, Wireless Sensor Networks, Internet of Things, Smart Environments, Automation

Abstract

In a world of constant technological development and accelerated population growth, an increased use of energy resources is being observed. With buildings responsible for a large share of this energy consumption, a lot of research activities are pursued with the goal to create energy efficient buildings and smart spaces. This dissertation aims to, in a first stage, present a review of the current solutions combining Building Automation Systems (BAS) and Internet of Things (IoT). Then, a solution for building automation is presented based on IoT principles and exploiting the advantages of Complex Event Processing (CEP) systems, to provide higher integration of the multiple building subsystems. This solution was validated through an implementation, based on standard lightweight protocols designed for IoT, high performance and real time platforms, and complex methods for analysis of large streams of data. The implementation is also applied to a real world scenario, and will be used as a standard solution for management and automation of an existing building.

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AES	Advanced Encryption Standard
AFS	Apache Software Foundation
AMQP	Advanced Message Queuing Protocol
ANSI	American National Standards Institute
API	Application Programming Interface
APP	Application
ARCNET	Attached Resource Computer Network
ASHRAE	American Society of Heating, Refrigerating, and Air-Conditioning Engineers
ATNoG	Aveiro Telecommunications and Networking Group
BAS	Building Automation System
BACnet	Building Automation and Control Networks
BLE	Bluetooth Low Energy
BREEAM	Building Research Establishment Environmental Assessment Method
BRMS	Business Rules Management System
CCTV	Closed-Circuit Television
CEA	Consumer Electronics Association
CEN	European Committee for Standardization
CEP	Complex Event Processing
CFL	Compact Fluorescent Lamps
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CSV	Comma-Separated Values
DALI	Digital Addressable Lighting Interface
DETI	Departamento de Eletrónica, Telecomunicações e Informática
DSL	Domain Specific Language
DTLS	Datagram Transport Layer Security
ECA	Event Condition Action
EHS	European Home System
EIB	European Installation Bus
EN	European Standard
epl	Event Processing Language
ETS	Engineering Tool Software
EXI	Efficient XML Interchange

FO	Fiber Optic
GATT	Generic Attribute Profile
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation and Air Conditioning
ID	Identifier
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IM	Instant Messaging
IoT	Internet of Things
I/O	Input/Output
IPSP	Internet Protocol Support Profile
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPSec	Internet Protocol Security
IP	Internet Protocol
ISO	International Organization for Standardization
IT2	Instituto de Telecomunicações 2
IT	Instituto de Telecomunicações
JMS	Java Message Service
JSON	JavaScript Object Notation
KNX	Konnex
LAN	Local Area Network
LED	Light Emitting Diode
LLC	Logical Link Control
LMSC	IEEE 802 LAN/MAN Standards Committee
LonWorks	Local Operating Network
LR-WPAN	Low Rate WPAN
M2M	Machine-to-Machine
MAN	Metropolitan Area Network
MAC	Media Access Control
MQTT	Message Queue Telemetry Transport
MQTT-SN	MQTT for Sensors Nodes
MS/TP	Master-Slave/Token-Passing
NFC	Near Field Communication
ODM	Operational Decision Management
OPC	Open Platform Communications
ORM	Object-Relational Mapper
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PAN	Personal Area Network
PANA	Protocol for Carrying Authentication for Network Access
PIR	Passive Infrared
PL	Powerline
PoE	Power-over-Ethernet
PTP	Point-to-Point
QL	Query Language
QoS	Quality of Service
QR	Quick Response
RDMS	Relational Database Management System
REST	Representational State Transfer
RF	Radio Frequency
RFID	Radio-Frequency Identification
RPL	Routing Protocol for Low Power and Lossy Networks
RTC	Real-Time Computing

RTS	Real-Time Streaming
SASL	Simple Authentication and Security Layer
SCADA	Supervisory Control And Data Acquisition
SCoT	Smart Cloud of Things
SIG	Special Interest Group
SMS	Short Message Service
SNVT	Standard Network Variable Type
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TP	Twisted Pair
TTL	Time To Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
UUID	Universally Unique ID
VLC	Visible Light Communication
WLAN	Wireless LAN
WPAN	Wireless PAN
WSN	Wireless Sensor Network
WSAN	Wireless Sensor and Actuator Network
WS	Web Service
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YARN	Yet Another Resource Negotiator
ZCL	ZigBee Cluster Library

CHAPTER 1

Introduction

It is clear today how much technology has evolved over the last decades, and how much it is now part of our lives. Technology is everywhere, from our small gadgets to large industries and buildings, where its usage is imperative. Nonetheless, technology is powered by electricity, a resource we continuously strive for. Renewable energies are expanding and addressing this issue with good results, but they still have a long way to go before reaching acceptable costs.

Moreover, buildings, which have been continuously growing in numbers and size, are responsible for a large part of the global energy consumption [1] [2]. With this in mind, a lot of investigation and research has been triggered with the goal of creating energy efficient solutions.

There are currently several solutions in the Building Automation Systems (BAS) domain, capable of automating multiple systems in order to reduce energy wastage. However, these are generally basic in the sense that they only support simple reactions to predefined events coming from the sensors, lacking complex correlations and dynamic integration of all the information provided by those events. Additionally, most are vendor specific solutions and despite using existing standards, integration and interoperability of new system features is still difficult to achieve.

On the other hand, a huge revolution is happening in the Internet, commonly referred to as Internet of Things (IoT). The IoT is a new technological paradigm where devices are capable of generating and sharing large amounts of data with each other, and also pushing it into cloud servers, where it can be analysed and valuable information extracted. These devices, in line with the technological evolution, are achieving smaller sizes, lower costs, and lower energy needs. They are thus becoming suitable for integration into everyday objects, or "things", providing them with the intelligence that allow us to call them "smart objects".

Equipped with sensors and actuators, smart objects are able to collect information from the environment and interact with the physical world, having then the potential not only to enrich our everyday life, but also to empower new ways of working by increasing our comfort and productivity.

Furthermore, the servers, responsible for receiving and analysing the data generated by smart objects, perform tasks with a continuously increasing degree of complexity. This calls for new methodologies and platforms that are able to cope with processing huge amounts of data, in the order of billions of events per second, analyse it, and infer conclusions in near real-time.

It is within the scope of these visions that the current dissertation fits, by proposing a solution that aims to combine and integrate the low cost and low power principles of the IoT in order not only to improve energy efficiency of buildings, but also to give their occupants an increased level of comfort and productivity. In fact, the use of these principles allows the creation of dynamic solutions where new data sources may be easily added as either new devices, able to interact with the environment, or modules that provide different types of information.

1.1 MOTIVATION

With buildings responsible for more than 40% of energy consumption [3], the development of efficient building automation systems that can reduce operational costs through improvement of energy efficiency is a motivation factor by itself. As an additional incentive is the deployment of IoT concepts, recognized as one of the most important research topics in the industrial and academic worlds for the immediate and near future. The work performed within this dissertation combines both motivations, and does so as preliminary work of an internal research activity of the Instituto de Telecomunicações (IT) of Aveiro, named “SmartLighting”.

The institute’s facilities in Aveiro, are composed by two buildings. The latest, referred as Instituto de Telecomunicações 2 (IT2), was inaugurated in 2014 and, despite including modern facilities, is not completely endowed with energy efficient solutions. However, there is an ongoing activity to requalify the existing lighting infrastructure, replacing luminaires with Compact Fluorescent Lamps (CFL) with Light Emitting Diode (LED) units. The simple exchange spurred an interest to enhance the infrastructure with intelligence and automation systems, leading to the SmartLighting project.

The project is being developed by two research groups within IT, the Aveiro Telecommunications and Networking Group (ATNoG) and the Integrated Circuits group. Furthermore, it has the cooperation of Think Control, an engineering consultancy company, and the Zumbtobel Group, a global player in the professional lighting business. The main goal of the project is to build and deploy a wireless network of sensors and LED luminaires that are able to dynamically adapt, by detecting changes in the surrounding environment and reacting to them in a coordinated fashion, taking into account a multitude of conditions.

1.2 OBJECTIVES

The key objective of this dissertation is to propose a solution capable of receiving and processing every environmental change in the IT2 building, and rapidly take actions, in order to provide an automated workplace. Moreover, the solution must allow a seamless integration of new devices and systems in order to be easily extensible and thus allow the addition of new features in the future.

In line with the SmartLighting project, an implementation of this solution is expected, endowed with the most basic functionalities for automating a real prototype composed by a couple luminaires equipped with multiple sensors able to read several environmental variables. The prototype is developed by two other students from the Integrated Circuits group, as their dissertation work. The implementation provides a centralized web interface for building managers to configure the system, by managing the devices and the building structure and, creating and activating simple rules that form the logic that runs the system.

To enable the automation of the real prototype, a gateway agent is also expected for connecting the devices to the platform using Bluetooth Low Energy, as well as a simple device simulator, for allowing not only to create virtual devices for generating data for performance testing purposes, but also to visualize the correct functioning of the platform by showing the state of every sensor and luminaire.

1.3 CONTRIBUTIONS

This work mainly contributes with a solution for integration of Complex Event Processing (CEP) with IoT in the context of building automation. As part of the SmartLighting project, the solution is tested and validated with real sensors following the complete workflow. Additionally, the solution was designed and elaborated aiming to ensure standardized protocol support for the IoT.

The implementation of the solution in conjunction with the developed prototype formed a final demonstrator for the SmartLighting project, which was first publically shown at Students@DETI. This is an annual event which takes place at Departamento de Eletrónica, Telecomunicações e Informática (DETI) of University of Aveiro where students present prototypes developed in project course units, as well as dissertation results or PhD progress.

Additionally, a complete demonstration was also performed at the Research Day event [4] of University of Aveiro. This is, also an annual event that aims to present the most relevant research achievements undertaken by the different departments and research units of the university.

Moreover, contributing to the academic and scientific development, a paper with the title "SmartLighting - A platform for intelligent building management" was submitted and accepted for publication at INForum 2016 [5]. The paper, describes the implementation, along with the prototype, and thus enabling a more assertive dissemination of the project and obtained results, reaching a broader target population.

1.4 STRUCTURE

This document is split into 6 chapters of which, chapter 1, Introduction, was already presented. The remaining chapters are organised as follows.

- **Chapter 2:** presentation of the state of art. In this chapter some key concepts regarding building automation, Complex Event Processing and the Internet of Things will be presented, focusing on the open-source protocols and standards used by current solutions in these areas;
- **Chapter 3:** brief introduction to the SmartLighting project, presenting its main goals and advantages to IT2 building's stakeholders. An analysis of multiple use cases is used to later describe the approach used for building a smart smart environment solution. It presents an architecture, followed by an explanation of its components;
- **Chapter 4:** description of the implementation of the proposed architecture using WSO2 CEP[1]. The solution's architecture is shown along with the technologies use, thus providing a detailed explanation of the followed approach during implementation;
- **Chapter 5:** presentation and analysis of results obtained from testing the implemented solution. The test methodology is described and the results are analysed under a feasibility perspective, with special focus in latencies which are crucial for real-life scenarios.
- **Chapter 6:** final conclusions about the chosen path and obtained results of this work, also addressing potential improvements for future follow up work.

[1] A Complex Event Processing (CEP) engine provided by WSO2 Inc.

CHAPTER 2

State of the art

In a world of constant technological development and continuous population growth, the strain on the planet's energy resources is notorious. A large share of the world's energy usage is taken by buildings, either residential or commercial, whose number and size keeps growing [1][2]. With such strong economic and social impact, legislation is introduced towards achieving higher levels of efficiency. In turn, this triggers a lot of research activities focusing on the problem of delivering energy efficient solutions.

This chapter aims to provide a short review on building automation concepts followed by the IoT and automation concepts that can be used to develop novel solutions. For each section, State-of-the-Art topics are also given to better expose the direction of these research fields and how they can be combined.

2.1 introduction to bas systems

Initially the need for control and later the need for improved building efficiency, led to the development of Building Automation System (BAS) solutions. Nowadays, the primary goal, and selling point, of a BAS is to achieve significant cost reductions over the lifetime of a building, mainly through energy savings. Furthermore, a BAS is also used to achieve a higher ranking in the sustainable buildings scale, such as LEED [6], BREEAM [7] and others, which increase the value of the building while contributing to help protect the environment.

Cost reduction is achieved by BAS by efficiently automating several systems inherent to a building such as Heating, Ventilation and Air Conditioning (HVAC), lighting, water, Closed-Circuit Television (CCTV) to name a few of those depicted in figure 2.1. Furthermore, given the fact that people spend much of their time inside buildings (in the office or at home),

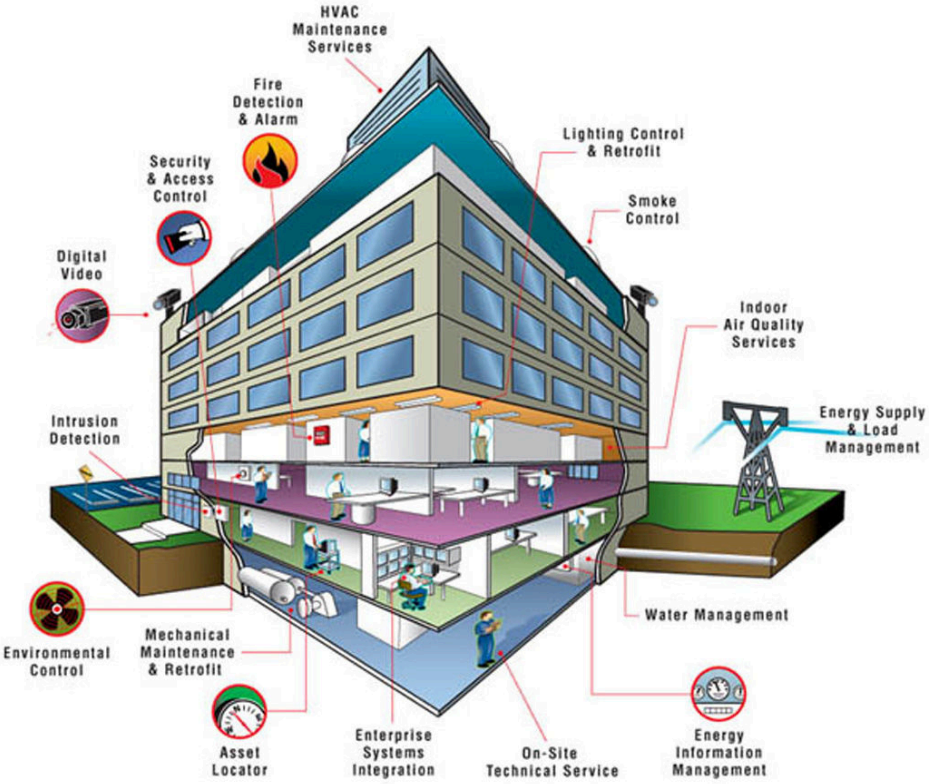

HVAC Maintenance Services
Fire Detection & Alarm
Lighting Control & Retrofit
Security & Access Control
Smoke Control
Digital Video
Indoor Air Quality Services
Intrusion Detection
Energy Supply & Load Management
Water Management
Environmental Control
Mechanical Maintenance & Retrofit
Asset Locator
Enterprise Systems Integration
On-Site Technical Service
Energy Information Management

Figure 2.1: Building services in a Building Automation System [8].

these solutions can help improve their productivity by delivering higher levels of comfort, through monitoring and adjusting environmental parameters. “The key driver of the building automation market is the promise of increased user comfort at reduced operation cost” [9].

Additionally, by automating different systems on a building, other costs besides those related to energy use, can be reduced. The centralized control and monitoring provided by a BAS can provide support for preventive maintenance, and also the means for early detection and location of faulty system sections or components. The ability to react early, and in some cases even correct issues without human intervention, is key in reducing the down-time of building services and systems, thus improving its overall efficiency. The history of BAS is fairly old, with some solutions appearing over one century ago. One of the first was the Automatic Temperature Control System presented by Warren Johnson in 1895 [10]. From an early stage, a common model, as shown in figure 2.2, started to emerge with three distinct layers: field, automation and management.

The field layer considers the devices located in the building and through which all the interactions are made. They are controlled by the automation layer which is responsible for the automation of the different building processes, while the management layer allows control and management of the entire system, as well as data collection and its analysis.

One of the major problems in building automation, as stated by the authors of [9], is that different manufacturers have created different building automation systems with

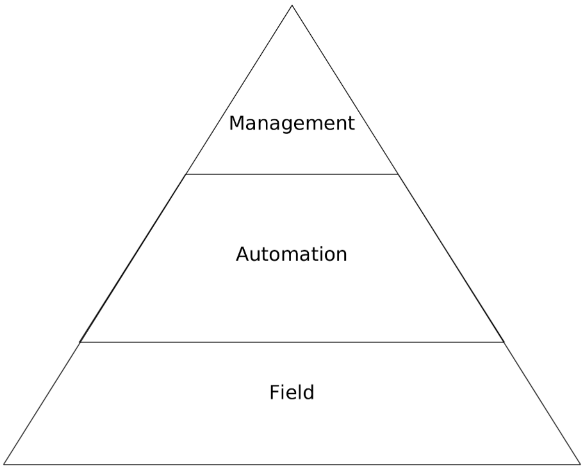


Figure 2.2: Building Automation System layers.

proprietary communication interfaces and protocols, with none of them providing all the types of applications. A need for expert know-how in each individual building service, over time, led to a segmented market. Thus, integration of different systems and components of multiple manufacturers became an often impossible task.

To solve this problem, several vendors began by opening up their system specifications, trying to make their solutions more captivating and securing a dominant market position. With time, standards often evolved as a combination of multiple vendor specifications. Currently the most notable standards in BAS are: BACnet on the management layer; LonWorks and parts of KNX on the automation layer; KNX and DALI (specific for lighting) in the field layer. BAS standards will be discussed in section 2.2.

However, with the ongoing IoT revolution, new solutions have been presented over the past few years, specially designed for constrained devices and low-power wireless links. This is interesting for the building automation area for several reasons. First, the interconnection integration problem stated before is solved by the use of an Internet Protocol (IP) network, which is a well-known open standard. Second, there are various advantages in the use of wireless devices, such as simplified installation, less cost for cabling infrastructure, mobility and high scalability. Third, the use of low-power devices, allows extended life time often using batteries, and sometimes even be self-sufficient. The IoT concept and its protocols shall be discussed in section 2.4.

Also at the automation level, new developments can be exploited. Besides the typical input-output relations, automation can be achieved by analysing and processing data from multiple sources and, based on it, actuate over different systems of the building, including HVAC, lighting equipment and others. This complex correlation of data sources requires mechanisms able to analyse and process heterogeneous sources, and issue commands with very low latency. These concepts will be further addressed in section 2.5.

2.2 Building Automation System (BAS) protocols

Building Automation is accomplished through an integrated control of multiple building services such as HVAC, lighting and others. Thus, in order for sub-systems to be able to communicate and exchange information amongst them, as well as allow the integration of solutions and components from multiple manufacturers, standard communication protocols are needed.

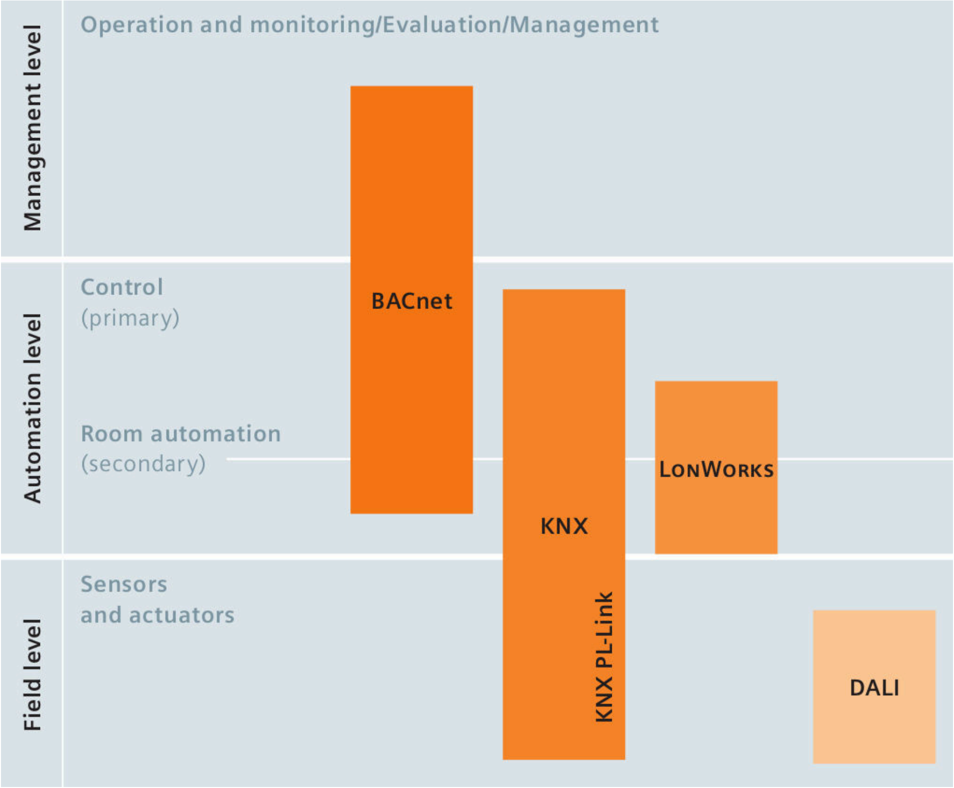


Figure 2.3: BAS protocols diagram [11] (Adapted)

In this section, the Building Automation and Control Networks (BACnet) [12], the Local Operating Network (LonWorks) [13] and the Konnex (KNX) [14] protocols will be presented. These are the main industry-maintained protocols, specifically targeting BAS. In addition to these, the Digital Addressable Lighting Interface (DALI) [15] protocol will also be briefly introduced, since it is the most proliferous digital protocol used in professional lighting control systems. Figure 2.3 shows a diagram with these protocols mapped across the previously presented BAS layers (figure 2.2).

Additionally, the EnOcean Alliance [16] has been providing energy harvesting wireless technologies, mainly targeting Building Automation. Their solutions provide ultra-low-power electronics capable of harvesting the energy they need from external sources, such as solar or kinetic energy. However, since EnOcean provides self-powered devices, which is an asset for the Internet of Things, it has been evolving towards this concept and thus will be included in the IoT section, 2.4.

There are other protocols also adopted in BAS solutions such as ModBus and OPC, which are often used in industrial automation. However, they are not so widespread in commercial and residential building automation, and thus out of scope of this chapter.

2.2.1 BACNET

The BACnet protocol, whose development began in 1987 by the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE), was first published in 1995 when it was adopted as an American National Standards Institute (ANSI)/ASHRAE standard. Later, in 2003, it was also adopted as an International Organization for Standardization (ISO) and a European Committee for Standardization (CEN) standard [9]. It is vendor-independent, without any license fees, and is under continuous development by ASHRAE [11].

BACnet was specifically designed for the management and control of building automation, allowing integration of different systems and components from multiple vendors. It makes use of several standardized physical and data link layers, as depicted in figure 2.4. This allows multiple network types, including Attached Resource Computer Network (ARCNET), Ethernet, Master-Slave/Token-Passing (MS/TP), LonTalk and Point-to-Point (PTP). Additionally BACnet also supports, Zigbee, a set of IEEE 802.15.4-based application specifications, and IP communications through the BACnet/IP specification [9].

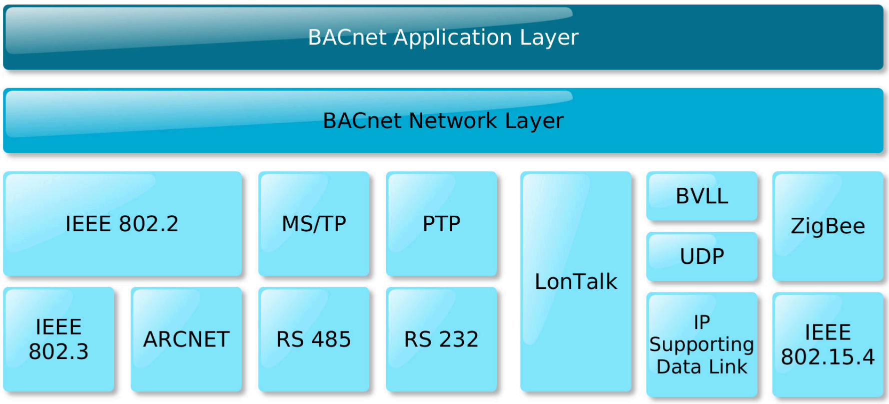


Figure 2.4: BACnet layers.

On the network layer, due to BACnet restrictions such as maximum message length, a specific protocol or adaptation, called the BACnet Virtual Link Layer (BVLL) is used. This allows to present a view of some network topology and function to the existing BACnet network layer. As an example, the BACnet/ZigBee Data Link Layer (BZLL) defined in Addendum 135-2008q [17], performs the adaption to the network layer between Zigbee and BACnet.

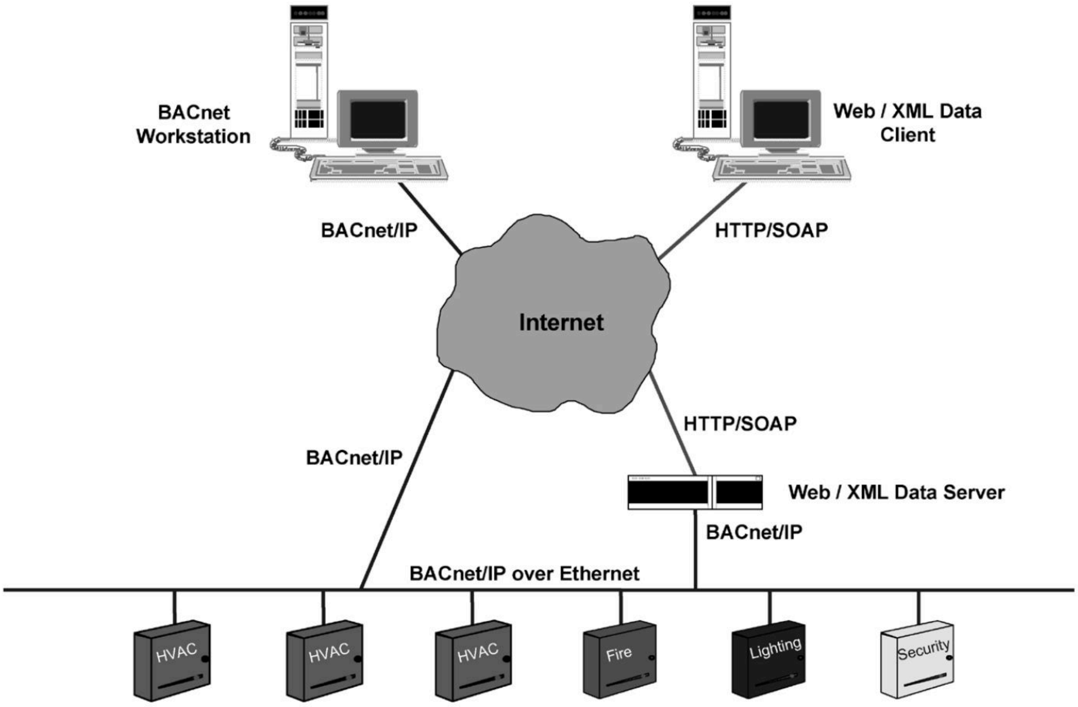


Figure 2.5: BACnet/IP configuration example [9].

Furthermore, BACnet/WS was introduced in 2006 [18], extending the BACnet standard to allow the integration of other systems using web services, which enables the access and manipulation of data in a BACnet server. Figure 2.5 shows an example of a BACnet/IP configuration, where both a BACnet/IP workstation and another one using web services can be used for accessing the BACnet network of devices.

The BACnet protocol defines an object-based access to BACnet devices, i.e. all the information is represented as objects [19]. Each object contains information relative to a function, divided in data elements called properties of the object. For example, a temperature sensor can have multiple properties related to temperature reading, so its not only possible to read the current temperature value, but also the type of the value (units) and maximum and minimum values. Several object types are defined by BACnet, but new objects or properties can also be added, without interfering with similar ones, by freely obtaining a vendor ID from ASHRAE.

The objects are then used by BACnet services, which define how to access and manipulate the objects. BACnet defines various services grouped into five categories: Alarm and Event, File Access, Object Access, Remote Device Management, and Virtual Terminal [9].

As examples, there are defined services for reading and writing object properties, namely *ReadProperty* and *WriteProperty* which are included in the Object Access category, as well as services for BACnet device discovering, which is the case of *Who-Has* and *I-Have* that belong to the Remote Device Management category.

In short, BACnet protocol offers enough services to try to entirely cover any building automation applications, making it an excellent choice in the development of complete building automation systems. However, the fact that it allows each manufacturer to focus on its own devices, and so their specific functions and tools, resulted in a big variety of tools. This can be seen as a downside when devices from multiple manufacturers are chosen [20].

2.2.2 LONWORKS

LonWorks is an open networking solution for building control and automation. Initially developed by Echelon Corporation in 1988, the communication protocol behind it, called LonTalk, was presented for standardization to ANSI/Consumer Electronics Association (CEA) and accepted as a standard for control networking (ANSI/CEA-709.1-B) in 1999, and later was also accepted as ISO/International Electrotechnical Commission (IEC) 14908. Currently maintained by LonMark International, an industry association, LonWorks has gained international recognition and is supported through a multitude of standards, through different application domains [21] [22].

The main objective of LonWorks is to provide a decentralized communication platform which can also be dedicated to the automation of several building subsystems, including HVAC and lighting. In fact, the peer-to-peer style of communication is one of the main differentiation factors when comparing with device networks like DeviceNet, Profibus and Modbus. Thus, instead of having a master device, through which all information flows, a LonWorks device can exchange data directly with any other device on the network, enabling basic automation operations without need for a central controller. Furthermore, it can use different types of communication media, namely Twisted Pair (TP) cables, Powerline (PL), Fiber Optic (FO), Radio Frequency (RF) or even exploit IP tunnelling mechanism with LonWorks/IP [11].

LonWorks is an event-triggered control network system, which consists of a dedicated controller (Neuron Chip), the physical medium transceiver, a network management tool and the communication protocol, LonTalk [19]. The prime component of each network device (called node) is the Neuron Chip, which provides the intelligence and networking capabilities to any device. It implements the entire LonTalk protocol stack and is comprised of multiple Central Processing Units (CPUs), memory, Input/Output (I/O), communications port, firmware, and operating system. Commonly Neurons comprise three 8-bit processors, from which two are responsible for the communication protocol execution, while the third provides all the application functions. The 8-bit structure is due to historical reasons when the chips were still licensed from Echelon, however, currently software code of this functionality also exists for 32 bit microcontrollers [19].

Figure 2.6 represents a typical LonWorks network in a building automation scenario. The nodes are split by network segments which are interconnected by routers and repeaters.

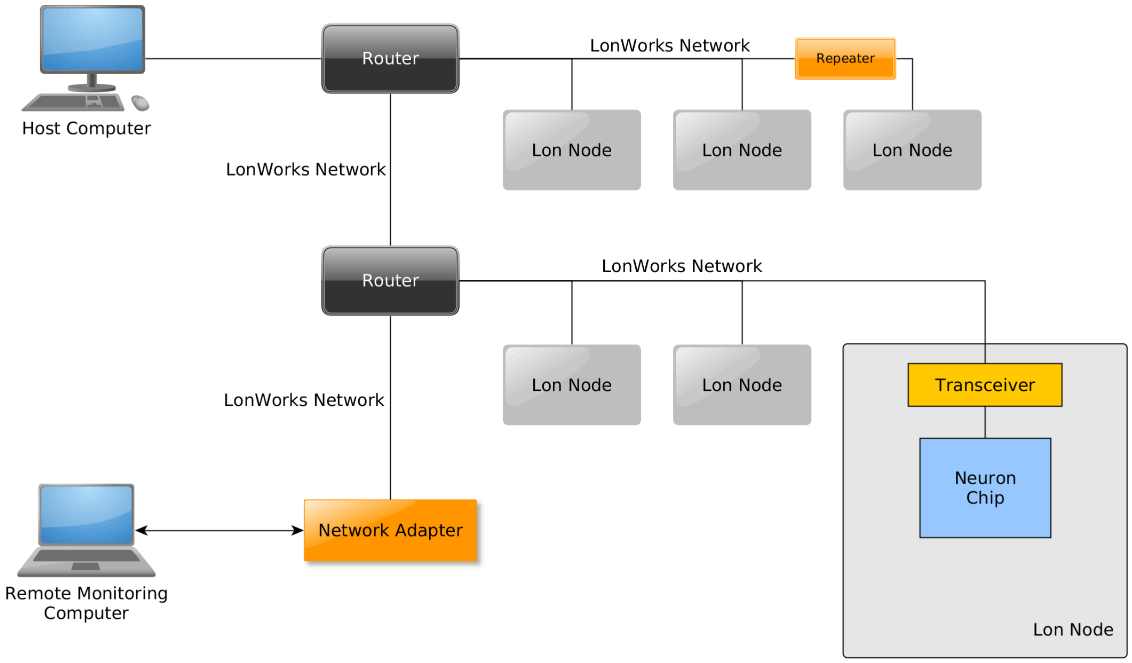


Figure 2.6: Example of a LonWorks building automation network.

Gateways are also often used to provide either remote connection or interfacing existing building systems.

Another great benefit of LonWorks is the use of Standard Network Variable Types (SNVTs), which are used to describe physical device's properties or parameters. As an example, an area SNVT, which is defined in square meters, has an index 110, is of type unsigned long and ranges from 0 to 65.535, with a resolution of 2 square centimetres [23].

Despite LonWorks huge potential to be a dominant solution for building automation systems, some issues led to a low market acceptance in some countries. Particularly, the high cost associated not only to the initial acquisition and tools but also extra costs for the per node royalty. Sometimes the inability for vendors to interact with some devices due to proprietary implementations of communication objects also deterred LonWorks widespread [20].

2.2.3 KNX

KNX is the result of combining the best aspects of three technologies for home and building control: European Installation Bus (EIB), BatiBUS and European Home System (EHS) [9]. With more significance in European market, KNX conforms to international ISO/IEC 14543 , European Standard (EN) (EN50090) and Chinese (GB/T20965) standards [24]. The standard is maintained by the industry association, KNX Association, and since early 2016 has been made open free for basic subscribers [14].

Similar to LonWorks, KNX can also be used over different communication mediums. In this case, TP, PL, IP/Ethernet (IP tunnelling) and RF (known as KNX) can be used RF [19]. As a field bus system, the main concept of KNX is to have a hardware independent solution that enables interoperability of field devices. KNX certified products are guaranteed to interconnect each to other. However, to ensure that the proper input is associated with the correct output action, thus enabling automation, requires a fairly complex configuration. For this configuration the Engineering Tool Software (ETS) must be used. Historically this was a complex process, where an ETS expert technician was required, sometimes even in the most basic installations. This made the KNX Association release two methods for configuration, as shown in figure 2.7 [25]. The simple E-mode is meant for installers with basic know-how, given that compatible components are already pre-programmed and loaded with a default set of parameters. The more complex S-mode is oriented for expert installers and technicians as it requires the full capabilities of the ETS platform, but it also provides the highest level of flexibility for configuring devices.

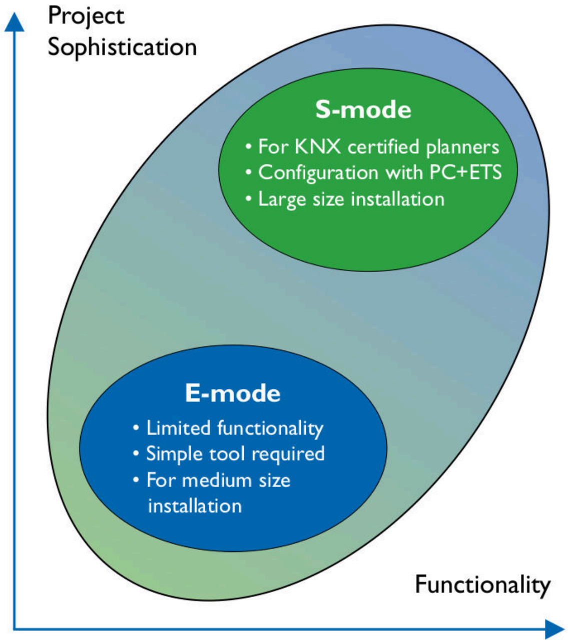

Project
Sophistication
S-mode
• For KNX certified planners
• Configuration with PC+ETS
• Large size installation
E-mode
• Limited functionality
• Simple tool required
• For medium size installation
Functionality

Figure 2.7: KNX configuration methods mapped against functionality and complexity [25].

A KNX network of devices is somewhat similar to LonWorks, it basically consists of zones which are connected through a backbone line. Each zone is comprised of lines that can have at maximum 254 devices. Yet, more devices can be added to zones with the use of sub-lines connected to the main ones via routers. Every KNX device has a unique individual address, corresponding to its position in the network (zone/line/device), which is used for unicast addressing. Multicast addressing is achieved through an additional group address, and is, in addition to its propagation mechanism, highly efficient. Furthermore, KNX communication follows an event-driven approach, with individual nodes making use of a shared variable model.

This allows the combination of devices into a group that can also be addressed as a group object, for read and/or write operations [9].

Summing up, KNX is a complete and open system for building automation, with the advantage of having a very rigorous device certification program, which has been leading to a reduction in manufacturer specific communication objects, and also in fewer flavours of management tools. In fact, the ETS platforms available are all derived from the one provided by the KNX association.

2.2.4 DALI

The DALI is the outcome of a combination of specifications from multiple lighting ballast manufacturers in the mid-1990s. Initially available in products as an industry standard, the DALI specification was adopted as standard IEC60929, annex E and G, in 2002. However, this early publication only considered the general definitions and the specifics for control gears (common ballasts), leaving out any other control device type. Recently, in 2014, a revised and extended version of DALI was published under IEC 62386, above all, introducing the capabilities to interface an extended number of devices such as motion sensors, switches, etc. [15].

In general, a DALI system follows the master/slave principle, where each controller(master) can address 64 devices (slaves) through an assignable unique address. The slaves can be divided in up to 16 groups, where each of them can have 16 different scenes [26]. It allows the control of the devices individually using the slave unique address, or the control of an entire group of devices using the group address. Additionally, it supports broadcast addressing, which makes it possible to control all units simultaneously. The protocol also provides several advantageous options, such as the identification of unit types, automatic search for control devices, simultaneous dimming of all the devices when a scene is selected and integration of emergency lighting [27].

DALI is a two-way communication system that runs on top of dedicated wiring. Its control structure resorts to specific commands that define the interactions between the controller and the devices. Despite being one of the most complete digital solutions for controlling luminaires, DALI does not allow implementation of automation rules, it rather defines how to interface the devices. In practice DALI gateways perform the connection to the automation and/or management platforms.

2.3 BAS SOLUTIONS

This section aims to briefly introduce some of the most proliferous BAS solutions currently available, focusing on their features and their supported protocols.

In fact, nowadays there are not that many market available products with the intent of reducing energy consumption in buildings. It is a consolidated market that comprised of strong players, which have developed and maintained their systems over the years. In fact, this goes in line with the long life-time required of any BAS solution. For instance, Siemens provides Desigo, APOGEE and Synco, three similar systems targeting different regions and different building sizes (Desigo) [28]. There is also Johnson Controls, who claims to have the world's leading BAS system, the Metasys [29], and Honeywell that provides a few separated but easily integrated systems, targeting building automation [30]. However, these reference products are often subject to significant customization depending on the project requirements for which they are used. They all claim to support most of the open standard protocols discussed in section 2.2 such as BACnet, LonWorks and KNX. However, as mostly used in business-to-business scenarios, and having closed source, the actual details and technical information is hard to obtain and thus will not be discussed in further detail, with the exception of the solution presented in subsection 2.3.1, due to its similarity with this dissertation's goals.

Additionally, there are multiple open-source platforms enabling home automation. That is the case of openHAB [31], Home Assistant [32], DomotiGa [33] and Domoticz [34]. These actually provide several interesting features, allowing to observe and control devices from various different technologies and systems. Furthermore, some of these platforms even provide automation based on basic rules, such as dimming lights when starting to watch a movie, or turn on your computer when you arrive home. However, those are basic rules made to work with a small amount of information, and do not support complex correlations between the information. In addition, they are designed to deal with a few events per minute, as is common in the residential segment. When compared to commercial and industrial buildings, houses are much smaller places and contain a limited number of occupants. In a professional building, there could be hundreds of people working, generating thousands of events per second, and fast responses are required in order to maintain comfort and productivity of their occupants.

2.3.1 THE EDGE: A PHILIPS AND DELLOITE PROJECT

All the different proposals of BASs converge to one common goal: create solutions that allows, through sustainable methods, providing a productive, comfortable and efficient environment to its occupants. Philips was responsible for the development of a BAS for the Edge building, in Amsterdam, for the consulting firm Deloitte. This project had, among others, the goal of increasing the buildings efficiency, by allowing for instance its occupants to, using a simple mobile application, control their workspace's light and temperature conditions, in order to adapt it to their personal preferences.

This solution includes, all the 15 floors which form the building, about 6500 LED luminaires, with half of them having sensors integrated and connected to the management system, the Phillips Envision Lighting System Management. This enables collecting data for posteriorly being analysed and re-used for improved management of the energy consumption.

Each luminaire is connected to the management system using an Ethernet cable, which allows support of IP to the end-node, and thus being individually addressable. Along with the use of Power-over-Ethernet (PoE), a combination of both energy and data in one single cable is achieved, eliminating the need of separated cables. This also simplifies the whole system in terms of protocols, since only the IP protocol is used for data transport.

As stated before, using a mobile application developed by Philips, each occupant may adjust the temperature and lighting levels relative, for instance, to a specific desk assigned to him (this is particularly interesting since there is no pre-defined desk for each occupant). To identify the desk, the mobile application makes use of Visible Light Communication (VLC). The LED luminaires emit a light beam with their location information encoded into it, that information is retrieved through the smartphone's camera and may be used to query a server on the user exact location [35].

By collecting all the data through the luminaires, the system is able to make data analytics, and then make suggestions in order to increase even further the efficiency. As an example, if a specific floor does not register any activity at Friday's afternoons, the system will suggest a shutdown of the lighting and HVAC systems at that time.

Moreover, this building, opened in 2015, was classified in 2016 as Outstanding by Building Research Establishment Environmental Assessment Method (BREEAM), being the most sustainable building until then, with a 98,36% score [36].

2.4 INTERNET OF THINGS (IOT)

2.4.1 CONCEPT

Internet of Things is a vision that has been evolving over the recent years, where in addition to the devices we usually see connected to the Internet, like our personal computers and smartphones, a large diversity of new device types will also be connected, forming an huge network of smart objects. These objects can vary from everyday devices like vehicles, televisions and music systems, to more constrained devices equipped with sensors and/or actuators which can sense, collect, transmit, analyse, and distribute data on a massive scale.

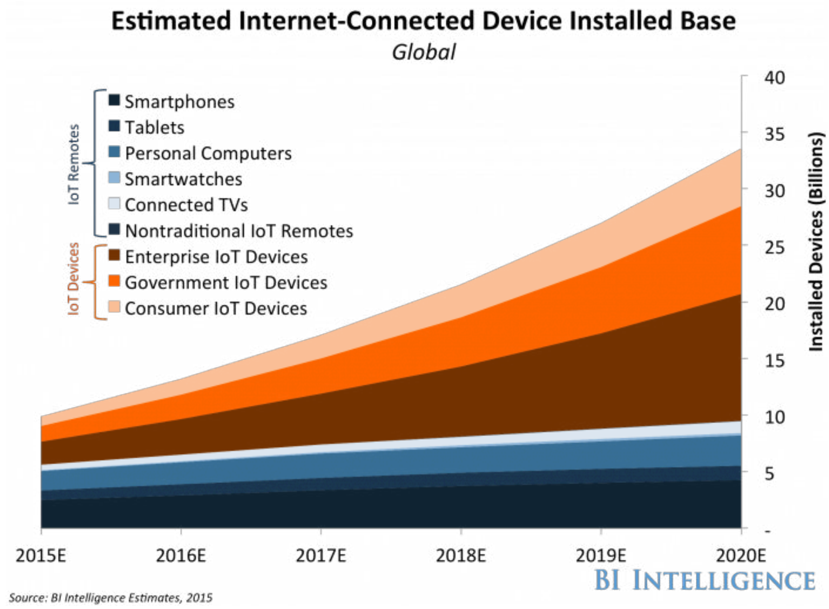


Figure 2.8: Global Estimated Internet-Connected Device Installed Base [38]

Considering the way people process information, this gives humanity the opportunity to have the knowledge to make better decisions [37].

The number of devices connected to the Internet will grow drastically. We can see in figure 2.8 that, by 2020, 34 billion devices are expected to be connected, with 24 billion of them being IoT devices. Considering the continuous increase of world population [39], we can verify that there are more devices than people connected to the Internet. In fact, we can also infer that, while nowadays the number of devices per person is around 2, in 2020 that number will increase to a little less than 5. While this number still seems low, we must consider that nowadays not all the world population has access to the Internet. According to [40], in 2015 only 43.4% of the world's population had access to the Internet, predicting this number to increase to 60% in 2021. With that in mind, the number of devices per person for 2020 should indeed be greater.

According to some authors [41], we will be surrounded by networks of interconnected devices, which will be providing content and services, to empower new ways of working and interacting, improving our comfort and quality of life. In fact, it is predicted in [38], that governments, which will be the second-largest IoT adopters, will mainly focus in increasing productivity and decreasing costs, but also in improving citizen's quality of life. Yet, businesses will be the leading adopter of IoT solutions, again by trying to increase productivity and decrease costs, but also by expanding to new markets with new product offerings.

The IoT concept was introduced by Ashton in 1999 [42], through exposing and presenting new business solutions requiring an interaction between Radio-Frequency Identification (RFID) devices and the Internet [43]. This initially simple concept has stood out in the recent years, essentially due to the overcome of some initial obstacles, such as the devices themselves which are now smaller, low powered and with low cost.

A few years before [44], Mark Weiser in an attempt to create a human-to-human technological interface, developed a similar concept, ubiquitous computing [44]. It consisted in combining the physical world and the technological world, through the interaction with sensors, actuators, displays and computational elements, incorporating technology in the everyday life of individuals [45]. It supports the idea that computing tasks which are typically done in a common desktop computer, can be made in several devices embedded into everyday objects, the smart objects. A smart object is defined in [46] as an item equipped with 4 main components: a sensor or actuator which is the gate for interaction with the physical world, a microprocessor that allows processing data obtained from the sensors, a communication device for exchanging information with other smart objects or other types of devices, and a power source to provide electrical energy to the device.

While both ubiquitous computing and IoT try to focus on the interaction between smart objects and humans, the IoT also focus on smart objects virtual representations and how they exchange information with each other and other platforms from the outside world. That is, the existence of methods for automatic identification using a unique and machine readable Identifier (ID), and the presence of standard technologies and communication protocols, as well as their security.

To finalize, IoT will actually change the world as we know today, to a better one. With the potential to automate quotidian tasks, improve comfort and decrease costs, IoT will represent the next evolution of the Internet, opening boundless possibilities for solution creation in several areas.

2.4.1.1 M2M

Machine-to-Machine (M2M), is a term frequently used over the past few years, often in association with IoT. Its concept refers to communication between machines, which are computing devices that perform specific tasks, in order to exchange information and perform actions, without the need for human intervention or interaction.

It is also not a new concept considering the fact that we already see M2M devices in several applications, mainly in telemetry, industrial, automation and Supervisory Control And Data Acquisition (SCADA) systems.

However, with the IoT revolution the concept of M2M has been enriched, essentially in the sense of giving meaning to the information exchanged by the machines in order to autonomously perform actions.

Ultimately, M2M should not be used as an IoT synonym, but as a subset of it. While M2M refers to the device-to-device only communications, IoT is a broad network, not only of endless M2M networks, but also of applications and interactions that can perform, for instance, data analytics and decision making. Additionally, IoT is not only about interacting with connected objects, it also allows interactions with non-connected objects such as objects with RFID and Near Field Communication (NFC) tags, which may use our smartphones as gateway to the IoT, just like printed bar codes or Quick Response (QR) codes.

In short, while M2M can be better seen from a vertical and closed perspective, IoT encloses a horizontal and meaningful approach where all the vertical applications are joined together in order to create and provide solutions, either in industry or for people and their environments.

2.4.2 WIRELESS SENSOR AND/OR ACTUATOR NETWORKS (WSAN)

A sensor is a device capable of detecting events or changes in its physical environment, and provide that information. Temperature, humidity, luminosity, motion and pressure are just examples of the countless physical or environmental data that a sensor can read. In contrast, an actuator is a device that, based on its input data, triggers or controls a mechanism that performs actions upon the physical environment.

Both sensors and actuators, when included in a broader device, along with a power source, a microcontroller and a transceiver, constitute what is called a mote, which is also known as a sensor node. Note that these components are enough to constitute a smart object as defined in the beginning of this section 2.4, being the communication device in this case a transceiver. Typically, these devices are designed to be small and have low power consumption, resulting in limited hardware, and thus constrained resources. Hence, operating systems capable of running on hardware of this nature are needed, being “TinyOS” [47] and “Contiki OS” [48] the most widely adopted. Considering these aspects, a sensor node is capable of collecting information or making actions, process that information and communicate through a wireless medium with other motes.

This network of sensor nodes properly distributed and linked by a wireless medium is what defines a Wireless Sensor and Actuator Network (WSAN), sometimes called Wireless Sensor Network (WSN) when no actuators are present. This way, a WSAN is capable of detecting and making several changes on multiple environments, by spreading diverse sensor nodes on it and cooperatively exchanging data with a central location, thus, allowing a remote and automated interaction with that environment.

With its first research in 1980, WSNs and WSANs have been used in diverse areas, being industrial automation one of its main adopters primarily due to its ability mainly to reduce cabling costs [49]. Nowadays, they are one of the primary enablers of the IoT, by converting their sensor nodes into smart objects through the use of IoT protocols and standards, discussed in the following subsection.

2.4.3 WIRELESS COMMUNICATION STANDARDS

The idea of having devices to "talk" to each other in the Internet of Things, is only possible if they all "speak" the same language. This is achieved with the establishment of communication standards, which, among other benefits, ensure interoperability between devices.

This section aims to present the most adopted standards for the Internet of Things, as well as other standards that evolved towards it. Given the fact that the majority of the devices in IoT will require a wireless connection, along with the fact that most of them will be battery-powered, only the standards targeting wireless connections for low power devices will be discussed. Additionally, the last subsection will present a brief summary of all wireless technologies discussed, and their key characteristics, by presenting a comparison between them in a table.

There are already several standards defined for the Internet, describing the protocols that have allowed the coexistence of endless applications even from different vendors and manufacturers. Part of these standards are defined and maintained by IEEE 802 LAN/MAN Standards Committee (LMSC), which is the group responsible for defining standards for Local Area Networks (LANs) and Metropolitan Area Networks (MANs) (the Institute of Electrical and Electronics Engineers (IEEE) 802 family of standards), targeting the two lowest layers of the Open Systems Interconnection (OSI) model. The OSI is a conceptual model that divides a communication system into 7 different layers, as shown in table 2.1. LMSC group also divides the Data Link Layer into two more layers: Logical Link Control (LLC) and Media Access Control (MAC).

Layer	Description
Application	High-level API that communicates with the operation system
Presentation	Data translation between application and networking service, including compression, encoding and encryption
Session	Allows session establishment between two nodes
Transport	Transmission of data segments between two network points
Network	Addressing, routing and traffic control in a multi-node network
Data Link	Ensures a reliable transmission of data frames between two nodes connected by a physical layer
Physical	Physical medium transmission of raw bit streams

Table 2.1: OSI model layers

IEEE 802 family have already specified a considerable number of standards, with some of them being widely used, such as the 802.3 for wired Ethernet connections, and for wireless connections, the 802.11 for Wireless LANs (WLANs) (commonly known as Wi-Fi) and the 802.15 for Wireless PANs (WPANs).

IEEE 802.11, is one of the most commonly used specifications nowadays. It is hard today to find a laptop or a smartphone without Wi-Fi built in. It allows us to access the Internet in a neat and easy way without inconvenient cables. With its first version launched in 1997, Wi-Fi is already expanded to a diversity of markets, always presenting new standards with new improvements, being one of their key priorities raising the network throughput, hitting its highest value with 802.11ac standard, supporting a data rate up to 6 Gbps [50].

One the other hand, in 1999 [51] another wireless technology specification was introduced, Bluetooth. Standardized by LMSC as IEEE 802.15.1, Bluetooth is currently maintained by the Bluetooth Special Interest Group (SIG). Over the years it has achieved its own market for data exchanging in short distances, especially on connections between mobile phones, wearables and electronic accessories. Unlike the IEEE 802 standards, which only specify services and protocols for the two lowest layers of the OSI model, Bluetooth has its own stack from the physical layer to application one.

Although both Wi-Fi and Bluetooth define solid specifications for providing reliable wireless communications, they were not initially designed having constrained devices in mind, and thus did not provide efficient means for having wireless communications with a low energy consumption and low capabilities devices. With that in consideration, another standard was specified in 2003 [52], the IEEE 802.15.4, from the IEEE 802.15 group.

2.4.3.1 IEEE 802.15.4

IEEE 802.15.4 was the first standard targeting Low Rate WPANs (LR-WPANs) by specifying the physical and MAC layers for them, with lower data rates, simple connectivity and battery saving in mind. With its first version in 2003, standardization process continued and improved versions of the standard have been launched in the latest years [53].

It is due to IEEE 802.15.4 that is possible today to replace most wired sensors with wireless ones and thus creating the WSANs. In contrast to IEEE 802.11, IEEE 802.15.4 has smaller payload, simpler modulation, less frame overhead and better power management mechanisms [54]. This allows sensors and actuators to operate using batteries that last months or years. This makes IEEE 802.15.4 one of the main enablers of the Internet of Things, as it provides the base structure for allowing constrained devices to be connected to the Internet.

Furthermore, IEEE 802.15.4 is designed for several types of LR-WPANs, where battery friendly algorithms may be adopted in order to improve both the performance and the lifetime of device batteries [55]. In fact, it serves as base for other specifications, such as Zigbee and

Thread (described below), where different approaches are followed to develop the upper layers of the OSI model (except physical and MAC layers).

2.4.3.2 6LoWPAN

Since IEEE 802.15.4 only provides the lowest layers from the OSI model, other standards are needed in order to obtain a full communication stack. This lead to the development of IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), a standard defined by Internet Engineering Task Force (IETF). It is defined in [56] as a networking or adaptation layer that allows the efficient transport of Internet Protocol version 6 (IPv6) packets within small link layer frames.

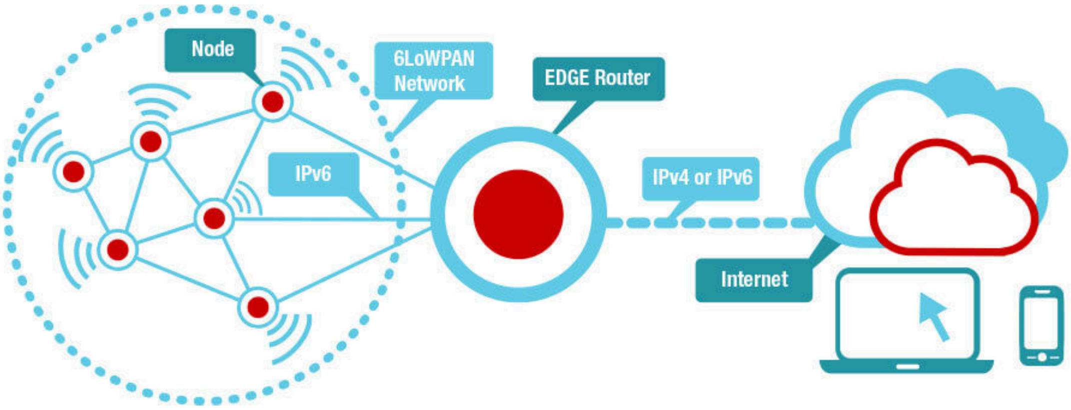


Figure 2.9: An example 6LoWPAN network connected to the Internet [57].

6LoWPAN offers several advantages for the Internet of Things. Its main advantage is clearly the fact that it allows the definition of IP-based networks, which allows 6LoWPAN networks to be connected to other networks (including the Internet) using simple edge IP routers, as shown in figure 2.9. Additionally, the use of IPv6 also brings some advantages over its predecessor Internet Protocol version 4 (IPv4), such as Quality of Service (QoS), mobility and multicasting, along with the fact that its use naturally avoids some problems related to IPv4. These are, essentially address range limitation, which already led into its scarcity more than once [58]. IPv6's addressing space is much wider, allowing even small devices to have a real-world reachable address. Plus, and also inherent to IPv6, 6LoWPAN has the stateless auto configuration feature, which allows a device to auto configure its address.

A not less relevant benefit is the fact that 6LoWPAN does header compression, fragmentation and reassembly in order to keep the packets size as small as possible, allowing the accomplishment of low rates and low power usage on their transmission.

Another extremely important feature, is 6LoWPAN's support for mesh networks, which allows having a robust, scalable and self-healing network of nodes. It is done by setting some nodes as routers, which are able to route data destined to other devices, allowing host nodes to sleep for longer periods of time, and a range extension of the network, since two nodes can exchange information without being in each other's range.

Regarding security, 6LoWPAN not only inherits the strong Advanced Encryption Standard (AES)-128 from IEEE 802.15.4 at the data link layer, but also supports the usage of Transport Layer Security (TLS) for Transmission Control Protocol (TCP) and Datagram Transport Layer Security (DTLS) for User Datagram Protocol (UDP). However, there are some security concerns still not addressed which makes 6LoWPAN not completely secure, such as the lack of Internet Protocol Security (IPSec), that could take care of the authentication and encryption of the packets at the network layer during a communication session [59].

2.4.3.3 ZIGBEE

ZigBee, developed by the ZigBee Alliance, is a protocol that, similarly to 6LoWPAN, uses the IEEE 802.15.4 standard as its base at the physical and data link layers, and adds its own unique stack on remaining layers (considering the OSI model). At the application layer, ZigBee provides ZigBee Cluster Library (ZCL) which is basically a repository of commands to be used in application profiles defined by developers.

Like 6LoWPAN, ZigBee also provides a mesh networking topology whose advantages have already been addressed in the previous subsection. It supports up to 64000 nodes in network with the multi-hop tree, multi-hop mesh or start topologies, where each node has a unique 16-bit short address, and provides several routing protocols in order for users to be able to choose an optimal routing strategy for their applications [60].

Just like Bluetooth, ZigBee has also conquered its own market. In fact, since it targeted constrained devices from the beginning, it dominates the low-power networking market. However, with the rapid evolution of the Internet of Things, other standards have been evolving towards this market (discussed in the following subsections). This in addition to the lack of IP-based networking of ZigBee, led the ZigBee Alliance to bring IPv6 network protocols to ZigBee. Therefore ZigBee IP was introduced, as an open standard, claiming to offer a scalable architecture with end-to-end IPv6 networking based on standard Internet protocols. These include 6LoWPAN, IPv6, Protocol for Carrying Authentication for Network Access (PANA), Routing Protocol for Low Power and Lossy Networks (RPL), TCP, TLS and UDP, to a create cost-effective and energy-efficient wireless mesh network [61].

Similar to 6LoWPAN, ZigBee IP also enables the establishment of self-configuring and self-healing wireless mesh networks through the use of several device types such as coordinators, routers, border routers and the regular hosts. Figure 2.10 shows an example network topology, where two devices are connected to a mesh network formed by a few router nodes and one coordinator, which is connected to the Internet using a ZigBee IP border router.

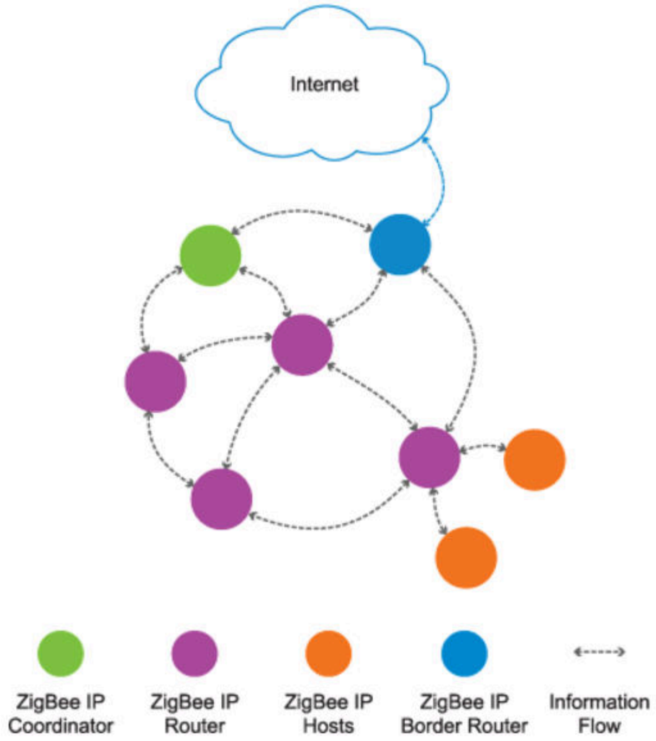


Figure 2.10: ZigBee IP example network topology [61].

2.4.3.4 BLUETOOTH LOW ENERGY

With the Internet of Things expansion, the low power sector has gained more attention. Bluetooth SIG noticed that, and presented an extension with Bluetooth specification 4.0. This was in fact a completely separate technology with a different stack and frequency map, taking advantage from previous Bluetooth versions only at the data models. This new specification was initially known as Bluetooth Low Energy (Bluetooth Low Energy (BLE)), still often used, and later with the brand name Bluetooth Smart.

With this new version, BLE became a high potential competitor for ZigBee in the IoT domain by providing, in relation to its previous versions, ultra-low power and low latency communication to low cost and small devices, while keeping a decent data rate.

Although the BLE component is not compatible with older versions of Bluetooth, recent devices support both single-mode implementations, named Bluetooth Smart, and dual-mode implementations, called Bluetooth Smart Ready, where the latter is able to communicate with both old versions and new versions of Bluetooth.

Interactions between Bluetooth Smart devices are made through the Generic Attribute Profile (GATT) protocol, which specifies how the data is organized in a device and the available operations to perform on it. GATT is divided in two roles: client and server. A GATT client is basically the entity that initiates requests and commands over a BLE link with the goal of performing operations on a GATT server or reading data from it.

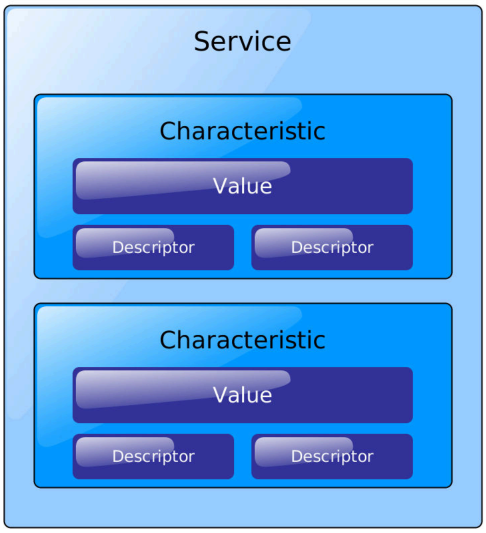


Figure 2.11: Bluetooth Smart service properties.

A BLE device can be a Master or Slave, but not both at the same time. A Slave, also known as peripheral, is a device that keeps broadcasting advertising packets and waiting for a connection from a Master device. Typically, a Slave contains a GATT server providing resources that can be accessed by a Master which runs a GATT client. A Master is able to scan for advertising packets from slaves, and then connect to them in order to establish the BLE connection and thus accessing their resources. It can connect to multiple slaves, while each slave can only be connected to one Master. For instance, a smartphone is able to listen for advertising packets from multiple peripherals such as smartwatches or a heart rate monitor, and connect to all of them.

A GATT server is responsible for storing information that can be managed by a GATT client, through requests, commands and confirmations. The attributes of the server are organized as multiple services, being each service also divided in several characteristics, as shown in figure 2.11. Additionally, and also presented in figure 2.11, each characteristic may have a value and several descriptors. A descriptor provides additional information about a characteristic, such as the units of its value or the format in which it is presented. Moreover, a GATT client can also make a request for receiving notifications from a characteristic whenever its value changes, which will cause the GATT server to asynchronously send the new value to the client.

All three types of attributes of a GATT server (services, characteristics, and descriptors) are identified by a 128-bit Universally Unique ID (UUID). In order to allow interoperability, Bluetooth SIG reserved a range of UUIDs and defined standard attributes. For instance, there are assigned UUIDs for several services such as Health Thermometer or Blood Pressure. The same applies to characteristics and descriptors, e.g the Characteristic Presentation Format defines the commonly known formats for a value, such as integer, float or string.

A major advantage of BLE over ZigBee, is its widespread integration in smartphones, tablets and even laptops, with a large share of them already equipped with Bluetooth Smart Ready. This makes it better suitable for some IoT applications such as health care and building automation, where the user's smartphone can be an integral part of the system, being able to interact with the other devices.

Additionally, with its most recent versions, Bluetooth 4.2 and Bluetooth 5.0, by taking advantage of the 6LoWPAN standard, Bluetooth SIG introduced new profiles, being one of them the Internet Protocol Support Profile (IPSP) which brings IPv6 networking to Bluetooth. This means that IPv6 packets can be sent and received by BLE devices, in addition to keep their core Bluetooth Smart capabilities.

In short, with the latest version of the specification, Bluetooth becomes one of the best solutions for many IoT applications, by providing decent data rates at acceptable ranges and low power consumption.

2.4.3.5 OTHERS

There are other open standards for wireless communication which despite not being fully or yet available, have potential to be adopted in the future. That is the case of Thread, which is an open standard, which also makes use of IEEE 802.15.4 standard and 6LoWPAN adaptation layer, to create secure, reliable and power-efficient mesh networks.

It is designed specifically for connected home applications, with special focus on easy installation, security, large networks support, high range and low power consumption [62].

Additionally, IEEE 802.11ah standard also known as Low Power Wi-Fi, and recently named Wi-Fi HaLow [63], is the evolution of Wi-Fi towards the Internet of Things. With an optimized physical and MAC layers, Wi-Fi offers extended range, power efficiency, and scalable operation, thus targeting constrained devices, and consequently, the IoT.

Although not yet released, Wi-Fi HaLow will have an extremely important feature: IP from scratch. While all other technologies such as ZigBee, Bluetooth and 6LoWPAN have been making some effort to add IP-based networking to their specifications, Wi-Fi HaLow will naturally include it. Plus, since wireless router companies constantly evolve their technology to support the latest IEEE standards, HaLow will possibly, and naturally, appear on our home's router.

Along with IP, Wi-Fi HaLow also claims to bring the same security existent in the well-established Wi-Fi standards, which is another feature of extraordinary importance.

2.4.3.6 WIRELESS TECHNOLOGIES COMPARISON

In this section a comparison between the so far discussed wireless standards is presented resorting to table 2.2, adding two more wireless technologies which, despite not being completely aimed to device-to-device communication, will have an important role in the IoT, those are RFID and NFC.

Briefly, RFID allows to uniquely identify items using radio waves through the attachment of tags on them. It supports both active and passive tags, where the key difference between them is the existence of a power source in the active tags. The lack of a power source in the passive tags (they are powered by the electromagnetic energy transmitted from the RFID reader) makes RFID a suitable technology for the IoT, allowing the identification of objects with an infinite battery life. Similar to RFID, NFC is new and more refined version of RFID, also allowing the read of tags, and operating in one of the RFID frequencies. Plus, NFC devices are able to act as both reader and tag which enables Peer-to-Peer (P2P) communication on NFC, a feature that made it a popular choice for secure communications such as contactless payment.

The existence of open standards allows the creation of interoperable solutions for the IoT. But the IoT market is not constituted only by open standards, and there are at least two other solutions for wireless communication already with a considerable share in this market, ANT and Z-Wave, and will thus be included in the comparison table. Similarly, EnOcean wireless technology also provides wireless communication between devices, and has the big advantage of being self-powered. However, these are all proprietary solutions. Nonetheless, and despite not being discussed in this state of the art, they will be included in the comparison table.

Considering this comparison, it is possible to conclude that most of the wireless technologies included can be suitable for different situations. Thread and Wi-Fi HaLow are still not fully available and thus, despite their potential, cannot be used in any situation for now.

NFC and RFID target a specific market where data transfers are made only between two devices. While NFC is considered an evolution of RFID, they are still both quite adopted in the market. RFID is suitable for situations where higher range is necessary, whereas NFC is mostly used for low range and secure operations such as data transfer between smartphones or credit card payments.

Regarding to the WSANs domain, proprietary solutions such as Z-Wave, ANT+ and EnOcean, provide mesh networks and have acceptable ranges and data rates, with exceptional battery life (infinite in the case of EnOcean). Yet, their price and lack of interoperability might turn them into undesirable options.

On the other hand, both ZigBee IP and BLE are suitable technologies for WSANs, such as for building automation, which is the target scenario of this dissertation work. In fact, ZigBee is already used in several sensor nodes in similar situations, while BLE is generally

	Topologies	**Frequencies**	**Max Data Rates**	**Max Range**	**Battery Life**	**IP Support**
ZigBee IP	Mesh Star Tree	2.4 GHz 915/868 MHz	250 Kbps	200 m	Years	Yes
BLE (4.2)	P2P Star	2.4 GHz	2.1 Mbps 800 Kbps(LE)	100 m	Months to Years	Yes
Wi-Fi HaLow	Star Tree	900 MHz	150 Kbps	1000 m	-	Yes
Thread	Mesh Star	2.4 GHz	250 Kbps	200 m	Months to Years	Yes
Z-Wave Plus	Mesh	908.42 MHz	100 Kbps	150 m	Years	Yes
ANT+	Mesh Star	2.4 GHz	20 Kbps	10 m	Years	No
EnOcean	Mesh P2P	868 MHz 902 MHz 928 MHz	125 Kbps	300 m	N/A	No
RFID	One Way	125 MHz 13.56 MHz 915 MHz	100 Kbps	100 m (Active TAG) 25 m (Passive TAG)	Years (Active TAG)	No
NFC	P2P	13.56 MHz	424 Kbps	20 cm	N/A	No

Table 2.2: Wireless technologies comparison

used in electronic gadgets such wearables. However, Bluetooth Smart has been evolving, and in its latest version has support for simple mesh networks and IPv6, while achieving extreme power savings and offering relatively higher data rates in comparison to its competitors. Hence, an interest in using BLE in constrained sensor nodes (and thus in WSANs) has emerged, and was selected for the SmartLighting as well.

2.4.4 PROTOCOLS

Internet of Things as described before, is a scenario in which “Things”, are connected to the Internet. In order to do that, and to be able to communicate with existing devices, they must communicate using the existing Internet protocols. However, most of the IoT objects are very constrained in their resources, such as memory and processing capacity, compared to the existing connected devices such as our smartphones and personal computers.

So, new and lightweight protocols are needed in order for constrained devices to be able to connect to the existing Internet, but adhering to the existing Internet Protocol Suite, presented on table 2.3, maintained by the IETF.

Since IoT devices seek different goals and, as stated before, have different capabilities, they also have different requirements. Then, their protocols must, mainly, support various connections, secure the information exchanged and be scalable. IoT protocols will be discussed

OSI Layers	DARPA Layers	Protocol
Application	Application	HTTP, FTP, SMTP, DNS RIP, SNMP, SIP, SSH, ...
Presentation		
Session		
Transport	Transport	TCP and UDP
Network	Internet	IP (IPv4 and IPv6), ICMP, ARP, ...
Data Link	Network Interface	Ethernet, 802.11 Wi-Fi, Frame Relay, ATM, ...
Physical		

Table 2.3: Internet Protocol Suite

in the following subsections, with the last one (subsection 2.4.4.5) comparing the protocols discussed, by summarizing their main aspects into a table.

2.4.4.1 MQTT

Message Queue Telemetry Transport (MQTT) is a lightweight messaging protocol, that follows a publish/subscribe model, and whose simplicity makes it suitable for constrained devices. It can work well on low bandwidth and high latency networks, and a single broker is able to support thousands of devices.

The publish/subscribe model is a message pattern where each device, instead of exchanging messages directly with another one, uses an intermediate server (the broker) to deliver the messages for it. In this model, every node is a client, and every client can publish messages on, or subscribe to, a topic. A topic is a hierarchical structured string, that can be used for

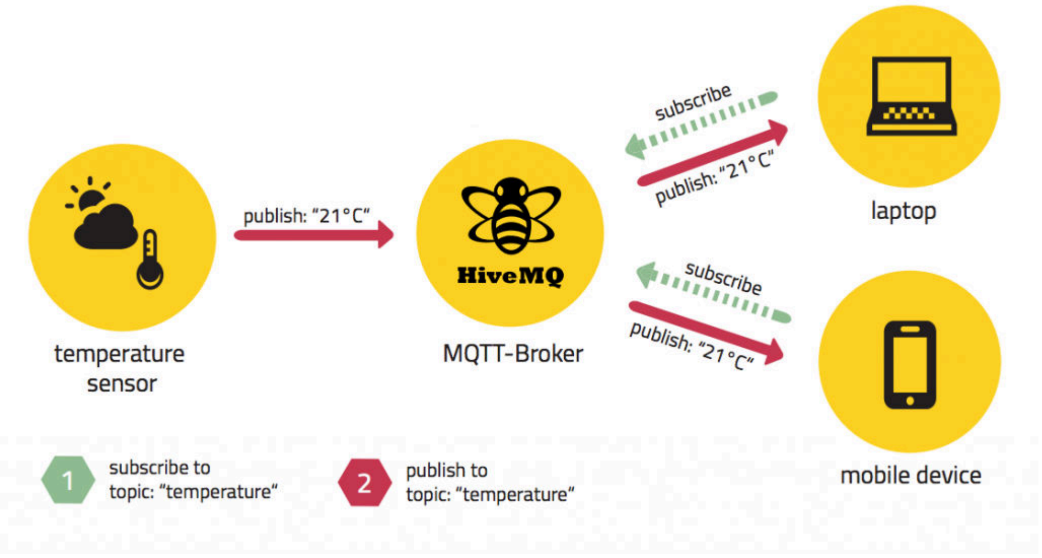


Figure 2.12: MQTT Publish/Subscribe model [64].

routing and/or filtering messages by the broker. This way, it is the broker's responsibility to distribute each message it receives to other clients, based on their topic subscriptions. Figure 2.12 helps conceptualizing this model, showing an example where two clients subscribe a common topic for receiving temperature information from a sensor.

Regarding security, although its specification does not provide any security at the transport level (despite some implementations [64] providing Secure Sockets Layer (SSL)/TLS encryption), it allows the authentication of packets using an username or password.

In relation to clients and its messages, some interesting options are available. One of them is the QoS, which allows setting higher levels of effort for the server to deliver the message, ensuring the message is delivered. The *Retained Message* option, enables the server to keep a message retained on server, for delivering on new client subscriptions. There is also the *Clean Session* flag, which sets the server to keep the client state (subscriptions and messages with high QoS), so every time the client disconnects, it can reconnect without losing its subscriptions and most important messages. To finish, there is the *Wills* option, which enables adding a message to be sent by the server to specific topics, when the client disconnects [65].

The not so good in MQTT, is the use of TCP protocol, which was designed for devices with decent processing resources and memory, where keeping a connection open all the time is not a problem. However, this is considered heavy for constrained devices, causing a higher energy use and thus, reducing their battery life.

Additionally, there is also the fact that the broker can be seen as a single point of failure, as all nodes require a connection to it. While this point of failure issue can be easily solved with the use of a load balancer and a cluster of MQTT nodes [66], addressing the TCP drawback is a harder task. Here, MQTT for Sensors Nodes (MQTT-SN) can be used, which allows the use of UDP with MQTT.

MQTT-SN is a protocol for constrained devices, as well as a variant of the MQTT. It is a rebuild of MQTT in order to address issues concerning to constrained devices, such as lossy networks, sleepy devices, low power devices and low bandwidth. It allows the use of UDP (although not limited to it) as the transport protocol, resulting in a more lightweight communication by removing the TCP handshakes and the need for a constant connection [67]. In addition to resolving the issues already stated, concerning to constrained devices, there is another advantage, which is the fact the it uses a topic ID instead of a long topic string that each MQTT client has to send on every message, resulting in smaller message sizes. However, clients using MQTT-SN protocol still need to connect through a gateway, which makes a bridge between the MQTT-SN packets and MQTT ones, which may also be seen as a drawback by adding more points of failure.

To summarize, MQTT is an open protocol suitable for the Internet of Things that offers an extraordinary performance [68] and various features, while working at minimum bandwidth, supporting real push notifications and near real time communication.

2.4.4.2 COAP

Having in mind the continuous growing of the IoT, and how important it has become, IETF defined Constrained Application Protocol (CoAP) [69]. In contrast to MQTT, which being a publish/subscribe protocol supports all kinds of relationships in the devices interaction, CoAP is an application layer protocol that follows a client/server model, which defines one-to-one relationships. Nonetheless, broadcast and multicast addresses are also supported, though with some limitations, by CoAP.

CoAP was designed to support Internet of Things from the beginning, by particularly targeting constrained devices. Therefore, it uses UDP as the transport protocol, mainly to take advantage of its connectionless and efficient communication. Here, each packet is sent individually without any handshaking, congestion control or any other mechanism present on TCP protocol. In addition, the non-existing or little fragmentation due to the small fixed header (4 bytes) and compact encodings specified on CoAP, makes it possible for devices to save power by waking-up faster and remaining more time in a sleepy state, since they take less time to exchange information.

CoAP can be seen as the well-known Hypertext Transfer Protocol (HTTP) but for constrained devices of the Internet of Things. In fact, it is also a document transfer protocol, and it is based on the Representational State Transfer (REST) model, where servers provide resources that are accessible by clients using the GET, POST, PUT and DELETE methods. Additionally, since both protocols share the REST model, they can be connected simply using application-agnostic cross-protocol proxies [69], giving a totally interoperable and transparent network as shown in figure 2.13.

CoAP also presents other interesting features such as resource discovery, in which servers provide a list of their resources so the clients can discover them, and the observe flag, that can be set on a GET request in order to inform server to keep replying on resource state

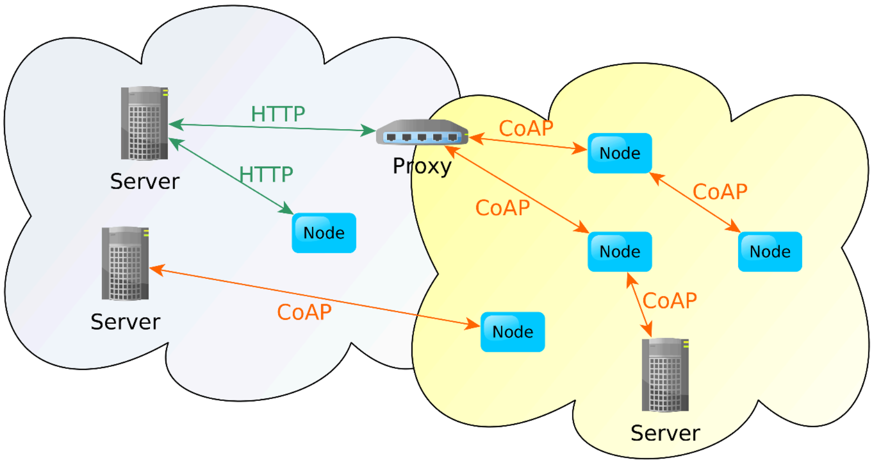


Figure 2.13: IoT with HTTP and CoAP

changes. This is actually analogous to the subscribe operation from MQTT, where instead of subscribing a topic, a device can observe a Uniform Resource Locator (URL). Indeed, it is possible to have a similar publish/subscribe model on CoAP, and actually a draft was already submitted [70].

Because IoT devices will be connected to the Internet, and because they might carry important and private information, they will consequently become a target of exploitation just like any other device we use nowadays. Hence, CoAP allows securing that information using DTLS, which basically provide similar security to UDP as TLS provides for TCP.

In short, CoAP is a lightweight protocol built for the Internet of Things that provides an excellent performance [68], and minimizes the power usage on constrained devices, which is very important for the growing IoT market.

2.4.4.3 XMPP

While not conceived for the Internet of Things, the Extensible Messaging and Presence Protocol (XMPP) [71] is a message-oriented protocol developed for Instant Messaging (IM), that is quite adopted on IoT. Initially named "Jabber", XMPP sought to approach not only near real-time IM, but also presence information and contact lists.

While running over TCP protocol, XMPP has some key features to offer to the IoT world, such as a simple email like addressing scheme e.g. *username@example.com/mobile*, both publish/subscribe and request/response models, security and authentication. Additionally, its decentralized nature, in the sense that anyone can have its own server, ensures high scalability.

Another major advantage of XMPP, is its extensibility. With its eXtensible Markup Language (XML)-based approach, high flexibility is allowed for creating multiple extensions to the protocol.

In contrast, despite this XML-based messages allowing high extensibility, they are text based and thus cause an higher communication overhead. Nonetheless, this is already being addressed, and should soon be solved by the Efficient XML Interchange (EXI), which is an efficient way to compress XML documents and fragments [72].

Briefly, although not designed for the Internet of Things, the high XMPP extensibility allows it to be easily adapted for new applications, and recent work have been making it suited to run in constrained devices, enabling it to join the IoT protocols suite.

2.4.4.4 AMQP

The high efficiency of a protocol is also attention worthy, and although a considerable amount of applications on the Internet of Things do not require reliable communications or a guaranty that a message was delivered, others do. For instance, in an application where a temperature sensor periodically reports its current value, the loss of one read is not really an issue. First, because another will be sent again shortly, and second, the probability of that lost value having a significant variation is low. However, in an health care application for example, the delivery assurance of an emergency message is crucial. This is where Advanced Message Queuing Protocol (AMQP) [73] comes up on the IoT.

AMQP is an open standard protocol, with a publish/subscribe model, designed to support message-oriented environments efficiently. It offers a reliable communication with message delivery primitives, in order to ensure each message arrives its destiny even if a failure or a reboot occurs. AMQP broker is mainly composed by exchanges and queues, as shown in figure 2.14, where an exchange is responsible for routing messages into queues according to predefined rules and conditions. A queue, is where messages are stored before being consumed by an application. The broker has one queue per consumer, which is deleted when the connection closes, and will survive even the broker restarts, ensuring there are no lost messages.

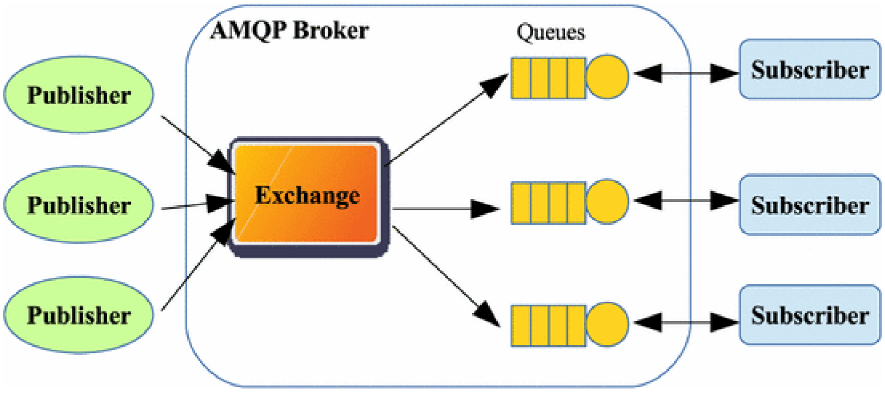


Figure 2.14: Publish/subscribe mechanism of AMQP [74]

Also, AMQP features high interoperability since it is a programmable protocol in the sense that its entities and routing schemes are defined by the applications themselves and not by a broker administrator. Plus, it is also very extensible and flexible by supporting custom exchange types, additional attributes on the queues such as a message Time To Live (TTL) per queue, and broker additional extensions and plugins.

Regarding security, the protocol uses TCP for a reliable transport of messages, allowing the use of SSL/TLS to secure all the information exchanged.

Summarizing, AMQP is an efficient protocol with special focus on not losing messages, able to process thousands of queued transactions per second. It can be seen as a more advanced version of MQTT, making a better and efficient use of resources and is thus quite adopted in the IoT, especially when security, efficiency, reliability and performance in delivering messages is needed.

2.4.4.5 PROTOCOLS COMPARISON

This section aims to show a comparison of the protocols discussed, with their main aspects summed up in table 2.4. This is followed by a brief on which situations each of them is more suitable.

	MQTT	**CoAP**	**XMPP**	**AMQP**
Publish/Subscribe	Yes	Yes*	Yes	Yes
Request/Response	No	Yes	Yes	No
Transport	TCP (UDP on MQTT-SN)	UDP	TCP	TCP
Security	Username/Password Authentication SSL	DTLS	TLS and SASL	TLS and SASL
QoS	Yes	Yes	No	Yes
Dynamic Discovery	No	Yes	Yes	No
Encoding	Binary	Binary	Plaintext	Binary
6LoWPAN	Yes	Yes	No	Yes
Real time	No	No	Near Real Time	No

Table 2.4: IoT Protocols comparison.

XMPP is a powerful protocol offering extended security and a near real time service, with the ability to use both publish/subscribe and request/response models. However, it is still not suitable for constrained devices due to its text based messages which cause a communication overhead. Yet, as already stated in its section, EXI format is already proposed [72], aiming to compress both XML documents and fragments in order to be used in resource constrained networks, such as in 6LoWPAN.

Regarding CoAP, it is basically for constrained devices, what HTTP is for regular ones. Despite supporting a publish/subscribe model (using the observe flag), it is primarily a request/response protocols and thus is more suitable for state transfer models.

MQTT and AMQP are both efficient message queuing protocols, following a publish/subscribe model, making them both suitable for event based situations. MQTT is simpler and its clients are easier to implement. It is suited for applications where simple clients communicate through a local and trusted network. However, when both security and reliability of message delivering is needed, as well as increased performance is needed at the broker, AMQP can be an asset.

2.5 AUTOMATION LOGIC

The application logic components are the central unit in any automation system. They are responsible for every action that is sent to the environment to be automated. They operate by receiving all the events from the environment, and generate actions accordingly. An event, can be any change detected in the environment, such as a movement, a change in temperature, a smartphone connection or even simply a phone ringing.

When the environment to be automated is relatively small, such as in a vehicle equipped with sensors and actuators, a low number of events will be generated and thus the application logic can be achieved using usual techniques with typical platforms. On the other hand, when the target environment has a considerable size, such as in a building, a huge number of events can be generated, and in this case a regular application will not be able to process all of them efficiently.

This calls for platforms that are able to process large streams of events in the minimum amount of time. Those will be addressed in the section 2.5.1. Additionally, implementing efficient automation logic on these high performance platforms might be a difficult task, where the use of efficient methods for event processing can be an asset. Section 2.5.2 introduces an efficient approach on this matter.

2.5.1 HIGH PERFORMANCE AND REAL TIME PLATFORMS

High performance and/or Real-Time Computing (RTC) platforms are combinations of software and hardware, that aim to give the highest performance and thus the minimum response times to applications. They normally take advantage of distributed computing, which consists on software capable of dividing a task into several and distribute them for multiple nodes. Each node is then responsible for processing its part of the task and by communicating and coordinating with the other nodes, they all contribute to achieve a common goal. This enables to process data in parallel, allowing to achieve the minimum processing time.

Regarding processing methods, there are generally two to be adopted: batch processing and stream processing. Batch processing is the process of collecting high volume data at once, process it and produce results. This processing type is mostly concerned about the produced results than the latency. In fact, it could take minutes, hours or even days to process. Batch processing can be useful in big data applications where processing time is not an issue such as analysis of operational, historical, or archived data.

On the other hand, stream processing tries to achieve the minimum processing time, providing real time or near real time processing. The data is collected from continuous streams of data and each data element is immediately processed in order to produce results as fast as possible.

Additionally, some systems try to deliver stream processing by offering micro-batch processing, where the approach is the same as in batch processing but for much smaller data batches coming from small time windows.

This section aims to let know the current solutions for high performance and real time processing, by briefly describing the most adopted open-source platforms at this field and, at the end, also briefly refer the most known commercial solutions.

2.5.1.1 APACHE PROJECTS

Apache Software Foundation (AFS) [75] has several open-source frameworks in this field. Apache Hadoop [76], an open-source distributed processing framework, was the first earning a reputation as a big data analytics engine. With its first version released in 2006, Hadoop uses batch processing to process large datasets across computer clusters, offering fault tolerance and good scalability, from a single node to thousands, each of them providing local storage and computing capabilities. It can even be used on low cost clusters with nodes having lower memory and processing capabilities, such as Raspberry Pi computers [77], where although the performance is, as expected, lower than in a cluster that uses traditional computers, it can still provide a high performance system with a variety of uses.

Hadoop is primarily composed by Hadoop Common, which provide a set of utilities to support other modules, Hadoop Distributed File System (HDFS) that offers a distributed file-system with high-throughput access, Hadoop Yet Another Resource Negotiator (YARN) which is the framework responsible for managing the cluster's resources and schedule its tasks, and finally, Hadoop MapReduce that consists on Hadoop's implementation of MapReduce [78] for parallel processing of large scale data [76].

Apache Spark [79], is another a framework for performing general data analytics on distributed computing cluster like Hadoop. However, Spark handles most of its operations in memory which allows it to be much faster. In fact, it claims to run up to 100 times faster than Hadoop [79]. Spark started as a research project at the University of California at Berkeley's AMPLab [80] in 2010, and was later moved to Apache, in 2013. One year later, it was announced by AFS as a Top-Level project [81].

Spark can run on top of Hadoop's YARN, also taking advantage of its HDFS from where

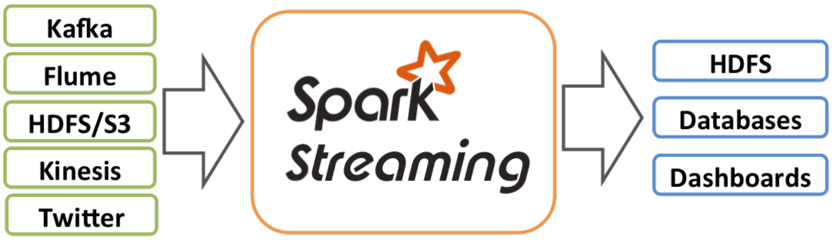


Figure 2.15: Spark Streaming diagram [79].

Spark can read data directly. It has support for multiple data sources such as Cassandra and HBase, and includes some interesting libraries such as MLib for machine learning, SQL for working with structured data, and GraphX for making graph-parallel computations [79].

One of the most relevant libraries is Spark Streaming, whose high level diagram is presented in figure 2.15. It is the Spark's attempt to provide stream processing for real time scenarios, providing high-level functions such as operating over a sliding window of data (windowing) and aggregating data. However, since it is done using micro-batch processing as shown in figure 2.16, data is not immediately processed when arrives, and thus provides at most near real time processing.

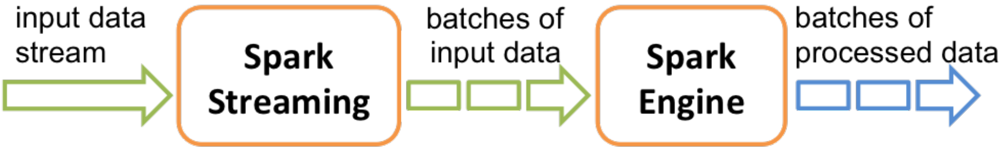


Figure 2.16: Spark Streaming batch processing [79].

In parallel with Spark development, there was another project that also achieved an Apache Top-Level status in 2014 [82], Apache Storm [83]. Storm is an open-source real time distributed computing framework, that basically does for real time what Hadoop does for batch processing, being able to process up to one million messages per second and per node [83]. As shown in figure 2.17, it is able to process events immediately when they arrive (stream processing), providing sub-second latency, and other interesting features such as fault tolerance, reliable data processing, and the fact that any programming language can be used with it, which is enabled by Apache Thrift [84].

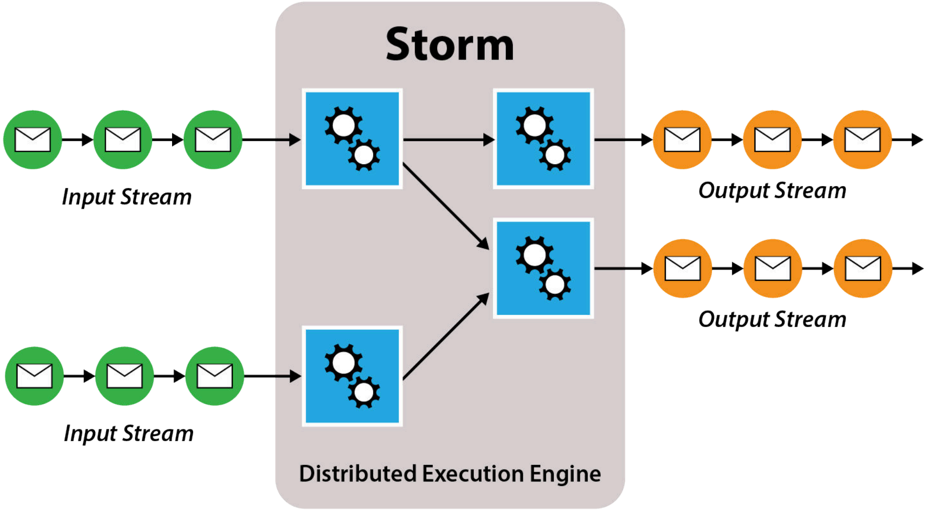


Figure 2.17: Apache Storm real time processing [85].

Moreover, Storm provides Trident which is an high-level micro-batching system similar to Spark Streaming, that provides higher level operations like windowing, aggregations or state management.

It is better suitable than Hadoop and Spark for applications that require minimal processing times, by using stream processing instead of batch processing which has its limitations on real time scenarios.

With the continuous growing of the Internet, the amount of data generated also grows brutally. This called for even better performance on big data processing. LinkedIn, which started using Apache Hadoop to process their large datasets, started noticing the batch processing limitations of Hadoop, and resolved to create their own proprietary framework for continuous processing of data, Apache Samza [86]. Samza was later open-sourced by LinkedIn in 2013 [87], and also became a Top-Level project in 2015 [88]. Like Apache Storm, Samza sought to build a lightweight framework for making stream processing, so it could have real time responses, in the order of sub-seconds. It was made to process feeds of messages, and like Storm, is able to process up to one million messages per second.

Additionally, by taking advantage of the Hadoop's YARN, it is able to provide fault tolerance, processor isolation, security, and resource management [86].

Another framework in this category is Apache Flink [89]. It started as Stratosphere project with its first publication in 2009 [90], and has evolved to an Apache Top-Level project in 2015 [91], and although not much adopted yet, it offers both batch and stream processing, with the ability to use the same algorithms in both modes. Flink implements its own memory management in order to reduce the garbage collection overhead, which allows it to provide an higher throughput than Storm, providing not only the same features such as fault tolerance and scalability (using YARN), but also libraries and Application Programming Interfaces (APIs) for graph processing, machine learning and CEP [89].

Finally, another Apache project that has very recently evolved to an Apache Top-Level project [92], and is probably the most promising one, is Apache Apex [93]. It was a commercial product of DataTorrent, the DataTorrent Real-Time Streaming (RTS), but is now an open source Apache project since 2015 [94].

Apache Apex is a unified big data stream and batch processing platform. It is able to process large scale data, with low latency and a high throughput. It also runs on top of Hadoop's YARN, inheriting most of its features, such as fault tolerance, processor isolation, security, and resources management. In addition, it provides a pipeline processing architecture, which allows it to be faster than the other platforms.

In summary, AFS has several open-source projects to address high performance and real time computing, being the most common the Apache Storm and Apache Spark for, respectively, stream processing and batch processing. However, Apache Apex is almost the merging of the two, providing both stream and batch processing but outperforming both, which will most likely make it a better choice for big data and real time applications in the future.

2.5.1.2 EBAY PULSAR

With the continuous growth of online shopping, consumer expectations have also increased and thus they require a fast a reliable service, that is able to respond to them in real time. In order to achieve this, Ebay presented an open-source real time analytics platform and stream processing framework, Pulsar [95]. Pulsar was designed to address real time applications such as fraud detection and business activity monitoring, being able to process hundreds of thousands of events per second and thus rapidly react to user activities. It is scalable, offers sub-second latency and a high availability, without downtime even on software upgrades [96].

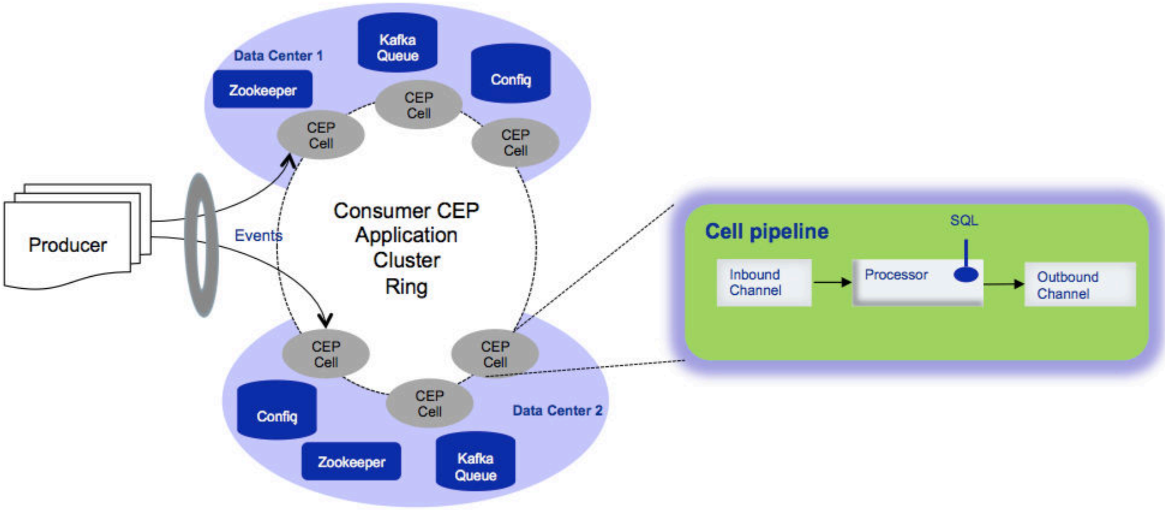


Figure 2.18: Pulsar deployment architecture [96].

Pulsar's processing logic is declared in Structured Query Language (SQL), and deployed in several nodes as shown in figure 2.18. Each node is called a CEP cell and may be located at different data centers. The architecture for each cell is formed by a main processor endowed with an inbound channel from where the events arrive to be processed, and an outbound channel for delivering the processed data.

To finish, Pulsar provides a high performance platform, with several qualities such as scalability, sub-second latency and flexibility, with the ability of being deployed in cloud infrastructures across data centers.

2.5.1.3 TWITTER HERON

Apache Storm was used by Twitter for several years for analysing the millions of events they get every day. In fact, it was Twitter who acquired and later open-sourced Storm under an apache license [97]. However, just like Ebay, with the increasing number of users and events every day, they needed better performance, which led them to introduce Heron [98].

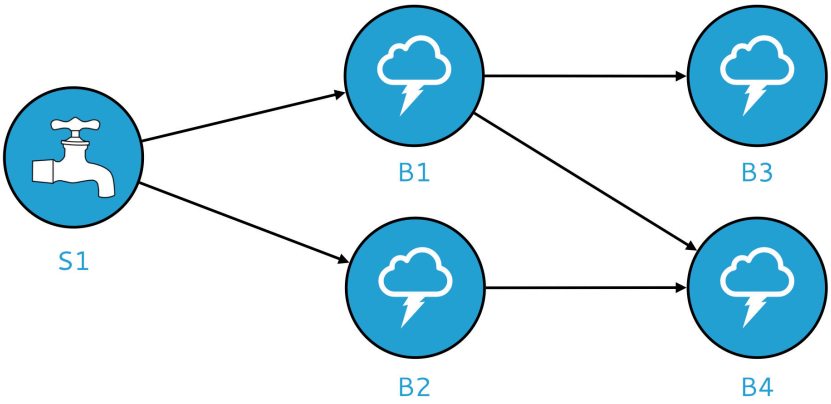


Figure 2.19: Heron topology example [98].

Heron, presented by Twitter in 2015, is a real-time analytics platform that offers architectural improvements and thus better performance over Storm. It is able to process billions of events per minute, with sub-second latency and has the advantage of having a fully compatible API with Storm. In fact, Heron was designed to be the successor of Apache Storm with the main goal of overcoming its performance and efficiency, by solving its limitations. To achieve this, besides providing a better management and isolation of processes, Heron implements additional mechanisms to improve its efficiency and reliability, such as a back-pressure mechanism which allows topologies to automatically adjust themselves in case of their components start lagging, thus avoiding tuple drops. A topology is, as in Storm, a directed acyclic graph used to process streams of data, which is divided in spouts and bolts as shown in the example of figure 2.19. Spouts are the entities responsible for spawning the tuples across bolts, which are responsible for processing them.

Compared with Storm, it can provide up to 14 times improvements in throughput and 10 times reductions in latency [99]. Additionally, Heron was recently open-sourced [100] under an Apache license, which makes it an even better choice for real time applications. However, it is still in a beta stage and thus not ready for production.

2.5.1.4 ENTERPRISE SOLUTIONS

Given the fact that both big data and IoT concepts have been evolving in the latest years, naturally many commercial products start emerging. One that was already referred in subsection 2.5.1.1, was DataTorrent RTS, which although being open-sourced in 2015, its development is kept active, adding features over Apache Apex, which is stated as its core.

Other solutions, mainly targeting streaming analytics, are Apama from Software AG, Infosphere Streams from IBM, Connected Streaming Analytics from Cisco and Azure Stream Analytics from Microsoft.

Regarding business rules and events, Operational Decision Management (ODM) from IBM and BusinessEvents from TIBCO are the most known commercial platforms.

However, these are all proprietary solutions and thus will not be discussed in this chapter, mostly because open-source solutions tend to have higher levels of adoption in the market as they generally provide software not only with more quality, but more secure, flexible, interoperable and customizable.

2.5.2 COMPLEX EVENT PROCESSING (CEP)

When processing large streams of events, some of them might be insignificant, either by being isolated either by the lack of other events or conditions. Therefore, only when a set of events along with certain conditions are matched, an action might be required. These are complex events, where a “Complex Event is an event that could only happen if lots of other events happened” [101].

CEP is the process of analysing large streams of information from multiple sources and, by detecting patterns and identifying meaningful complex events, quickly infer a conclusion from them and possibly generate an action. It can be very useful in a large variety of applications, by allowing to predict situations and thus avoiding issues or seizing opportunities.

While CEP emerged on financial industry, mainly for recognition of profitable transactions, it is now used in a lot more industries, essentially in real time and big data applications, such as trading and data analytics. Credit card companies, for instance, can use CEP for better managing frauds, by shutting off credit cards when a fraud pattern is detected, and before there are significant losses.

As an example where CEP can be applied, consider the following atomic events, along with figure 2.20 to illustrate them:

- A car is being driven on the road.
- The car has bald tires.
- The car’s driver is sending a text message from his phone.
- It is raining.

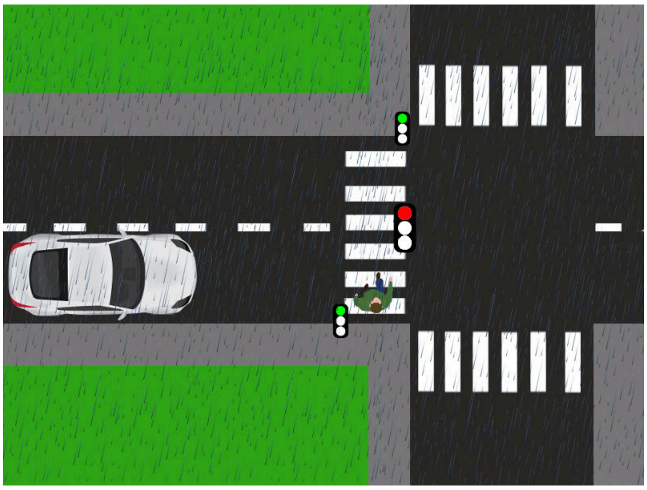


Figure 2.20: CEP Scenario

- Semaphore is showing red light for transit, and green light for the crosswalk.
- A person is walking into the crosswalk direction.

From this scenario, some complex events can be observed. First, from the fact that the car has bald tires along with that fact that it is raining, we can conclude the car would slip on a sudden braking. Then, since the person is walking in the crosswalk direction, and the semaphore is showing green light on it, we can deduct the person intends to cross the road. Finally, considering that the driver is sending a text message and thus is not paying complete attention to its driving, in addition to the fact that semaphore light for the transit is red, we conclude that he will end up making a sudden stop. Considering all this complex events we can predict the car will hit the person, something we can avoid if all this complex events could be processed in almost real time.

Of course this is a very simple example, a CEP system is able to analyse much more complex scenarios with huge amounts of data from countless sources. Essentially, a CEP system, or a CEP component of a system, provides a set of tools that allows for instance filtering incoming data, storing windows of event data, computing aggregations of these event data, and detect patterns or sequences on the acquired data. This means that any platform that offers these capabilities, is actually considered a CEP platform.

In order to use these tools, most CEP systems follow one of two approaches: rule-based or query-based. Query-based systems allow to consume event streams, taking advantage of the CEP tools, using a query language, typically similar to SQL. On the other hand, rule-based systems, which generally adopt Event Condition Action (ECA) rule semantics, divide a rule in three main components: event, condition and action. The event component defines or describes the event that triggers the rule, while the condition part is where a conditional expression is evaluated resulting in a boolean value. Finally, the action component, is the action to be triggered if the condition results in a true value.

With the rapid evolving of the Internet of Things, which brings huge quantities of data and events to the Internet, CEP has also gained a lot of attention, and there is already a large variety of both commercial and open-source platforms available that support it. Section 2.5.3 aims to present some of them.

2.5.3 CEP SYSTEMS

Several CEP enabled platforms have emerged over the years and there are now plenty of them available. In fact, most of the real time platforms discussed in subsection 2.5.1.1 support or provide libraries to support it. That is the case of Apache Flink with its native FlinkCEP library and Ebay's Pulsar through Esper integration, a software that will be introduced below. Regarding enterprise solutions, there are several providing Complex Event Processing such as DataTorrent RTS, Apama from Software AG, IBM Infosphere Streams, Microsoft Stream Insight, StreamBase from TIBCO and Oracle Complex Event Processing.

Moreover, there is also open-source software specifically designed to address Complex Event Processing, which will be referred in the subsections bellow.

2.5.3.1 ESPER

Esper [102], developed by EsperTech, is an open-source java-based software built to provide all the tools required for Complex Event Processing. With its first version launched in 2006, Esper is a processing engine capable of analysing real time streams of events, by performing continuous queries, and provides an high throughput and low latency.

Esper provides the CEP required tools and functions for dealing with large volumes of event data, such as aggregate functions, patterns matching, events windowing and events joining. All these functions are accessible through Event Processing Language (epl), its own Domain Specific Language (DSL), which is an SQL-like declarative language. As an example, consider the following query:

```
select feed, avg(cnt) as avgCnt, cnt as feedCnt
from TicksPerSecond.win:time(10 seconds)
group by feed
having cnt < avg(cnt) * 0.75
```

Snippet 1: Esper query example.

Considering that "TicksPerSecond" is a stream that contains the number of ticks per second in an example data feed, this simple query is able to look over all the records present in it in the last 10 seconds, and select the feeds whose ticks per second is below 75% of the average in those 10 seconds.

While Esper is able to run in single mode and in a single machine, it can also be integrated in other platforms such as Apache's Storm or Spark Streaming and thus achieve better performance. Since it is a Java application, it supports multi platforms. In addition, it is not limited to Java, and provides a C# version for running in the .NET framework.

EsperTech also provides an enterprise version of Esper, offering additional features, such as richer debuggers, REST-style Web services, multi-window Graphical User Interface (GUI) and push services.

2.5.3.2 DROOLS FUSION

Drools [103] is a Business Rules Management System (BRMS) solution provided by Red Hat. It is an open-source software built in Java, that uses rule-based approach and provides a collection of tools that enables the separation of logic and data in business processes, where data resides on domain objects and all the logic in rule files. A simple rule consists as follows:

```
rule "HelloWorld"
when
        message: Message (type == 'Hello')
then
        message.printMessage();
end
```

Snippet 2: Drools rule example.

This rule is triggered when drools receives an instance of the "message" class with its variable "type" set to "Hello". This triggers the rule's action which is the execution of the "printMessage()" method.

Furthermore, Drools also provides decision tables, allowing to separate the rule's data from the rule itself. The rule resides in a template with defined variables, while the values for that variables resides in a decision table. In addition, it is possible to have decision tables defined in a spreadsheet, allowing the use of known spreadsheet software such as Microsoft Excel, OpenOffice.org or Libreoffice.

This does not yet describe Drools as a CEP enabled platform. For that, the CEP high-level tools must be provided. That is where Drools Fusion gets into the picture. It is the module responsible for adding complex event processing capabilities into Drools, such as sliding windows and temporal operators.

2.5.3.3 SIDDHI AND WSO2 CEP

Siddhi is an open-source query-based CEP engine. It started as a research project at University of Moratuwa, Sri Lanka [104], and is now being improved by WSO2 Inc.

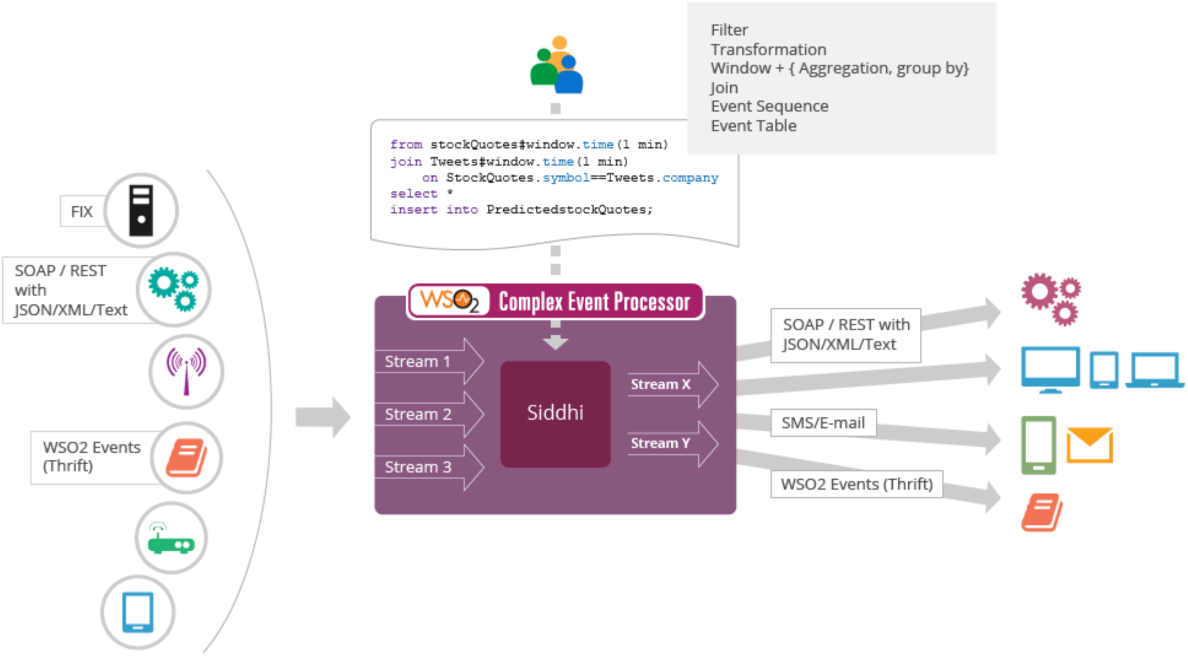
One of Siddhi's assets, is the fact it is really lightweight, being its minimum requirements a Pentium processor clocking at 800MHz or equivalent, with a minimum of 500 megabytes of memory. Like Esper, it can run in single mode or can be integrated with other platforms.

Currently it has support for multiple functions, like windowing, filtering, joining, aggregating and pattern/sequence detection. It also allows the definition of event tables, enabling Siddhi to work with stored events either in memory either in a Relational Database Management System (RDMS). All of this is done using the Siddhi Query Language (QL), which is similar to SQL.

As another feature, there is the fact that Siddhi can process events using other programming languages (JavaScript, R and Scala), by defining functions. In fact, even custom extensions can be written in Java by extending defined Siddhi interfaces, which makes it a highly extensible platform.

As stated before, Siddhi is now being improved by WSO2. Using Siddhi as base, they developed an even more complete CEP engine, WSO2 CEP [105]. It is also open-source and offers a low latency and lightweight platform for real time event detection and correlation.

Besides improving Siddhi, what WSO2 also did was enrich it with multiple tools. WSO2 CEP provides a complete Web GUI interface not only for managing and monitoring every

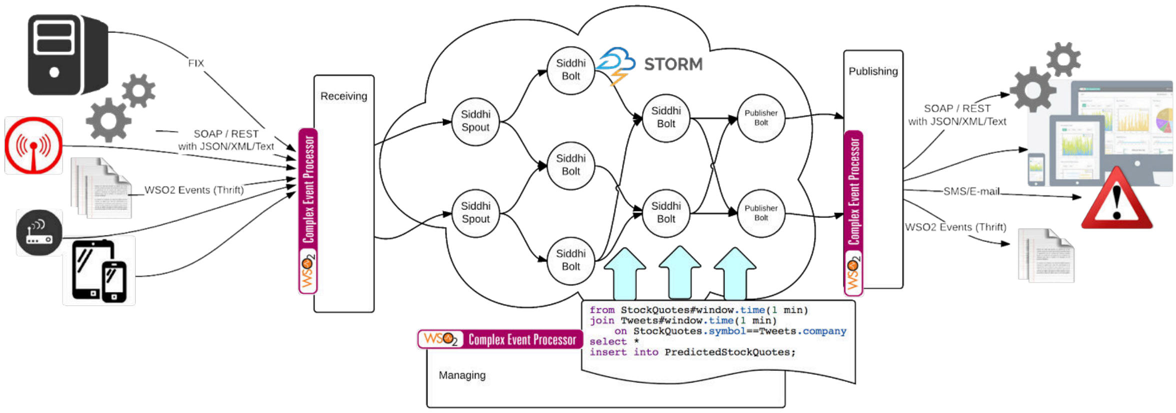


```
from stockQuotes#window.time(1 min)
join Tweets#window.time(1 min)
    on StockQuotes.symbol==Tweets.company
select *
insert into PredictedstockQuotes;
```

Figure 2.21: WSO2 CEP High Level Architecture [105].

aspect of the platform, but also to define all the Siddhi QL queries. It allows the creation of nice dashboards for data visualization, as well as collecting statistics, monitor and visualize operation metrics through the web interface.

Moreover, WSO2 CEP has support for multiple formats for data transport, namely JavaScript Object Notation (JSON), XML, Text or Map, and also supports multiple ways of receiving and publishing events (receivers and publishers) such as HTTP posts, Java Message Service (JMS), Short Message Service (SMS), Simple Object Access Protocol (SOAP), REST, e-mail and MQTT. Figure 2.21 shows its high level architecture diagram, where it is possible to observe on the left the multiple receivers and available formats, which are then transformed in event streams that can be queried by the Complex Event Processing, using Siddhi QL. After queries are performed, several new events may be generated and are thus sent to their different destinations through the publishers (on the right).



```
from StockQuotes#window.time(1 min)
join Tweets#window.time(1 min)
    on StockQuotes.symbol==Tweets.company
select *
insert into PredictedStockQuotes;
```

Figure 2.22: Distributed Mode - High Level Architecture [106].

Additionally, to achieve higher performance, it is also possible to run WSO2 CEP in a distributed manner, using Apache Storm [106], as shown in figure 2.22. Here, multiple instances of Siddhi engines can be spawned across a cluster of nodes, resulting a highly scalable system. The event processing is done in multiple nodes called Siddhi bolts, which are managed by the Siddhi spouts. Siddhi spouts are responsible for retrieving events from receivers and partitioning them through the multiple Siddhi bolts. Finally, every Siddhi bolt passes its events to the publisher bolts, which will then publish them using the publishers.

Chapter 3

Smart Environment solution

The goal of this chapter is to present a solution for providing intelligence to a building, through smart automation, in order to improve the overall efficiency and usability. In this process advantage is draw from IoT principles and deployment of an high-performance real-time platform for analysing and processing large streams of data with complex correlations. The implementation, targeted for a real world scenario integrated in the SmartLighting project, will be presented in the following chapter.

Following this short introduction, the chapter ensues, in section 3.1, with a description of the SmartLighting project, and its goal of developing an efficient lighting control system in IT2 building.

To design such a solution, there are several aspects that must be considered in order to ensure reliability. For that purpose, certain requirements must be laid out which identify the necessary capabilities or qualities that the system must have to satisfy its stakeholders. These will be addressed in the section 3.2, and some example use cases will be identified in section 3.3. Finally, in section 3.4, the system architecture is presented using a diagram, along with a description of each component and the interactions between each other.

3.1 SMARTLIGHTING PROJECT

3.1.1 OVERVIEW AND OBJECTIVES

The SmartLighting project's purpose, is to replace the current lighting infrastructure of IT2 building, which is currently based on CFL luminaires, with a new lighting infrastructure equipped with a network of LED luminaires and sensors, capable of collecting environmental data, such as temperature, luminance level, humidity and motion.

This new infrastructure will be backed up by an intelligent management system, able to process all environmental data and react to it in real-time. Furthermore, the system will also be able to analyse all the collected data and derive additional information, such as user behaviour patterns and statistics on the building's power usage.

Also, the system is designed to enable easy integration with other systems and applications, allowing inclusion of additional features in order to provide its occupants with increased comfort and enhanced usability.

An example of such extended features lies in the inclusion of external data sources such as network traffic, that enable the identification of building's occupants through their smartphones. This would then allow the system to get and/or learn the user's preferences, and apply them accordingly. In particular, a management solution integrating both lighting controls and HVAC systems, would be able to adjust the light intensity and the temperature of the occupant's office on arrival, or even turn his computer off when he leaves the building.

Another interesting source of data would be an external meteorological service. Weather conditions and predictions could enable the system to take preventive actions. As an example, the prediction of cold days could lead the system to change the HVAC profile to start heating the rooms earlier. Also, when a wind storm is predicted, the system could open all the blinds in order to avoid them being broken.

Additionally, interactions between the user and the system can be delivered, through either a mobile Application (APP) or a web interface. It has the potential to not only provide a user with a way of controlling his environment, but also to send him alerts or notifications which may be either from the controlled environment or other linked systems. As an example, the integration of a meeting scheduling system would enable the management platform to notify each of the participants before the event, remembering them the actual time and location of the room. Furthermore, such information may also be used to provide operational warnings to building managers, informing them that the room was used and might need cleaning services.

In fact, using the smartphone as an interface tool may enable a lot more features beyond the control of the surrounding environment, e.g based on location information. First, the smartphone itself can be used to validate the occupant's identity in the security system when he enters the building. Then, it is possible to locate the occupant in the building, eventually allowing other people to find him if he shares that information. This solution further enhances the presence detection systems inside the the rooms, complementing the sensor data from motion sensors. Furthermore, notifications may be triggered when the occupant arrives or leaves the building. For instance, an occupant can request the system to notify him when another occupant arrives.

3.1.2 STAKEHOLDERS

The people involved in any project are a big dependency for its success, either by having a direct influence in the project's execution, or simply by being the users of its outcome. It is crucial to identify the stakeholders and their expectations, in order to ensure that each of them is motivated and/or pleased with the project, thus achieving the better results.

In the SmartLighting project, the involved parts are essentially 4, namely:

- Developers Team
- IT2 occupants
- Building owners
- Building management team
- Building systems maintenance team

IT2 occupants and its owners are the primary stakeholders, the first by being the ones who will take advantage of the system's features and usability, and the latter by taking direct economic benefit from both the reduced energy consumption and the reduced maintenance costs. The building management team will benefit mostly from easier processes derived from the centralized control panels and dashboards that provide an overview of all systems. Furthermore, the building systems maintenance team will benefit from the notifications and alerts, that can also be sent to the management team, and ultimately allow faster detection of issues and enable a quicker response, sometimes solving problems even before they appear (e.g scheduled maintenance).

Finally, the developers team is fully committed and interested in creating a system that is able to satisfy all the other stakeholders and thus make the project a success.

3.1.3 USE CASES

In order to better understand the features offered by the management system, and its interactions with users, some example use cases are presented in this section. These examples cover the first contact the user has with the system and the system's reactions to the presence of the user inside the building, with and without direct user interaction.

User arrives at the building

The first situation to consider is when the user starts interacting with the system, which occurs the moment he arrives at the building. Figure 3.1 shows the possible interactions for this situation. Here, the first step is the user's identification, which can be done automatically at the moment the user's smartphone connects to the building's wireless network, if configured to automatically do so. The user may also be identified using his regular ID card, or alternatively, through the use of virtual cards, taking advantage of NFC technology, beacons or QR codes.

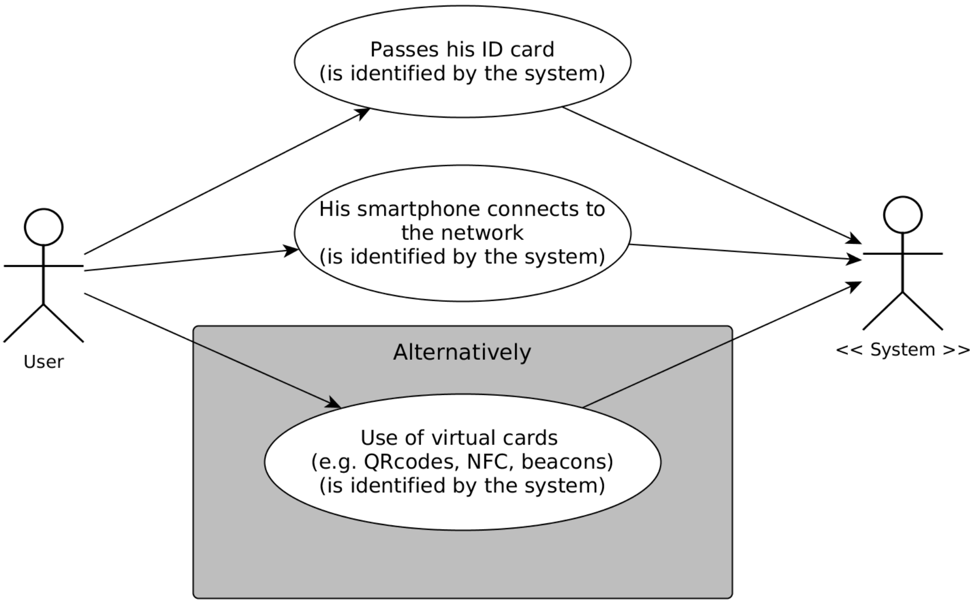

Passes his ID card
(is identified by the system)
His smartphone connects to
the network
(is identified by the system)
User
<< System >>
Alternatively
Use of virtual cards
(e.g. QRcodes, NFC, beacons)
(is identified by the system)

Figure 3.1: Use case diagram: User arrives at the building.

User enters the hallway

Following a typical scenario, the second use case occurs when the user enters the hallway (figure 3.2). There, a screen presents generic information related to the building such as current energy consumption, the number of persons inside and available meeting rooms. Additionally, the system may also present personalized information, configured by the user, that can go from a simple welcome message to notifications, eventually informing he is late for a meeting and where that meeting is taking place. Here, the system can also start sending notifications or messages to the user's smartphone via the mobile APP. For instance, the user can receive information of a more private nature, such as a message informing that another user would like to speak to him.

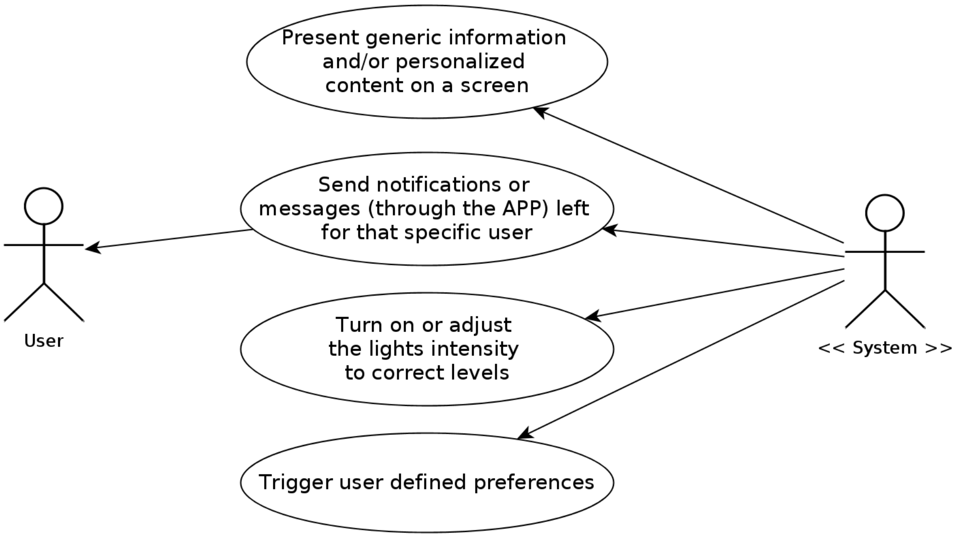

Present generic information and/or personalized content on a screen
Send notifications or messages (through the APP) left for that specific user
Turn on or adjust the lights intensity to correct levels
Trigger user defined preferences
User
<< System >>

Figure 3.2: Use case diagram: User enters the hallway.

Since the user has entered the building, the system may also start, if necessary, to adjust light intensity, mainly based on the background light levels or the detection of movement. At this stage, user defined preferences may be loaded and trigger changes to either the user surrounding space or to his destination, e.g his office. Here, the features available are strongly dependent on the integrated systems. As an example, the user can have configured the system to turn his personal desktop computer on when he arrives at building, or, in a more futuristic space, start preparing his coffee.

A similar use case can also be defined when the user leaves the building. Instead of turning on his settings, the system would perform actions, such as turning the personal computer to an off or standby state, switching off his desk lamp or even closing the automated blind in his office. Additionally, his state could be set to “Away” in the internal building user directory, which could be visible to others. The main difference in this use case is the detection of the user’s departure which besides the previously mentioned methods for login/logout, would also include detection of absence through, e.g. long periods of inactivity.

User moves through the building

The third use case considers the movement of the user through the building, e.g. when he is on his way to his office (figure 3.3). As he walks, the system continuously adjusts the lighting in the space surrounding him, providing the necessary brightness whilst trying to reduce energy consumption. Here, a feature typically known as “corridor function” is applied. Based on the user location and the movement direction, the intensity of the nearest luminaires is increased while those further away from him are kept at a lower level. As he moves, the luminaires in his path start to brighten and those behind him begin to dim down, and eventually shutting off after a given period of time with no activity.

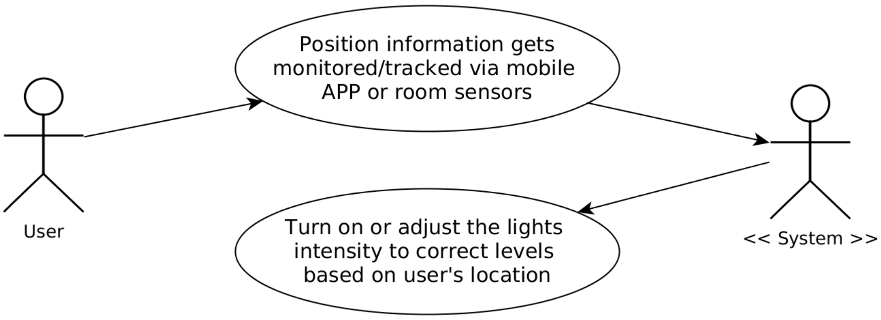

Position information gets monitored/tracked via mobile APP or room sensors
Turn on or adjust the lights intensity to correct levels based on user's location
User
<< System >>

Figure 3.3: Use case diagram: User moves through the building.

The dynamic process of adjusting the luminaire's output requires a decent method for detecting the position of the user along with a fast response time of the automation and management platform. As users are quite susceptible to variations in light levels, a proper tuning of the brightening and dimming times and final brightness values is required, and may be improved by learning typical usage conditions such as common destination, average speed, etc.

User arrives at his desk

Finally, the user enters in his workroom and sits at his desk (figure 3.4). If he is the first getting into the workroom, the system will start by lighting the room. Next, more specific preferences are applied to the surrounding environment. These may include opening the automated blinds, setting the HVAC level and adjusting the brightness level of his desk lamp, or the luminaires closer to him, to a level that the user considers more comfortable.

Moreover, the user can keep interacting with the system, either using the web interface or the mobile APP. He can adjust all his preferences and take control of all the building services surrounding him or receive notifications, such as the missing paper in his workroom printer.

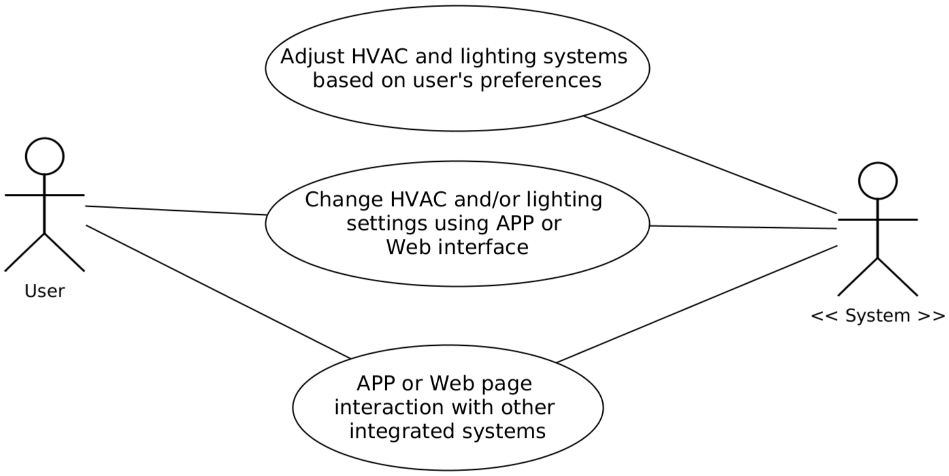

Adjust HVAC and lighting systems based on user's preferences
Change HVAC and/or lighting settings using APP or Web interface
APP or Web page interaction with other integrated systems
User
<< System >>

Figure 3.4: Use case diagram: User arrives at his desk.

All this information can be filtered, for better usability, from the user position information. As an example, a regular user should not be able to change the conditions of a room he is not currently on.

It is also important to note that, a user might not be alone at his workplace, and thus the system must consider preferences from multiple users. Here, the system shall consider priorities based on user's access level, and may only partially apply each user's preferences or combine them into an intermediate level.

To conclude, it is important to emphasize the low number of situations where an interaction from the user is requested. Mostly, these consist of changing his preferences or interacting with third-party applications, such as the example meeting room scheduling system. This reduced interaction calls for a dynamic and "smart" decision making process from the management system, which must control the automation systems, often based on complex and real-time situations.

3.2 REQUIREMENTS

As stated, the proposed solution is being designed for a real-world scenario, integrated in the SmartLighting project which was elaborated in section 3.1. Therefore, the requirements in this section must also fulfil the project's objectives defined in section 3.1.1.

Fast Responsiveness

One of the most important requirements to be satisfied in any BAS is fast responsiveness. Even if not-critical for some functionalities, many others require sub-second delays. For instance, if the system detects that the room temperature is below a certain threshold, and takes some seconds to start the heating system, occupants will not even notice the delay as the process itself may take several minutes. But if someone enters a room and the system takes more than a second to turn on the lights, the occupant will certainly notice the delay, and surely be unsatisfied with it.

Flexibility and Scalability

Other important requirements are the levels of flexibility and scalability. The system must not only support easy integration of other applications, so that new functionalities can be attached, but also effortlessly accommodate its growth, i.e. allow to easily add new devices, either sensors or actuators, without difficult configurations or cumbersome processes. Furthermore, as a consequence of adding a large number of new devices, scalability should also be supported at the servers side. Independent of the number of added computing nodes the highest performance levels and real-time responsiveness must be maintained.

Failure Handling

A secondary requirement is failure handling. The system should be able to recover from faults and, even when the whole management platform fails, minimum functionalities must be maintained by some of the infrastructure systems. As an example, in case of management platform failure the lighting system should default to a locally automated mode, or at least a manual mode.

Basic Functionalities

A system that satisfies these requirements is, in fact, a good base for developing a BAS solution. The functionalities and features built on top of it, are what creates an intelligent and automated environment. Typically, a BAS must, at least, perform an automated control of the lighting and HVAC systems, an therefore should also be addressed by this solution.

To make an efficient lighting control in any space, there are three main factors to take into consideration: when, where and how much light is needed. The amount of light needed can be easily obtained from the information provided by illuminance sensors. This is then processed in order to calculate the adequate percentage of light output to be applied on luminaires. It is the question of when and where to apply it, that tends to be more complex to resolve. Although simple solutions can be achieved with simple motion detection information to infer the presence of people, it is difficult to get precise data with low power sensors and at a low deployment cost. Typically infrared motion sensors are used, which are the cheapest, but unless combined with other more expensive types of sensors, such as microwave or ultrasonic sensors, they tend to generate a lot of false positive detections, thus making them unreliable. However, people nowadays are most of the time "connected" to the different wireless access points of a building, either with their laptops or smartphones, or through the new trend of wearables such as smartwatches. Information from the network can be fetched and combined with access points positions, in order to detect the presence of building occupants. Similarly, BLE beacons can also be used indoor with the same goal, where beacons associated with rooms, can periodically send signals that occupant's devices can receive and then report back with their estimated location to a server.

Regarding the control of HVAC, it is typically easier to achieve due to the inherent slower dynamic of temperature changes. It is only dependent on the information retrieved from sensors, such as room temperature, humidity and sometimes gas sensors. Yet, it can also be improved with the use of information gathered for the lighting system, because the number of people in a room also has influence in the temperature and air quality.

Data Correlation

From the functionalities described so far, we can observe that correlation of information from multiple sources is needed in order to make the different components of the system more accurate. This is, in fact, another important requirement for the proposed solution and, actually a differentiating factor from others currently available in the market.

User Control

In line with the SmartLighting project, the system must also provide means for the building's occupants to manually control the equipment around them. However, to ensure only authorised actions are performed, there needs to be in place a system component responsible for managing users and access control.

User-friendly Interfaces

The solution under discussion in this chapter has the intent to provide a smart and comfortable environment to the building's occupants, and reduce the energy costs by efficiently managing components either disabling them or putting them in a minimum state, when not needed. However, and as a final requirement, the management and configuration platform must also be user-friendly for the management team, since they usually are not endowed with programming skills. Thus, they should be able to configure every automation rule and manage the building's virtual structure, i.e. the division of the building in floors, rooms and areas in a human friendly and intuitive fashion.

3.3 USER DRIVEN CASES

Albeit there are several stakeholders in any BAS, there are mainly two actors for which use cases can be specified, the building's manager and its occupants/users. Although the solution presented in this chapter aims to address all the requirements established in the section above, more focus will be given to requirements like scalability and extensibility rather than functionalities. In fact, the key feature of this solution is to provide an easy way for adding new system behaviour rules, which can be used to extend functionalities even without additional devices. By letting the building manager dynamically add new rules, a custom behaviour can be implemented from already existing sensor information from multiple areas. For instance, information from sensors in neighbouring areas could be used for adjusting the on, off and delay times of luminaires. Therefore, there are countless use cases regarding interaction of the management system with the automated environment. Thus, those defined for SmartLighting project, in section 3.1.3, are considered as minimum requirement.

Building occupant

Regarding to the interaction of the user with the system, either through a mobile or a web application, figure 3.5 shows the primary use cases to be approached in this solution where, besides the common operations of login, logout and profile edit, we have five main use cases, each doing the following:

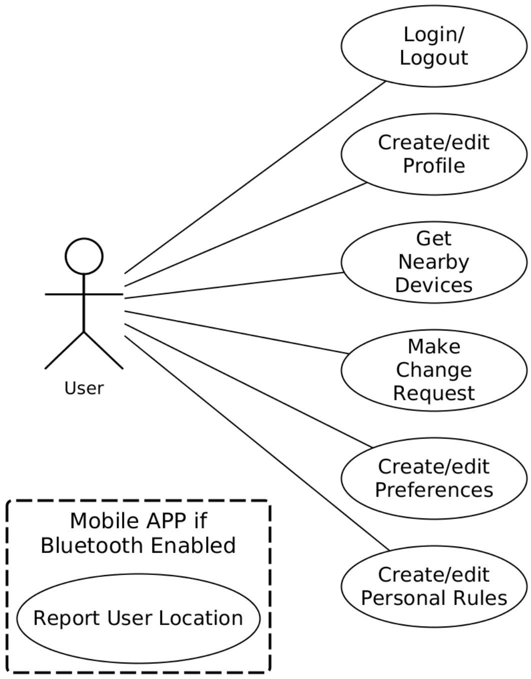


Figure 3.5: Use case diagram: interaction with building occupants.

- Get nearby devices: request the system a list of devices near the user. Here, in an initial design, the user will get all the devices from the room he is in;
- Make change request: this is the process of requesting a change of state in a device. It could be for instance a light, a group of lights or the air conditioner unit. The system can deny the change, apply it fully or partially depending on the user's access rights;
- Create/edit preferences: create or edit preferences device's pre-sets, so they can be applied automatically. Again, user's access rights might not allow it;
- Create/edit personal rules: this is where the user can create his personal automation rules. The platform should allow the integration of subsystems in order to provide this kind of functionalities to the user. As an example, the user could add a rule to turn his computer on when he enters the building, and turn it off when he leaves;
- Report user location: this is an automatic operation done only on the mobile APP. If the Bluetooth is on, whenever the device gets a signal from a beacon, it sends a message to the system with the beacon ID and the user ID, allowing the system to know the user's location.

Building manager

Building manager use cases, which can be found in figure 3.6, address the operations related with the configuration of the system and the creation of rules. A description of the use cases follows:

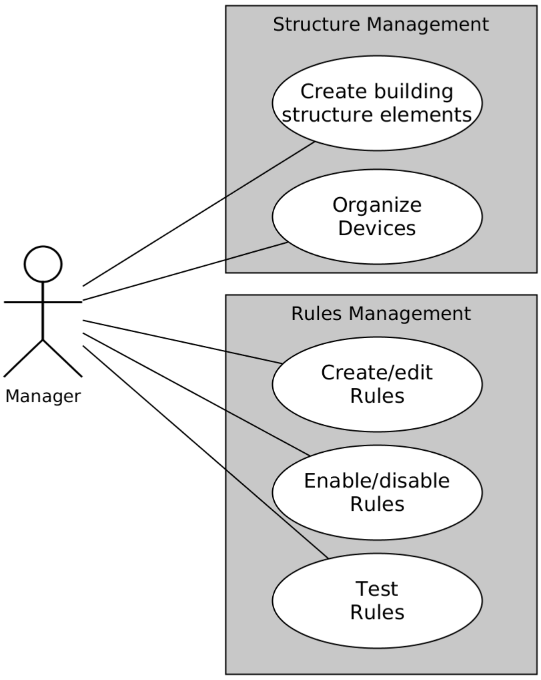


Figure 3.6: Use case diagram: building manager interaction

- Create building structure elements: this is where the manager creates a virtual representation of the building, where he can specify the buildings, floors, rooms and areas inside each room;
- Organize devices: distribute the devices (sensors and actuators) by the areas of each room;
- Create/edit rules: this is the process of creating automation rules to be applied in the building;
- Enable/disable rules: allow the manager to enable or disable rules without deleting them;
- Test rules: this is a special feature in which the manager can first test a rule in a simulation environment before applying it in the building.

3.4 SMART ENVIRONMENT SOLUTION ARCHITECTURE

Having in mind the SmartLighting project's requirements, the proposed architecture fits with the intent of using IoT and CEP principles in order to achieve not only an energy efficient solution, but also a smart, automated and comfortable workplace. This section aims to present the architecture, by specifying the different components that form it and describe their goals and interactions.

3.4.1 OVERVIEW

The architecture diagram presented in figure 3.7 shows the four different layers in which this solution is divided.

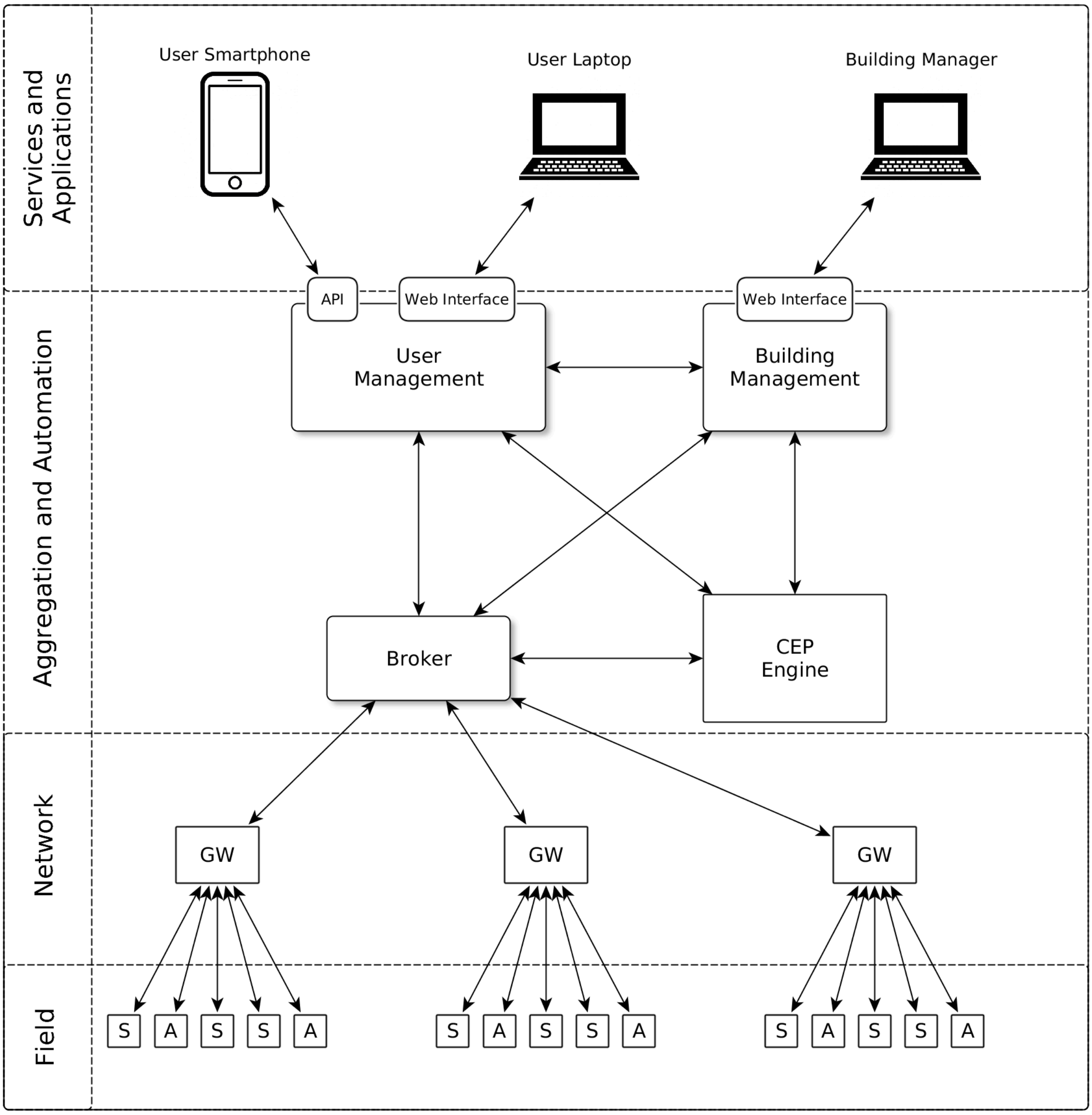


Figure 3.7: Smart Environment: architecture diagram

Services and Applications Layer

On top, the services and applications layer, is where both the building user and the manager interact with the platform. The manager is able to administrate all the building's structure and automation rules through a web application. Here, a user-friendly interface should be provided along with intuitive dashboards for easy monitoring. Regarding the typical user, the building occupant, besides a web application that allows him to configure and manage its personal preferences, also a mobile APP can be provided to change settings and states of nearby devices. Additionally it may be used to provide his location to the system, an information that can help detect presence in rooms but also support new features such as "finding friends". This layer also provides interfaces for other services such as additional dashboards and analytical tools for alarmist, management and forensic analysis.

Field and Network Layers

Devices, that can sense and/or actuate, stay at the field layer, where all the interactions with the environment are made. Sensors are responsible for collecting data and detecting changes, and then push that information to the automation layer. The later may respond with actions to be executed in order to change the monitored environment conditions through the actuators. The devices communicate with the automation layer through gateways, which are found at the network layer, and shall be responsible for inter-connecting the WSANs and the aggregation and automation layer.

In order to ensure devices are trustworthy, and that only they are able to connect to the platform, both the gateway and devices can implement a challenge-response authentication. This enables the gateway to discover new devices automatically, and to only process authorized devices.

Additionally, in this solution the network layer is also considered to be the basic automation layer. That is because gateways should also be endowed with lightweight processing engines, which is a very important feature in failure handling. If the connection to the complex automation layer fails, gateways must become responsible for providing minimum functionalities. Considering this, gateways shall also be configured with basic automation rules, since they are the closest processing engines to the WSANs and thus might provide faster reactions. For instance, in corridors, the lights should turn on based only on motion detection and illuminance readings from local sensors. Thus, if a rule for this is configured at the gateway processing engine, faster responsiveness might be provided and even on a platform failure, the lighting infrastructure would keep working without users noticing.

Aggregation and Automation layer

Regarding the prime part of the whole architecture, at the aggregation and automation layer, it is divided in four main components that interact with each other to provide the automation and intelligence to all the different building's systems. The broker component, refers to a message broker responsible for receiving, routing and delivering all the exchanged information between the WSANs and the rest of the components. The use of a message broker has several advantages. First, it becomes the central place from where all the information passes and thus all other components can get information directly from it. Then, it avoids additional burdens on other components regarding to the reception/delivery of messages from/to multiple sources/destinations. In addition, they are an extensibility enabler, by easily allowing new systems to connect to it for data retrieval. For instance, a data analytics component could be easily added for statistics or machine learning purposes.

Still in the aggregation and automation layer, one of the most important components is the CEP engine. The CEP engine must be a high performance and real-time platform with support for CEP such as the ones discussed in chapter 2. This platform is responsible for processing every event generated by sensors, and using complex event processing, infer actions in real time. Here, an action is not necessarily an action on the environment using actuators. It can also be an alert that can be pushed to another component or directly sent to the building manager.

Although the CEP engine can run by itself, it is governed by rules that are usually too complex for building managers. That is where the building management component gets into the picture. Its purpose is to provide a user-friendly interface for the building manager to create, edit or delete rules. Then, based on these rules it generates the complex code that forms a rule to be applied on the CEP engine.

Additionally, it is also in this component that the user creates the virtual representation of the building, and distributes the devices by areas in rooms. Thus, this component is also responsible for providing the information about a device's location, i.e. to what building, floor, room and area it belongs, and configure the devices accordingly.

Finally, the user management is the platform responsible for managing all the system user's related functionalities. It provides both the web and the API interfaces through which the users interact. On the internal side, it is responsible for sending the users locations to the broker (so it can also be processed in the CEP engine), control field devices also through the broker, and apply user defined rules either using the building management platform, or directly in the CEP engine.

3.4.2 COMPARISON WITH IOT REFERENCE MODEL

The given architecture is also able to fit in the IoT reference model, shown in figure 3.8. The IoT reference model was introduced by Cisco in order to simplify and standardize IoT systems and their connections. It provides a seven-levels architecture specifying the tasks performed on each one of them, in order to achieve an high extensibility, scalability, simplicity and supportability [107].

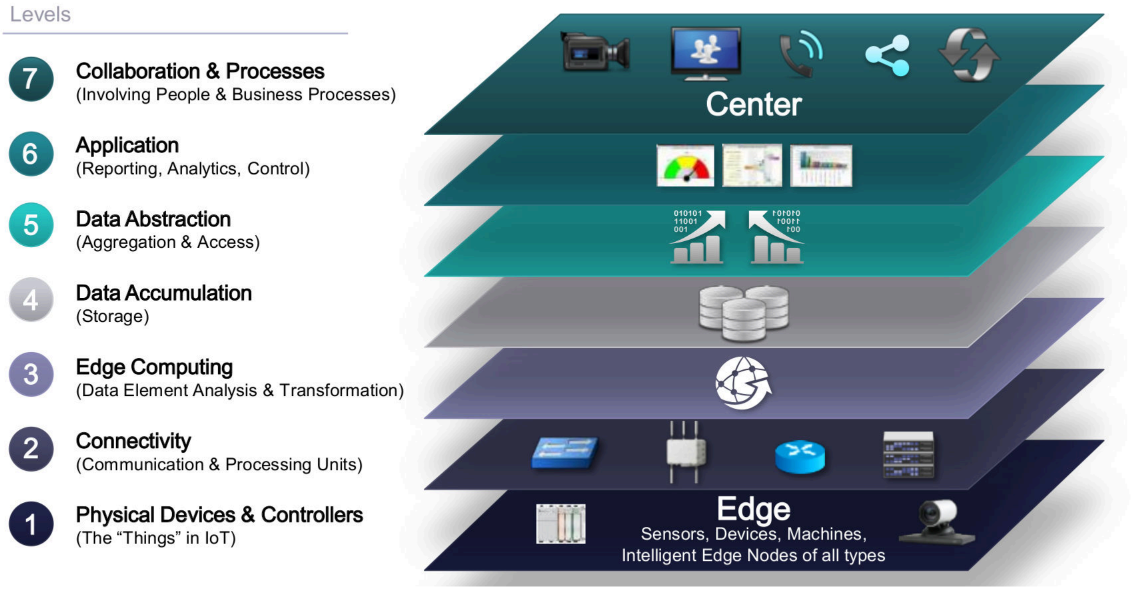

Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types

Figure 3.8: IoT World Forum Reference Model [107].

The first level, Physical Devices and Controllers, is where the IoT devices (or "Things") are found, generating events and exchanging information with the upper levels, which is analogous to the Field layer specified in the presented architecture. Similarly, the Connection level of the IoT reference model can also be directly associated to the Network layer of the present model, where switching and routing of events is performed in a reliable manner.

Regarding our core layer, Aggregation and Automation, it originally only addresses the fifth level of the IoT model, Data Abstraction. This is where data is combined, transformed and selected, by the CEP engine along with the building and user management components. The third and fourth levels can also be included in this solution with the use of a more complex platform replacing the broker. In fact, the next chapter presents the implementation of the proposed solution where the broker used is taken from an existing platform, where Edge Computing and Data Accumulation is done.

Finally, Application and Collaboration & Processes levels of the IoT Reference Model can also be naturally associated to the Services and Applications layer of the presented architecture, allowing the integration of new systems and services and thus the interaction with people and business.

Summary

This section has presented a model that addresses all the specified requirements and use cases, with a simple architecture capable of creating an abstraction layer between all the complexity and both building occupants and the building's manager. Additionally, the architecture fits in the reference model for IoT provided by Cisco, which defines a standard way for development of systems for the IoT by describing each of the seven levels that form it, along with the interactions between them.

The CEP engine is responsible for achieving the fast responsiveness of the system, while providing means for correlating information in a complex manner, which is also an important defined requirement.

Furthermore, to address the failure handling requirement, processing engines may be used in the gateways, at the network layer, so the system is able to pursue its main function even when the automation platform or the connection to it fails. This is allowed by using gateways as local engines with the basic and time-critical functionalities, leaving the main platform with the responsibility to perform more complex automation. Alternatively, such behaviour may be triggered only when a failure is detected. Nonetheless, each device should also be endowed with an automatic mode, for situations considering network failures with the gateway.

The specified use cases for the building occupant user, in section 3.3, are carried by the user management component of the aggregation and automation layer, where the user shall be able to manage its preferences and rules, and manually control a device. The latter is done by requesting the building management component (using its API) to put the device in manual mode, and then directly send actions to the devices using the broker. Regarding the building manager use cases specified in the same section, they are all approached by the building management component which provides a simple web interface for that purpose. There multiple tools exist in order to provide all the functionalities and features such as the lighting and HVAC automation.

Furthermore, with the use of a central broker, and the provision of APIs for interaction with the user and building management platforms, the presented solution promotes the integration of new platforms that can access the sensors data and provide new features, addressing the extensibility and scalability requirements.

CHAPTER 4

Smart Environment Implementation

In chapter 3, the high-level architecture of a possible solution was presented, addressing all the requirements for endowing any building with intelligent and efficient capabilities in order to provide an automated environment and reduce energy wastage.

This chapter's objective is to describe an implementation following this model, explaining the steps taken and the reasoning behind each decision For this purpose diagrams and code snippets will be used when pertinent. It starts by defining the objectives to be accomplished, in section 4.1, and then referring the platforms and protocols adopted, in section 4.2. After that, a description of how access to devices is done is presented in section 4.3. The diagram of the whole implementation is shown in section 4.4, followed by an introduction to the main concepts of WSO2 CEP in section 4.5. A detailed explanation of the core component of the implementation, the building management, is given in section 4.6.

As previously stated, this implementation is intended to be applied in a real world scenario, in line with the SmartLighting project. Within the same project, work from two other dissertations from the Integrated Circuits group of IT are being developed, focusing the creation of devices, particularly luminaires, equipped with sensors and actuators. A device simulator was also developed to verify if the implementation behaves correctly. It is described in section 4.7. Finally, as the physical devices connect to the platform using Bluetooth Low Energy, a gateway was implemented, as described in section 4.8.

4.1 OBJECTIVES

Despite the fact that implementation of the whole solution, as presented in chapter 3, would be ideal to perform in depth validation, it would be unrealistic when considering the amount of work it would require, and the available time. Thus, considering that this implementation is targeted for a real scenario, at the IT2 building, a set of objectives were defined in order to ensure the most relevant functionalities were present.

Considering this, the user management component defined in the architecture of the presented solution was left for future implementation, waiving features related to user interaction with the system. In contrast, all the use cases defined for the manager interaction with the system were implemented, in order to allow management of all the rules necessary to create a smart and automated environment. The use of automation rules at the gateways was also discarded from this dissertation at the initial implementation, despite this fact, the means to do so are provided.

Extensibility, which was one of the most important requirements defined, was approached in this implementation not only by allowing the integration of other platforms and services, but also by enabling the addition of new tools and functions to the building management platform. This allows to increase the number of options for customizing rules. Nonetheless, basic functionality should be available from scratch, allowing at least the automation of the lighting infrastructure.

4.2 ADOPTED TECHNOLOGIES

As shown in the architecture model adopted by this implementation, components like a CEP engine and a message broker are both needed in order to provide complex and high performance processing of events. Therefore, this section aims to state the chosen technologies as well as explaining the reasons behind the decision process.

One of the main goals of the SmartLighting project is to take advantage of the Internet of Things concept, and use low power constrained devices equipped with sensors and actuators. Hence, the use of protocols adapted to this kind of devices is necessary. The most commonly used protocols in the IoT were presented in chapter 2, where CoAP and MQTT are seen as the most suitable for constrained devices.

Although they are both proven to be lightweight and offer high performance, MQTT was chosen because of its publish/subscribe model, which brings more advantages to this kind of applications. With this model, devices can subscribe multiple topics, allowing them not only to subscribe their own topic, and receive events targeted individually at them, but also to subscribe to more generic topics, enabling them to receive events sent to a group, such as their area, room or floor. Additionally, this implementation makes use an existent MQTT broker on the Smart Cloud of Things (SCoT) platform, which is the work of another dissertation, whose purpose is to integrate different data sources in one common platform [108].

Regarding the choice of a CEP engine, while it was shown in chapter 2 that one can be built by combining a CEP system and a real-time platform, solutions as Drools (with Drools Fusion) and WSO2 CEP already provide that combination. The later was chosen for three main reasons. First, it is a full open-source solution that provides all of its features out of the box, while the best of Drools is provided in an enterprise version by Red Hat, the JBoss BRMS [109]. Second, Siddhi QL is very similar to SQL, which is a powerful and familiar language. Finally, WSO2 CEP provide several ways for both receiving and publishing data, including MQTT, SMS and e-mail, with these last two being interesting for notifications and alerts.

Finally, to be referred that objects definitions from another dissertation's work was also implemented, creating a standardized way for both reading and writing information from/to devices. This will be better explained later, along with the rest of the implementation.

4.3 ACCESS TO DEVICES

An intelligent and automated environment can only be achievable with the use of devices able to detect changes in the environment (sensors) and trigger or control mechanisms that are able to make changes in that same environment. Currently, there is a plethora of sensors and actuators capable of measuring/acting-upon a diversity of variables. Thus, it should be expected for new devices to be added, moved or removed in the future. With that in mind, standard ways of accessing the devices as well as message formats should be defined in order to support extensibility of the system.

4.3.1 OBJECTS DEFINITION

In line with another dissertation's work, an object representation was implemented for this platform, following the guidelines of the IP Smart Objects Alliance. This ensures a standard way for reading and writing device's properties. Objects are abstract representations of an actuator or sensor type, containing a generic description and some additional information, and each unique object is identified by the object ID. A device can have multiple objects, which represent the different sensing and actuating capabilities it provides. For instance, a device can have objects representing a luminaire actuator, a temperature sensor and a motion sensor.

Inside each object, the information is organized in resources. These represent either values that can be read and/or write, e.g. a sensor configuration parameter or an actuator's current state, or actions that can be triggered remotely, such as the request to dim a luminaire's output brightness. Each resource of a given type is also identified by an ID number. Additionally, since a device can contain multiple objects of the same type, as well as objects can contain multiple resources of the same type, e.g. for creating arrays of values, both objects and resources contain an instantiation number for being easily accessed. Figure 4.1 helps understand this representation.

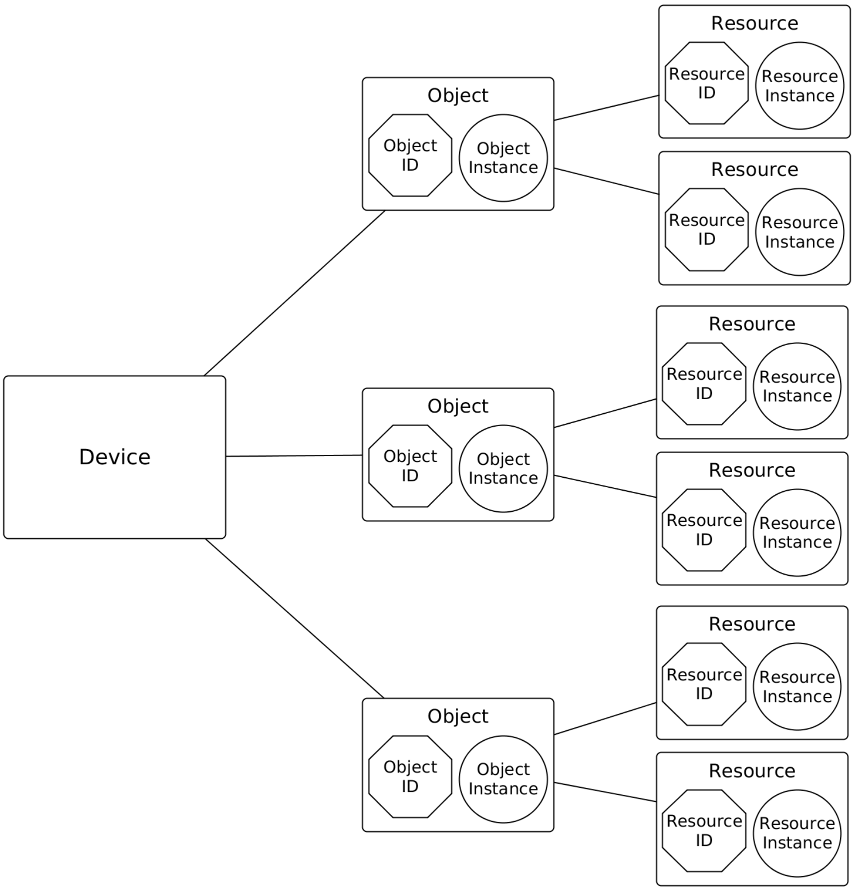


Figure 4.1: Device objects representation

The access to a device's property is made, hence, through a path containing an object ID, its instantiation number, and similarly, a resource ID with its instantiation number, in the following manner:

.../Object_ID/Object_Instance/Resource_ID/Resource_Instance

As an example, a luminaire with a resource to turn it on or off, and another one for controlling its dimming level, would have the representation shown in figure 4.2. This way,

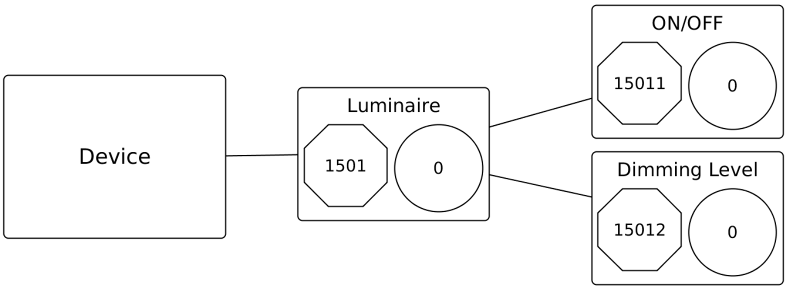


Figure 4.2: Luminaire object representation example

the access to the luminaire's dimming level, either for reading or changing it, can be easily done using, for instance, a Uniform Resource Identifier (URI) that contains both the IDs and the instance numbers of the object and the resource, as follows:

$$\boxed{\text{.../1501/0/15012/0}}$$

Relating this structure to the MQTT protocol, where every message is sent by publishing it in a specific topic of a broker, and received by every client who has subscribed to the same topic, it makes sense to include the URI within this topic.

Additionally, MQTT has support for wildcards, which are specific characters that can be used for replacing one or more levels of the topic's multi-level hierarchy. This implies subscribing simultaneously to more than one topic. For this purpose MQTT currently supports two wildcards: *+* and *#*. *#*, which must be used as the final character of a subscription, allows the access to all remaining levels of the hierarchy. For example, the following topic may be used for subscribing to all events of a luminaire in a device. Note that 1501 is the ID associated to a luminaire, as in the examples presented before.

$$\boxed{\text{.../1501/\#}}$$

Similarly, the *+* wildcard can be used in a single level for matching all the topics where the rest of the hierarchy exists, i.e the topic excluding the levels where the *+* is at. For instance, to subscribe all the events of dimming levels in a luminaire, the following topic could be used:

$$\boxed{\text{.../1501/+/15012/+}}$$

As a drawback, MQTT does not support wildcards on publishing. Yet, this was resolved by making devices subscribe topics with a new wildcard, which is a solution specifically designed to address this problem. Considering the example presented before in figure 4.2, if the device subscribes the following topics, the term “all” can be used for publishing, exactly in the same manner as the + wildcard is used for subscribing.

```
.../1501/0/15012/0
.../1501/0/15012/all
.../1501/0/all/all
.../1501/all/all/all
.../all/all/all/all
etc.
```

4.3.2 MESSAGE FORMAT

The definition of the message format is an essential part of any platform. Since most of the messages exchanged in the whole system are between constrained devices and the automation platform, JSON was chosen as the data format, as it is compact and typically faster.

```
{
    "event": {
        "metaData": {
            "attribute_1":VALUE,
            "attribute_2":VALUE,
            ...
        },
        "correlationData": {
            "attribute_1":VALUE,
            "attribute_2":VALUE,
            ...
        },
        "payloadData": {
            "attribute_1":VALUE,
            "attribute_2":VALUE,
            ...
        }
    }
}
```

Snippet 3: WSO2 CEP JSON event format.

Considering the choice of using WSO2 CEP as the engine for this implementation, and in order to enable an easier integration, its event format was adopted in this implementation. Therefore, an event in WSO2 CEP is defined as in snippet 3, where the event’s attributes may be divided in three main logical sections: Payload Data, Correlation Data and Meta Data. Payload Data refers to the most important data to transport in the event, corresponding to the actual values that need to be processed. The Correlation Data is intended to transport information that allows correlating events with other events. Finally, the Meta Data is where other attributes that describe the event itself may be included.

This implementation considers two types of messages exchanged: events and operations. An event is a message that contains a resource value. The event can be sent to a device for writing that value into a resource, or as a device's report of a resource value change. Either way, information about the object and resource must be included in the event message. An example of an event generated by a device containing a motion sensor, to report a motion detection is presented in snippet 4. Here, only Payload Data is considered because all the information to be transported is mandatory. This includes information about device, object and resource, along with its value.

```
{
    "event": {
        "payloadData": {
            "device" "motion_1",
            "object": 3302,
            "object_instance" 0,
            "resource": 5500,
            "resource_instance": 0,
            "value": 1
        }
    }
}
```

Snippet 4: Example message format for an event generated by a device.

Regarding the events sent to devices for changing resources values, an example is shown in snippet 5 where an event for turning on a luminaire is presented. In the attributes for the object, resource, and their instances, the value -1 may be used for targeting all the available options for that attribute. For instance, in the example of snippet 5, the attributes "object_instance" and "resource_instance" are set with the value -1, thus the event must change the value of all resource instances containing the ID 15011, and belongs to any object instance that has the ID 1501.

Moreover, considering the solution stated before for publishing messages with wildcards in MQTT, where the objects and resources information is provided in the topic, the same event can be simplified to the one in snippet 6, if published in the following topic:

.../1501/all/15011/all

```
{
    "event": {
        "metaData": {
            "operation":"set"
        },
        "correlationData": {
            "object": 1501,
            "object_instance" -1,
            "resource": 15011,
            "resource_instance": -1,
        }
        "payloadData": {
            "value": 1
        }
    }
}
```

Snippet 5: Example message format for an event sent to a device.

```
{
    "event": {
        "metaData": {
            "operation":"set"
        },
        "payloadData": {
            "value": 15
        }
    }
}
```

Snippet 6: Example simplified message format for an event sent to a device.

Hence, the object and resource information are not mandatory in the event message, which led to moving it to the Correlation Data. Thus, the Payload Data only includes the value to be written to the resource. Furthermore, an additional attribute ("operation") is used as Meta Data for describing the operation performed. Instead of "set" for writing a value to a resource, the value "get" can also be used in order to force the report of an event with the current value of the resource(s) specified. In this case, the Payload Data can be omitted.

Operations format are very similar to events. They are mainly used to configure a device's MQTT topics, either the subscribed ones or the ones in which it must publish every event. The possible operations to be made on devices will be addressed later on section 4.6. Nonetheless, their format is as shown in snippet 7, where the "operation" attribute, included as Meta Data, specifies the name of the operation to be made and the Payload Data contains all the necessary parameters for that function.

```
{
    "operation": {
        "metaData": {
            "operation": OPERATION NAME
        },
        "payloadData": {
            "parameter1": VALUE,
            "parameter2": VALUE,
            etc.
        }
    }
}
```

Snippet 7: Example operation message format.

4.4 ARCHITECTURE

Considering both the objectives and the adopted technologies defined in sections 4.1 and 4.2, figure 4.3 shows the diagram of the currently implemented platform, now considering the WSO2 CEP engine and the SCoT platform as the central broker. In comparison with the diagram presented in figure 3.7, it is noticeable the exclusion of the user management component, left for future work as previously mentioned. Additionally, the diagram also shows that, besides the management of the WSO2 CEP engine which is only attainable using SOAP, all other communications rely on MQTT protocol, using the MQTT broker existent in the SCoT platform.

Hence, the work of this dissertation is mainly focused on the Building Management component, further addressed in section 4.6. It is able to configure and control a server hosting the WSO2 CEP system, and use the SCoT platform for creating the Smart Environment, in this case an automated building. Additionally, considering that this work targets a real world scenario, a gateway implementation is also presented in section 4.8, allowing the connection of real devices to the platform.

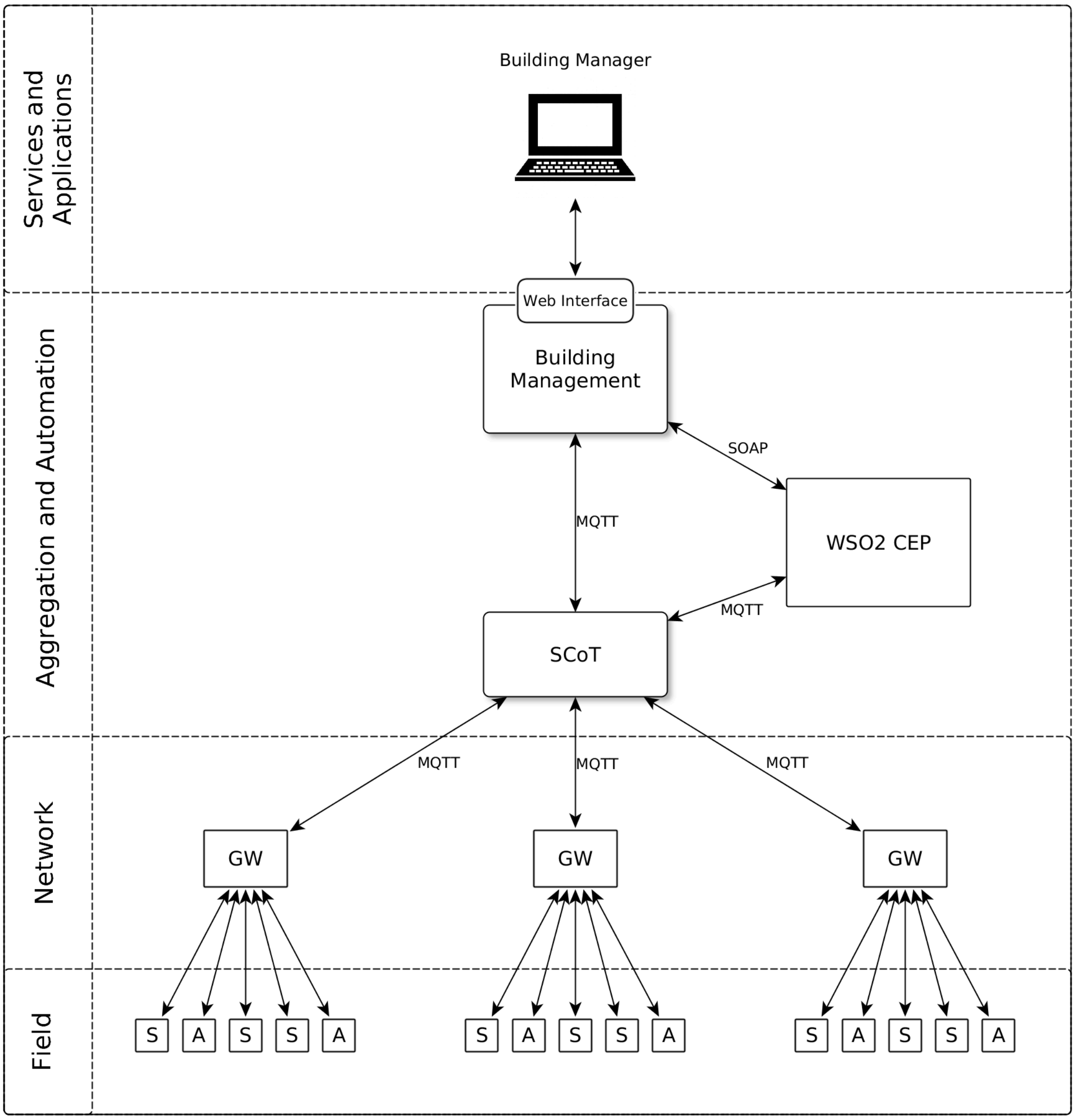


Figure 4.3: SmartEnvironment: architecture diagram

4.5 WSO2 CEP

WSO2 CEP was the chosen CEP engine for this implementation, as explained in section 4.2, for being fully open-source, containing a powerful and familiar language, and providing several ways for both receiving and sending information. Despite already described in chapter 2, this section intends to better explain the primary concepts of WSO2 CEP, by identifying and describing the main elements that form the structure of a continuous stream processor. Figure 4.4 shows a basic example of a flow diagram containing these elements, which are described as follows:

- Event Stream ("InStream" and "OutStream"): this is the rudimentary element of the whole CEP platform. Event streams are what define the data and messages format for

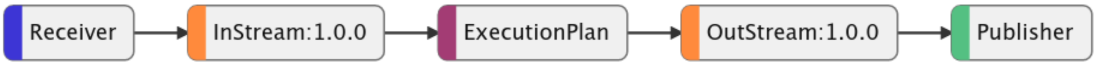


Figure 4.4: WSO2 CEP flow diagram example

a continuous stream of information. From the execution plans point of view, it works as an SQL table where queries can be made;

- Receiver: as the name suggests, it refers to an interface for receiving data. WSO2 CEP supports several types of receivers that are able to collect data from multiple sources and insert it into event streams;
- Publisher: the tasks performed by a publisher are in everything similar to the ones performed by a receiver, except on the event flow direction. When connected to an event stream it is responsible for sending out every event from that stream;
- Execution Plan: this is where all the logic is implemented. An execution plan is a list of functions and queries made in Siddhi QL that allows processing and correlating data from multiple event streams, resulting in new events that are inserted in new streams for being published.

The elements described above are, in fact, everything that is needed to create an automation platform capable of rapidly processing large streams of events. However, this is all defined statically using a web interface, where simple changes could take hours to define. As an example, to add new sources of data, it would be necessary to create new event streams and corresponding receivers, and change the Siddhi code to include those new sources, along with other new operations to do over those streams, such as filtering data. This is what lead to the creation of the building management platform, further discussed in section 4.6, whose main purpose is to develop modules capable of dynamically managing all these elements of WSO2 CEP, based on rules specified in a user-friendly web interface.

The process needed to have a working WSO2 CEP system is very easy. In fact, the only requirement is to have Java 1.8 installed. WSO2 products are generally managed internally using SOAP web services, which are used by the web interface to manage and configure the system. They are actually the only interface available for managing the WSO2 CEP services. Thus, in order to allow the use of these web services by the Building Management component, a configuration file had to be changed to enable their discovery, by setting the *HideAdminServiceWSDLs* property to *false*.

4.6 BUILDING MANAGEMENT

The growing success of building automation is largely due to the autonomy given to the managers responsible for each building. In order to guarantee this autonomy, a system was implemented around Siddhi language allowing easy interaction with users, even if they only hold a few basic concepts, thus bypassing the complexity of the language.

The person responsible for the building management is, therefore, provided with a highly intuitive and dynamic interface in which he only needs to define and configure the desired rules and apply them to the building. It is the system that is responsible for generating the Siddhi code that forms that rule.

Additionally, in order to minimize the configuration effort, the system is also capable of discovering the devices and provide means for the building manager to define to which areas the devices belong to. Furthermore, it also automatically configures all the devices for publishing/receiving events to/from the right topics.

Hence, the building management platform, built in Django[1], comprises two main applications: a structure manager; and a rule manager. The high-level diagram for this platform is shown in figure 4.5.

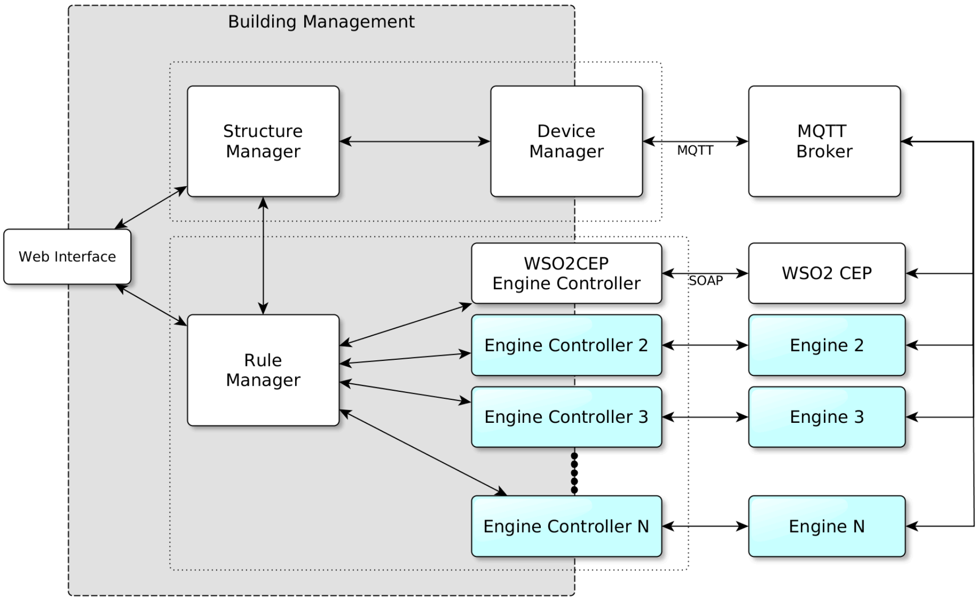


Figure 4.5: Building Management diagram

The Structure Manager is the application responsible for configuring devices according to their location in the building. It provides a web interface for creating every element of the building's structure, as well as the devices positions, and makes use of the Device Manager module for configuring them through MQTT.

[1] A Python Web framework.

Similarly, the Rule Manager shares the web interface with the Structure Manager application, from where the building manager can configure rules in a graphical manner. Then, using the WSO2CEP Engine Controller, it applies the rules on the WSO2 CEP engine, which also makes use of MQTT for controlling devices. We can also see in the diagram additional Engine Controllers (in light blue colour), which intends to show that additional engines may be added to the system as long as an Engine Controller implementation is provided.

The following sections aim to better describe both these applications, referring to all their modules and the potential they provide.

4.6.1 STRUCTURE MANAGEMENT

The structure manager application, is the part of the system that holds a virtual representation of the building, in order to allow more complex selection of data in the rules. For instance, a rule can be applied to a set of specific sensor types in all the bathrooms of a specific floor in a specific building. This structure is persisted in a simple database, whose diagram is shown in figure 4.6.

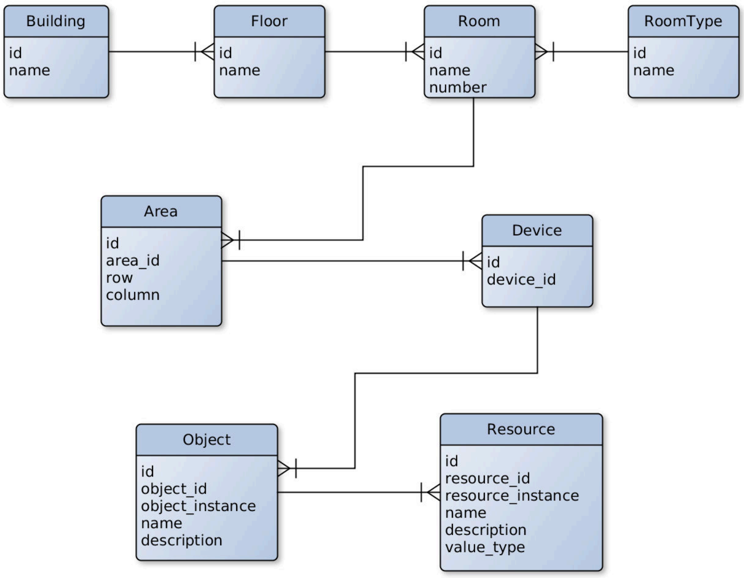


Figure 4.6: Structure Manager Database

Besides the entities corresponding to the structure elements, which mainly have an ID and a name, the diagram also shows entities for areas, devices, objects and resources. The object and resource entities correspond to the objects representation discussed in section 4.3, and thus also contain fields for holding their ID and instance number. Additionally, a name and a description is also held in the database mainly for better describing both the objects and the resources in the graphical web interface.

The device only contains its ID in the database and a fixed string corresponding to the unique device ID, while the Area entity, was defined in order to enable grouping of devices inside a room. It is important to note the “row” and “column” fields of the Area entity. They allow the representation of room areas in a matrix, providing a notion of their location. This is important towards allowing even more complex rules. For instance, a rule can be configured to, when motion is detected, light up not only the current area but also the areas around it. Moreover, the purpose of having an area ID is to include a natural numeration of areas inside each room, and thus allowing repetition of IDs across different rooms, i.e each room has its areas numbered in a natural order starting from number 0.

Linked to the structure manager application, is the device manager module. This is the module responsible for discovering and configuring devices. Internally, and for each device, based on the information available on the database, it generates the topics necessary for enabling the device to publish and receive events. It makes sure it subscribes not only its individual topic but also the topics of the structure elements it is in. In addition, as previously discussed in section 4.3, it also generates the topics necessary for supporting the new wildcard for event publishing.

Having this in consideration, the topics for addressing device’s resources, follow the format:

```
../Building/Floor/RoomType/Room/Area/Device/Obj.ID/Obj.Instance/Res.ID/Res.Instance
```

The module makes use of seven simple operations that must be supported by devices, who have a list of subscribed topics from which they listen for events and a list of publishing topics to which they push each of their own events. The operations, which are sent to the devices using MQTT messages with the format already specified in section 4.3, are shown with a brief description in table 4.1.

Operation	**Description**
subscribe_topic	Add a topic to device’s subscriptions
unsubscribe_topic	Remove a topic from device’s subscriptions
add_publish_topic	Add a topic for device’s publishing topics
remove_publish_topic	Remove a topic from device’s publishing topics
report_device_info	Request a device to report the information about the objects and resources it provides
unsubscribe_all	Remove all device’s subscriptions
remove_publish_all	Remove all device’s publishing topics

Table 4.1: Devices operations

Lastly, this is all managed through an intuitive web interface, where the manager can easily create, edit or delete buildings, floors, room types and rooms. Moreover, in each room, the manager can dynamically create areas, to which he can associate devices, with a drag-and-drop interface.

4.6.2 RULES MANAGEMENT

The rules manager is the most complex application of the whole Smart Environment implementation. Although WSO2 CEP provides a lot of possibilities for creating complex queries with its powerful language, Siddhi QL, they are difficult to map to a Graphical User Interface. With this in mind, one of the first principles taken into consideration when starting this implementation, was to design a solution that would be highly extensible, with pluggable new modules.

In order to achieve that, and taking advantage of the Django's Object-Relational Mapper (ORM), the database, whose diagram is shown in figure 4.7, was implemented in a way that new modules could be added just by dropping them on a specific directory of the application. These modules just need to inherit one of the defined entities, which are shown in green in the database's diagram, and enrich them with extended features. In fact, the "Pattern" and "Sequence" are special modules that inherit a "function", in order to endow the system with the ability of detecting patterns or sequences. Appendix A shows an example of a module implementation, where all the mandatory methods are shown. It is important to note that even the parameters that must be specified by the building manager in a module can be defined in its implementation. It is the platform that is responsible for generating the necessary web forms to match them.

The diagram shows that, a Rule, which can be applied on an Engine, is formed by actions. Each action must contain a Target and a Function. The Target defines to where the final events will be sent, while the Function defines how and when that events are sent. It is at the function that all the necessary calculations are made. The input data for the function is provided by the InData module. The latter, must necessarily have a Listener, which defines from where the events will be received, and optionally include other modules that allows selecting and make basic transformations over the event stream.

There are four types of modules that enable the transformation of the event stream. Those are the Aggregators, Converters, Windows and Filters. Filters can be used with event streams to filter events based on the given filter condition. Regarding Windows, they allow the CEP engine to collect a subset of events based on a criteria. A very common example of Windows are the time windows that can capture all the events that arrived in a given time period. Aggregators, as the name suggests, are commonly used with Windows to perform aggregate calculations, such as summing the events or calculating averages. Finally, Converters are used to make calculations on each event. Their most common use is to make unit conversions.

As already stated, two special functions are provided in order to allow the detection of

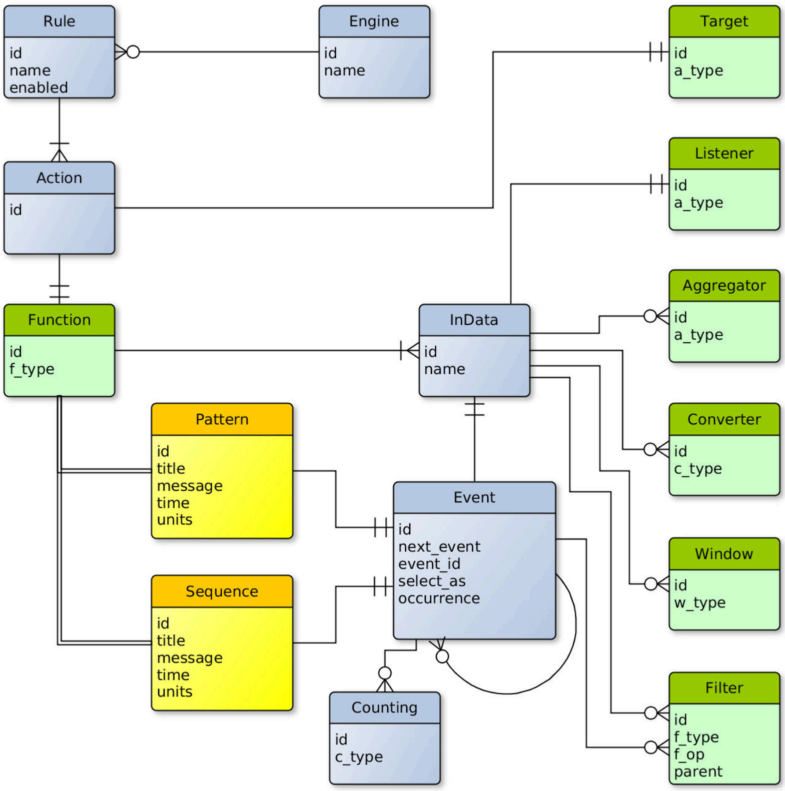


Figure 4.7: Rule Manager Database

patterns and sequences, which are one of the most frequent and powerful methods of the Complex Event Processing concept. Along with other optional parameters, they are both mainly formed by Events. Events are organized in ordered linked lists (defining the order of the desired detection), each of them contain an InData object. Generally, the InData is at least endowed with Filters for matching a specific condition. Thus, a sequence is found when all the specified Events, have a match in their InData, in the order they were defined. Patterns are very similar to sequences, but they allow other events to exist between matched events. As an example, a pattern could be the detection of an increase of temperature in a room of more than 10 degrees within 15 minutes. While that can be made by detecting that change between any two events in a 15 minutes window, a sequence could only detect that change if they were consecutive events.

In order to provide a functional system, implementations for all the inheritable modules were made. Table 4.2 shows the implemented modules along with a brief description on what they do.

All these modules can be used in a user-friendly interface for creating complex rules, that are then applied by the system on the CEP engine. However, instead of directly creating the Siddhi code to be sent to the WS02 CEP, the system represents the whole rule in a JSON object. The reason leading to this was, one more time, to allow a high extensibility of the system. By creating JSON rules in a specific format, the system is not obliged to use WSO2

Type	Module	Description
Window	Time Window	Capture events in a predefined time
	Length Window	Capture a predefined number of events
Aggregator	Average	Calculate the average value of all the events in a window
	Any	Returns 1 if any of the values in the window is greater than 0
	None	Returns 1 if all the values in a window are 0
Converter	Lux To Percentage	Calculates an output percentage value to apply on a luminaire's dimming level based on a value in lux units
	Set 1	Sets the value to 1
	Set 0	Sets the value to 0
Filter	Time Greater Than	Allows filtering events that arrive after a given time
	Time Less Than	Allows filtering events that arrive before a given time
	Equal	Filter events with the value equal to a given value
	Not Equal	Filter events with the value different than a given value
	Greater Than	Filter events with the value greater than a given value
	Less Than	Filter events with the value less than a given value
	Greater or Equal Than	Filter events with the value greater or equal to a given value
	Less or Equal Than	Filter events with the value less or equal to a given value
Listener	MQTT	Receive events through MQTT
	HTTP	Receive events through HTTP posts
Target	MQTT	Send events through MQTT
	HTTP	Send events through HTTP posts
	E-mail	Send events by email
Function	Set Value	Set the value received from input streams to the output streams
	Set Percent	Set a percentage of a value from input streams to the output streams, based on boolean data from another input stream

Table 4.2: Implemented Modules

CEP specifically, and thus other CEP engines can be used and even coexist. To achieve this, as already shown in figure 4.5, an engine controller must be implemented to support a different CEP engine. The primary task of an engine controller is to parse the JSON rule and convert it to code supported by its engine. With this approach, although not implemented, an engine controller is all that is needed to support automation at the gateways.

To better understand the JSON rules generated by the system, consider the example rule shown in snippet 8, which contain a single action.

```
{
  "name": "Labs Lights",
  "subrules": [
    {
      "actions": [
        {
          "target": {
            "type": "mqtt",
            "topic": "/SM/out_events/IT2/Floor_1/Lab/1.10/1/*/3302/*/5851/*",
            "value_type": "int"
          },
          "function": {
            "name": "setif_value_percent",
            "listen_boolean": {
              "type": "single",
              "listeners" : [
                {
                  "type": "mqtt",
                  "topic": "/SM/IT2/Floor_1/Lab/1.10/1/+/3302/+/5500/+",
                  "value_type": "int"
                }
              ],
              "window": {
                "type": "time",
                "units": "seconds",
                "value": 6
              },
              "aggregator": {
                "type" : "any"
              }
            },
            "listen_value": {
              "type": "single",
              "listeners" : [
                {
                  "type": "mqtt",
                  "topic": "/SM/IT2/Floor_1/Lab/1.10/1/+/3301/+/5700/+",
                  "value_type": "float"
                }
              ],
              "window": {
                "type": "length",
                "value": 5
              },
              "aggregator": {
                "type" : "avg"
              },
              "converter": {
                "type" : "lux_to_percentage",
                "value": 50
              }
            },
            "percent_if_true": 100,
            "percent_if_false": 50
          }
        }
      ]
    }
  ]
}
```

Snippet 8: JSON rule example.

The first important aspect to note from this rule, is the existence of the "subrules" key. Since the system supports reading and publishing data from different places in the building, each rule may be divided in several smaller rules that addresses each of those places. Then, we observe that the action is formed by an MQTT target and a "setif_value_percent" function. The function contains two sources of data: one for listening to events that will be treated as boolean values (corresponding to motion sensors) and other for listening for events to be treated as float values (corresponding to illuminance sensors). The boolean data is read for a time window of 6 seconds and aggregated with the "any" Aggregator which, as seen on table 4.2, detects if there was any motion in the last 6 seconds. The data containing the values of the illuminance sensors, is aggregated with an average over the last 5 events, and converted using a "lux_to_percent" converter. Hence, the function sends to the target an event with the value of 100% calculated percentage if motion is detected in the last 6 seconds, and 50% otherwise. These values in particular are application dependent.

4.6.2.1 ENGINE CONTROLLER

In the same line with the pluggable modules, engine controllers can also be dynamically added to a specific directory of the application. In fact, they just need to implement a python interface to be ready for use by the system. The interface contains few methods that are used for managing the rule on the engine (add, update, remove or check if it is supported), and to set some general configurations, such as the engine's host address. These methods are described in table 4.3, along with the parameters they should receive.

In order to use WSO2 CEP engine, its controller had to be developed by implementing the interface methods shown in table 4.3. The final controller architecture is presented in the diagram of figure 4.8, where besides the Engine Interface to which the Building Management system interacts with, other components that back it up are shown.

The WSO2 Configurations component shown in the diagram is responsible for persisting the main configurations of the whole engine. It is intended for general configurations which

Method	Parameters	Description
get_config	-	Return a python dictionary with the engine configurations. May include relevant addresses and ports regarding to the engine.
set_config	Python dict with configurations	Override current engine configurations with the ones provided as parameter.
add_rule	JSON rule	Add a rule to the engine.
update_rule	JSON rule	Update a rule in the engine (the rule name is also included in the JSON string).
remove_rule	JSON rule	Remove a rule from the engine.
rule_exists	JSON rule	Check if a rule exists in the engine.
supports	JSON rule	Check if a rule is supported in the engine.

Table 4.3: Engine Interface Methods.

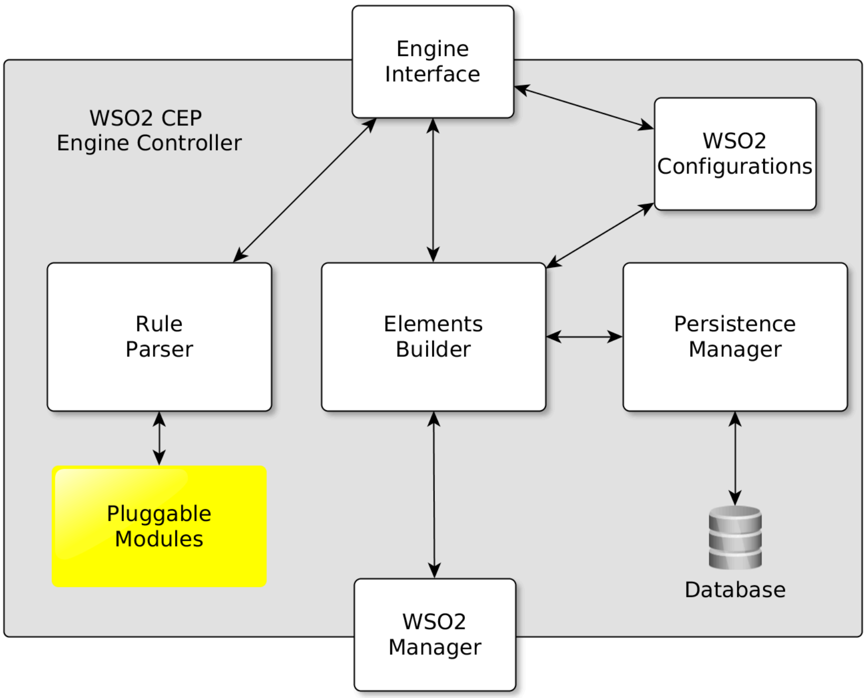


Figure 4.8: WSO2 CEP Engine Controller Architecture.

include the WSO2 Server address and port, as well as the login credentials for accessing its services, and similar fields the MQTT server it uses for receiving events. It relies on python's configparser module, which implements a basic configuration file parser, allowing the provision of a default configuration file for configuring the engine. Furthermore, the first two methods on table 4.3 are implemented using this component, thus enabling the Building Management component to also be able to change these configurations.

The Rule Parser, as the name suggests, is intended for parsing the JSON rule and not only convert it to Siddhi code, but also identify the needed event streams, receivers and publishers. Here, a similar approach to the one used in the Rule Management system (described in section 4.6.2) is adopted, allowing new modules to be easily added to the controller. In fact, each time a new module is added to the Building Management system, its implementation must also be included in a controller in order to support it. WSO2 CEP engine controller currently implements all the modules referred in table 4.2. Appendix A shows an example of both a module at the Building Management platform and its implementation for WSO2 CEP engine controller.

After parsing a rule, the Rule Parser module provides a structure containing both the Siddhi code for the rule, along with the identified event streams, receivers and publishers. Then a central module, the Elements Builder, is responsible for receiving that structure and build all the elements necessary for creating a rule at WSO2 CEP engine, as presented in section 4.5.

It was also important to include here an additional module, the Persistence Manager.

Its main purpose is to hold some information about the rules, event streams, receivers and publishers created, and enable their reuse in other rules. It makes use of SQL Alchemy (an open-source python ORM), for keeping a local SQLite database with that information. Its main methods aim to inform the Elements Builder module, when it is deploying a rule, of which elements must be created and which ones must not. Moreover, it also provides methods for finding orphan elements, whenever a rule is removed from the engine, allowing it to act as a garbage collector and thus remove every unused element.

Finally, the WSO2 Manager module makes use of the only available interface for managing the WSO2 CEP, SOAP Web services, applying the entire rule on the engine. It allows getting, adding, updating and removing each of the elements existent on the WSO2 CEP engine, using simplified Python dictionaries and thus creating an abstraction layer between Python and WSO2's SOAP services.

To finish, another important feature delivered by this approach, is that an engine controller can be replicated, using the same engine but with different controllers and thus different configurations. Hence, it is possible, for instance, to use a replicated engine in a simulated environment, allowing to test any rule before applying it in the real world. The next section presents a devices simulator, which despite being mainly developed for testing the platform, can also be used with this purpose.

4.7 DEVICE SIMULATOR

The device simulator is a software implemented in Python that is able to reproduce the behaviour of both sensors and actuators. It meets both the objects definition and the message formats defined in section 4.3, and implements all the operations defined in table 4.1.

The simulator is organized as shown in figure 4.9, where we can see two main modules: the Task Scheduler and the Web Server. The Task Scheduler is responsible for reading the configurations defined in the configuration file, corresponding to the existing devices and their properties, and launch different threads for each of them. Thus, each device runs independently and is able to send/receive events, using MQTT protocol.

All the configurations are loaded from a single configuration file, where besides some general configurations mainly related to the MQTT server and web server properties, the objects and resources that each device should have are specified. Moreover, it is also in this configuration file that value generators may be appended to resources. Value generators are responsible for simulating events in order to provide a test scenario as close as possible to reality and thus allowing to infer better conclusions.

There are a few types of generators provided for use with the simulator, allowing not only

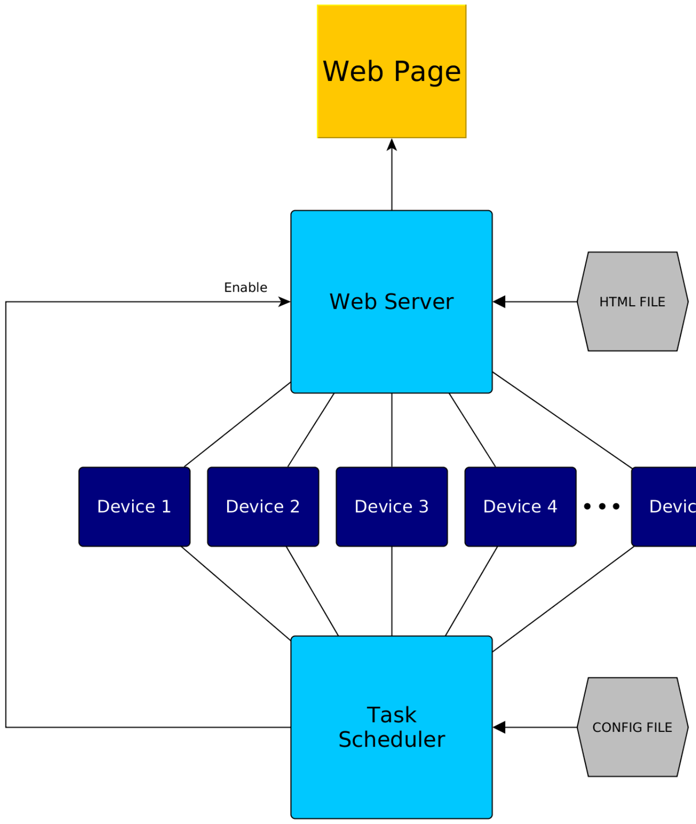


Figure 4.9: Simulator - Architecture

to simulate events with random values but also with real values from existing databases, either using raw or Comma-Separated Values (CSV) files. Actually, most of the events currently generated by the simulator contain values from databases provided in [110].

Additionally, the log of an existing mobile application (Motion Detector [111]) which can be used for detecting motion using a built in camera can also be used to provide real data to the simulator. In fact, in combination with Tasker [112] and its MQTT plug-in, it is possible to provide live motion events to the simulator using a smartphone or a tablet.

Besides the creation of simulated devices, the simulator is also endowed with a web server built with Flask, that allows visualization of the device's states. Here, the IT2 building's plant was included, not only because it is the building where this implementation will be deployed, but also to simulate a number of devices close to the expected for the SmartLighting project. Thus a view closer to the future automation and intelligence platform behaviour is expected. Figure 4.10 shows the main overview given by the simulator's web server, where only the luminaire's states can be seen (as little squares) over the building's plant. When a luminaire is on its square is coloured, being the colour dependent on the type of room. Green colour is used for corridors, yellow for bathrooms and red for rectangular luminaires. Blue is used for the rest of the rooms.

When a room is clicked on the plant, its overview is shown, as seen in figure 4.11, with

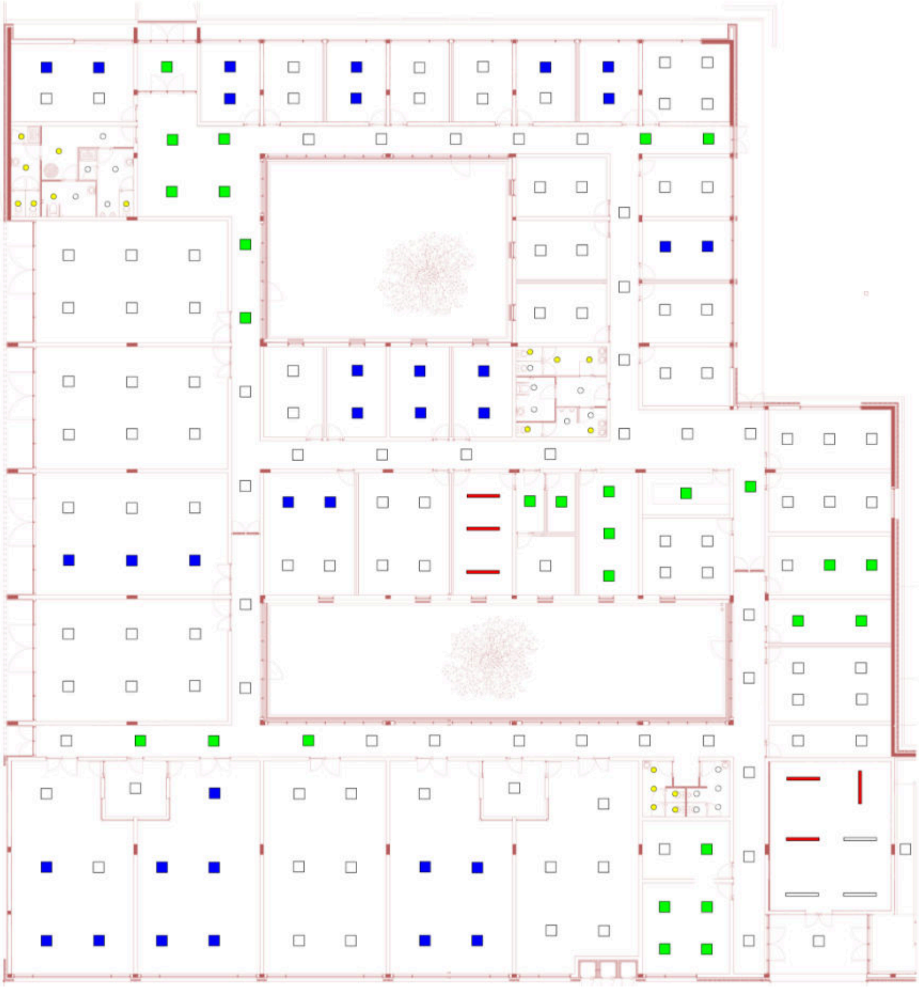


Figure 4.10: Simulator - Plant View matching IT2 building.

all the available information from devices. Here, the luminaires show their dimming level inside their symbol, while the circles, which refer to motion sensors, turn red when motion is detected. With the green colour are the illuminance sensors which show their current value in lux. Finally, at the right side, three device values are show, which represent the air conditioner state, and the values from a temperature and a humidity sensor.

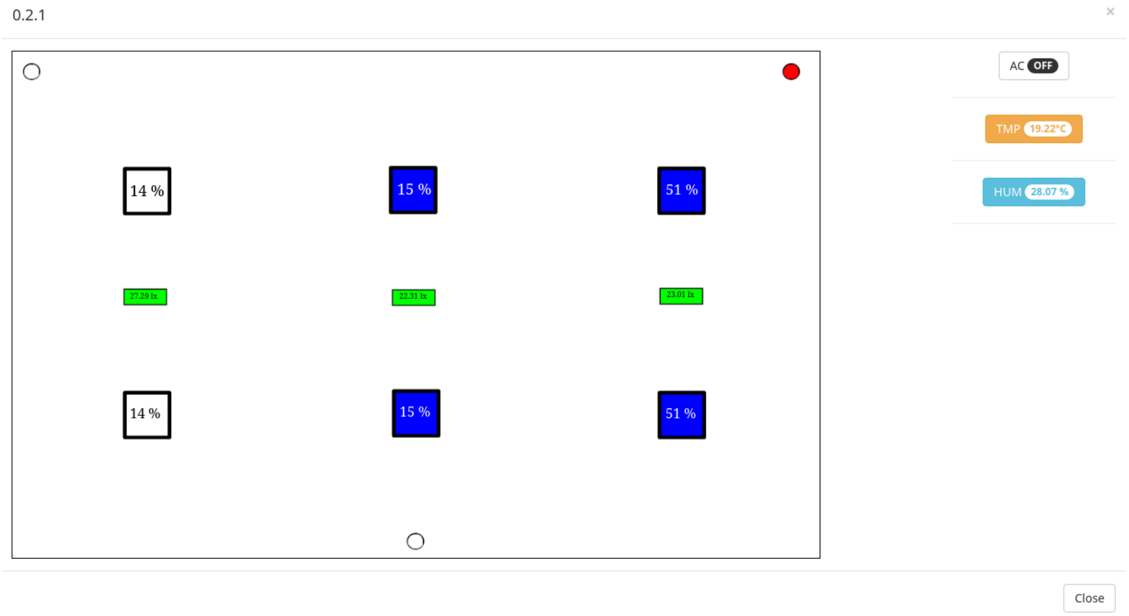


Figure 4.11: Simulator - Room View.

4.8 BLE GATEWAY

In order to allow the connection of the real devices through BLE, developed as the work of two other dissertations from the Integrated Circuits group of IT, a Python agent was developed for transporting information between them and an MQTT broker. Besides transporting information between the two different technologies/protocols, it is the gateway who is responsible for providing the objects definition and use the messages format defined in section 4.3, as well as the operations defined in table 4.1. In fact, it is the provision of these aspects that creates the abstraction layer that makes it possible to have both real and simulated devices working simultaneously.

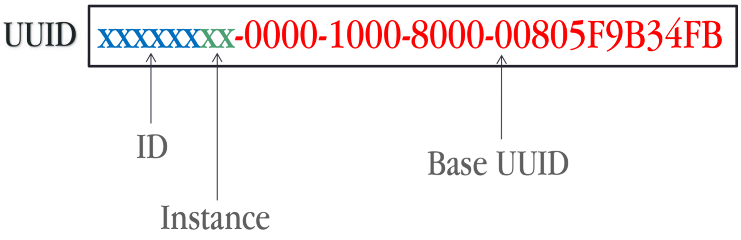


Figure 4.12: Object/Resource mapping in BLE's UUID

Bluetooth Low Energy, as described in chapter 2, divides a device's properties in services and characteristics which are, in comparison with the objects representation presented in section 4.3, similar to objects and resources. Having this in mind, the implementation for the gateway was focused on mapping services to objects and characteristics to resources. However, instead of a simple device ID, BLE uses a more complex approach, by using UUIDs

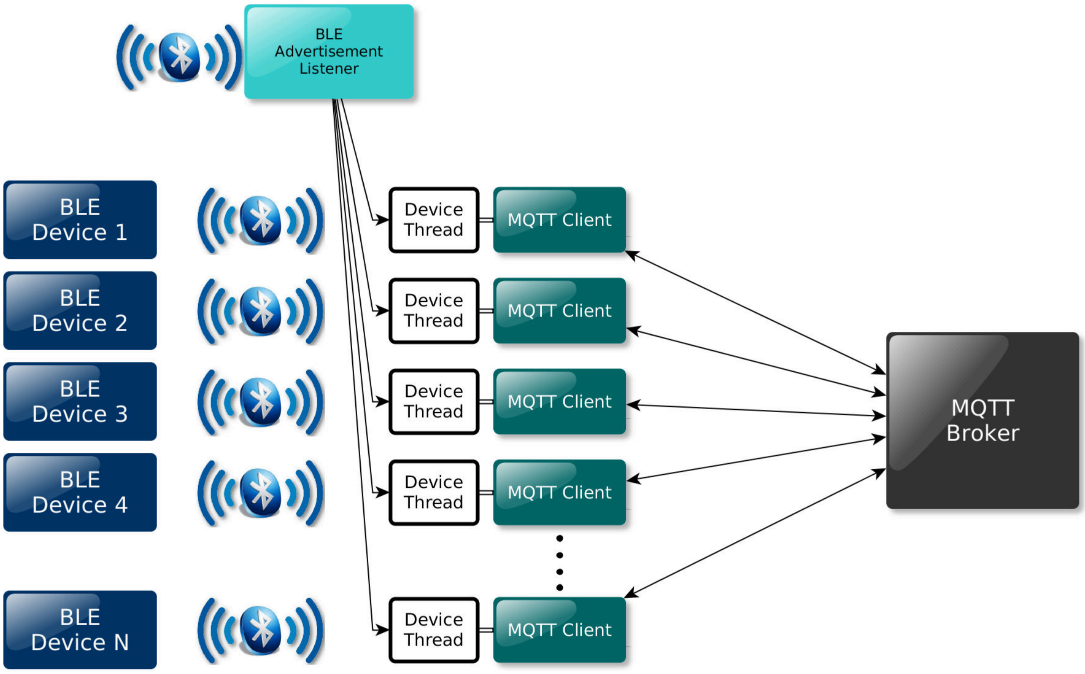


Figure 4.13: Simulator - Room View

for identifying both the services and characteristics. Thus, in order to achieve the desired mapping, the ID and the instance of both the objects and resources are included in the UUID, as shown in figure 4.12. Additionally, the base UUID must be replaced with one corresponding to the entity that is defining this new services and characteristics, since the one on the figure corresponds to the Bluetooth SIG. A random one is used for testing purposes.

In figure 4.13, where the gateway's architecture diagram is shown, we can see how the gateway operates. It contains a BLE advertisement listener, that launches a new device thread for each BLE device found. Then each device thread, having its MQTT client, is responsible for listening for new notifications from BLE devices in order to publish them as events to the MQTT broker, and write new values in BLE device's characteristics each time it receives an event from MQTT broker.

The connection to BLE devices relies on bluepy library [113], which is an open-source Python interface to BLE on Linux. Besides other objects for representing, for instance, a Service or a Characteristic, bluepy provides two main classes: Scanner and Peripheral. Scanner is the class that allows having an always active thread, listening for BLE advertisements from devices. It is provided with a Delegate object containing the function to start a new device thread, which is called every time a new device is found.

Regarding the Peripheral class, it is the core of the bluepy library that provides all the methods for interacting with the BLE device, and thus an object of it is instantiated in each device thread. It is constructed with the unique 6-byte MAC address of a device as parameter, leading to an immediate connection to the BLE device containing that address.

After that, the interaction with the BLE device is mainly made through methods for getting the list of services, characteristics and descriptors the device provides, reading and writing characteristics, and receiving notifications.

For receiving notifications, the Peripheral class provides the method *waitForNotifications*, which takes exactly one argument consisting on the time in seconds for which the method should block and listen for notifications. It is also provided with a Delegate object containing the method to be called when a notification is available, which generates an event and sends it through the MQTT client. However, since the Peripheral class is not thread-safe this approach is not the most correct and results in an increased latency at the BLE connection. The problem is, when the device thread is listening for notifications, it is blocked on *waitForNotifications* method and thus is not able to write new values in characteristics until it finishes waiting for notifications.

By looking at the library's source code, it was possible to verify that the *waitForNotifications* method consisted on a *poll*[2] method with a registered file descriptor corresponding to the output of a another subprocess. Thus, the problem was solved by making some changes in the library itself, which consisted in adding a dummy file descriptor to the existing *poll*. Hence it allowed the inclusion of an *interrupt* method consisting of writing a single character to the dummy file descriptor, and thus enabling the interruption of the process of waiting for new notifications whenever a write operation in a characteristic is needed.

Summary

Considering the presented architecture, as well as the approaches followed in the implementation of each of its components, it is important to understand whether it meets the requirements. In fact, it was already discussed in chapter 3, how the presented solution addresses the defined requirements. Thus, since this implementation strictly follows the presented solution and its architecture, most of the requirements are naturally met. Additionally, the extensibility of the system is enhanced, through the use of an approach for easily adding new modules to the system and thus supporting new functionalities and features. Hence, the system is further able to meet the specified use cases.

However, as already stated in section 4.1, some parts of the solution were left for future implementation. Therefore, the use cases related to user interaction with the system are not met, as well as the failure handling requirement. This, was directly associated with the provision of processing engines at the network layer or, more specifically, at the gateway.

[2] The *poll()*, similarly to *select()*, is a method provided by the operating system that enables synchronous I/O multiplexing, allowing programs to monitor several file descriptors, and block until at least one of them is ready for reading or writing.

CHAPTER 5

Evaluation and Results

In this chapter, the crucial step of validating the solution proposed in chapter 4 is addressed. It begins with a description of the deployment scenario, in section 5.1, which also includes the tools used, some of which particularly developed for this process. This is followed, in section 5.2 with an analysis of the obtained results for several performance metrics. Taking into consideration the requirements presented in chapter 3, the effectiveness and performance of the platform is evaluated by addressing both the WSO2 CEP system load and performance, as well as the end-to-end latency, under different usage conditions.

5.1 DEPLOYMENT SCENARIO

The final deployment scenario, as depicted in figure 5.1, comprises the Smart Environment platform, the WSOP2 CEP engine, a simulator for virtual devices and a physical gateway for connecting physical devices, being that all of these communicate using the MQTT broker included in the SCoT platform.

The gateway is responsible for connecting the physical devices to the SCoT platform, by mapping them as objects which are to accessible by the rest of the components. It is deployed in a Raspberry Pi version 3 which includes built in Wi-Fi and BLE, the later used for the wireless connection to the physical devices.

Regarding the simulator, its intent was to add complexity in a controlled fashion by adding simulated devices to the scenario. This allowed to test the platform's performance with a larger amount of events generated per second. It was implemented and hosted in a Linux server equipped with a single-core processor and 1 gigabyte of memory.

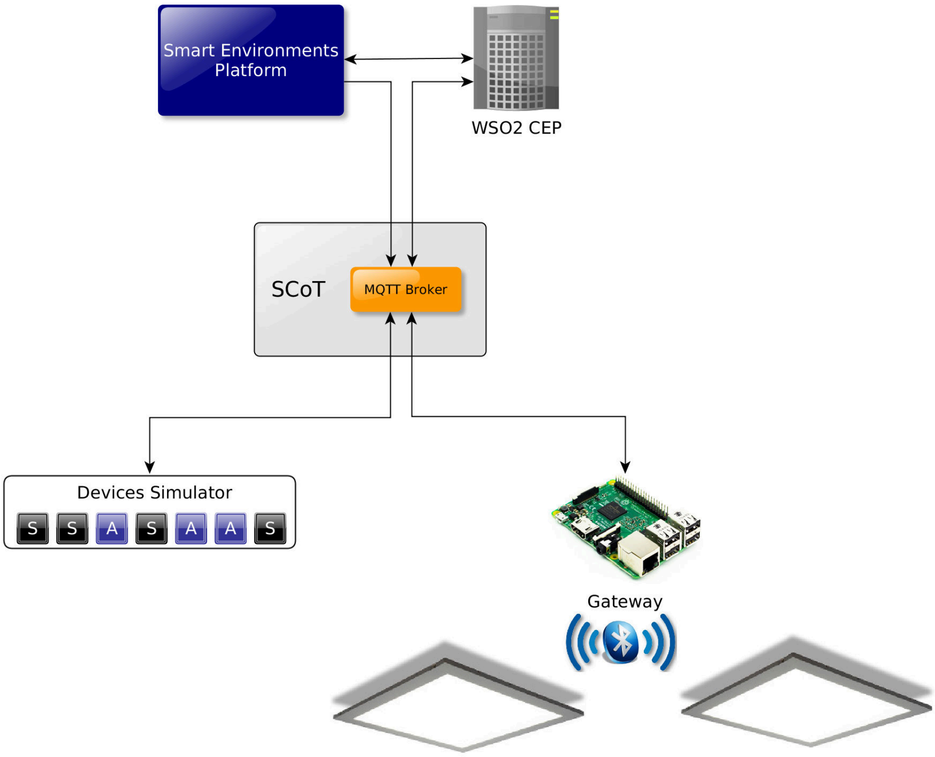


Figure 5.1: Deployment Scenario

The Smart Environment platform, which is hosted in the same server as the simulator, is the component responsible for providing the user-friendly web interface, in which the building manager is expected to specify the management and automation rules that establish the automated environment. The platform would then generate all the necessary elements to deploy those rules in the WSO2 CEP engine.

Finally, the WSO2 CEP engine is the core component of the whole scenario, being responsible for processing all the logic specified in the rules. It was hosted in a Linux server endowed with a quad-core processor and 4 gigabytes of memory.

The final deployment configuration could be controlled through the device simulator. Through it, it was possible to include multiple sensors for reading temperature, humidity, illuminance and motion, and actuators for changing both the state and dimming level of luminaires as well as air conditioners states. Temperature and humidity sensors generated events every 10 seconds, while illuminance sensors report their value with a period of 5 seconds. Motion sensors were thought to have a random periodicity between 1 and 10 seconds.

Regarding physical devices, the scenario was able to include devices through the gateway. In particular, three devices (developed by the project colleagues as their dissertation work). Two of them contained a Passive Infrared (PIR) sensor for motion detection, an illuminance sensor, and a luminaire with all its resources. The third device was equipped with additional sensors for reading temperature and humidity, as well as sensors for detecting gas or flame.

The scenario includes 679 simulated devices which include sensors for reading temperature, humidity, illuminance and motion, and actuators for changing both the state and dimming level of luminaires as well as air conditioners states. Temperature and humidity sensors generate events every 10 seconds, while illuminance sensors report their value with a period of 5 seconds. Motion sensors though, have a random periodicity between 1 and 10 seconds.

5.2 PERFORMANCE RESULTS

This section presents the performed results, starting with a description of the testing configuration, in subsection 5.2.1. The WSO2 CEP provides several information about its performance, which are shown and discussed in subsection 5.2.2. This is followed by the end-to-end latencies achieved at the field devices, presented in subsection 5.2.3.

5.2.1 TESTING CONFIGURATIONS

The process chosen for testing the platform was divided in two phases: i) configuration A considered a basic building structure with simple rules; and ii) configuration B considered an increased complexity in the number of rules as well as a much larger number of simulated devices.

In the first phase, under configuration A, a small segment of a building was implemented with only four small offices along with a larger open office area. These areas were endowed with simulated devices, which were distributed using the provided web interface of the Smart Environment platform. Additionally, a specific room was created in the platform, in which the real devices were linked to and also configured. With this building configuration, one rule for each type of rooms was configured, also using the web interface. Both rules included two actions targeting, respectively, the luminaires states and their dimming levels. The rules defined that, in each area, lights are turned on if motion was detected during the a 10 seconds interval. Regarding their dimming level for that period, a recommended value was calculated from the average of the last 5 events containing the area's illuminance. If motion is detected, the calculated value is used for the first 5 seconds, otherwise the dimming value is set to 25% of that. There was an additional difference between the small offices and the open spaces, whereas in the small offices all the room's lights are turned on if motion is detected, in the open space only the areas surrounding the location of motion detection would have the lights turned on. Appendix B shows one of the actions defined in the web interface, and then presents the flux diagram from two rules, thus showing how much complexity the implemented platform hides from the building manager.

In the second phase, under configuration B, the full IT2 building was considered, as depicted in figure 4.10, and an estimation of existing devices was deployed into the platform. This meant that a total 682 devices (3 real devices and 679 simulated) were used. Here, the number of rules and their complexity also increased significantly by targeting also the air-conditioner system of all the offices. With this approach, not only the performance of the platform and its latency can be tested under strain, but also how these variables affect with the system growth.

5.2.2 WSO2 CEP

The CEP engine is one of the most relevant parts of the whole platform. It is responsible for processing every event and deliver responses as fast as possible. Thus, the analysis of its performance is crucial to understand if it meets the requirements, particularly, the ones related with the fast responsiveness of the platform. Hence, in order to study the performance of WSO2 CEP, information about the server hosting it was collected for one hour of usage, namely its CPU and memory usage, and the average system load.

It was possible to observe the CPU usage remained maximum for both configurations, and even with a small amount of events being generated the CPU is always used in its entirety. This allow us to conclude that WSO2 CEP takes full advantage of it in order to deliver the lowest latencies.

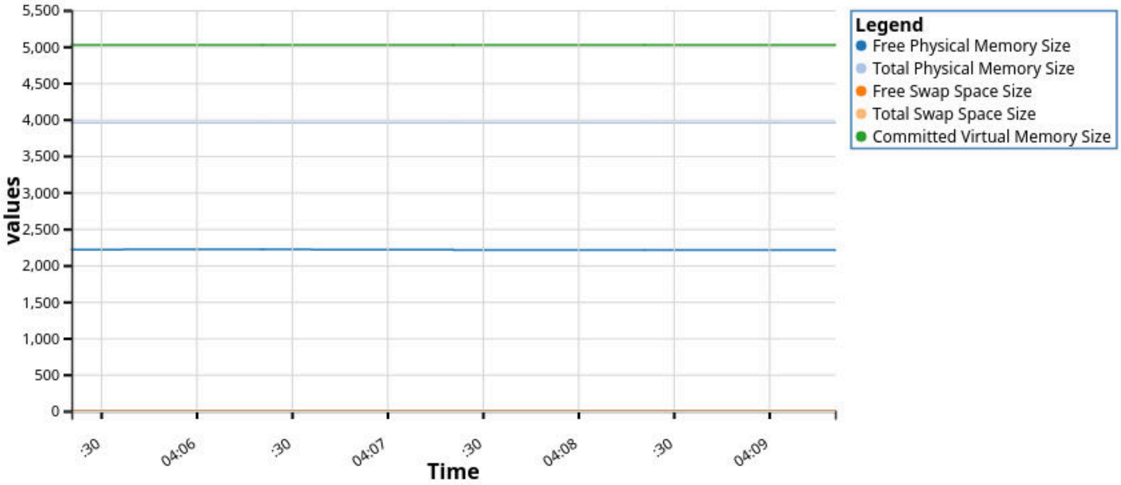


Figure 5.2: Configuration A - Physical Memory Details (MB)

Regarding the memory usage, this CEP system manages it very efficiently, having more than half the physical memory free in configuration A, as shown in figure 5.2, with a relatively small increase of its usage in configuration B (around 1 gigabyte), where more devices and more rules are used, as shown in figure 5.3.

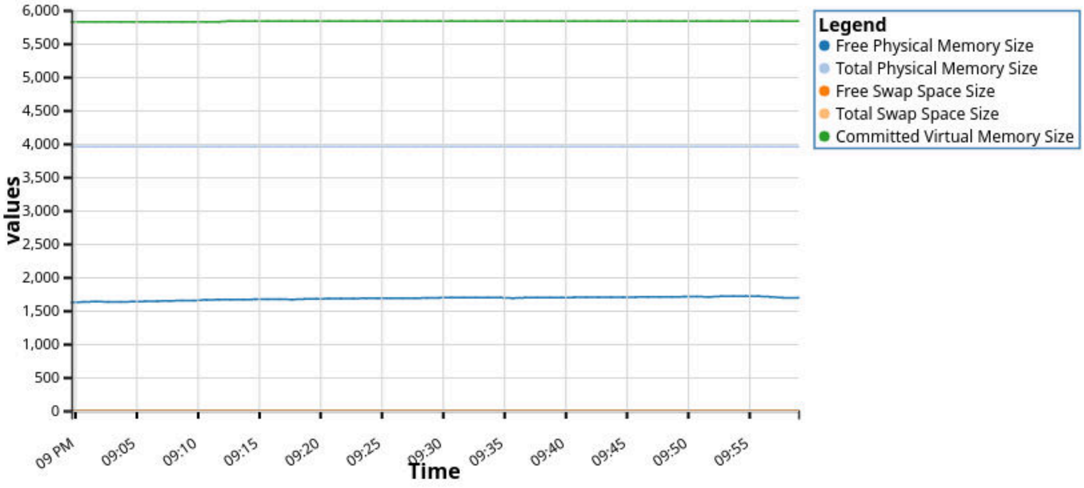


Figure 5.3: Configuration B - Physical Memory Details (MB)

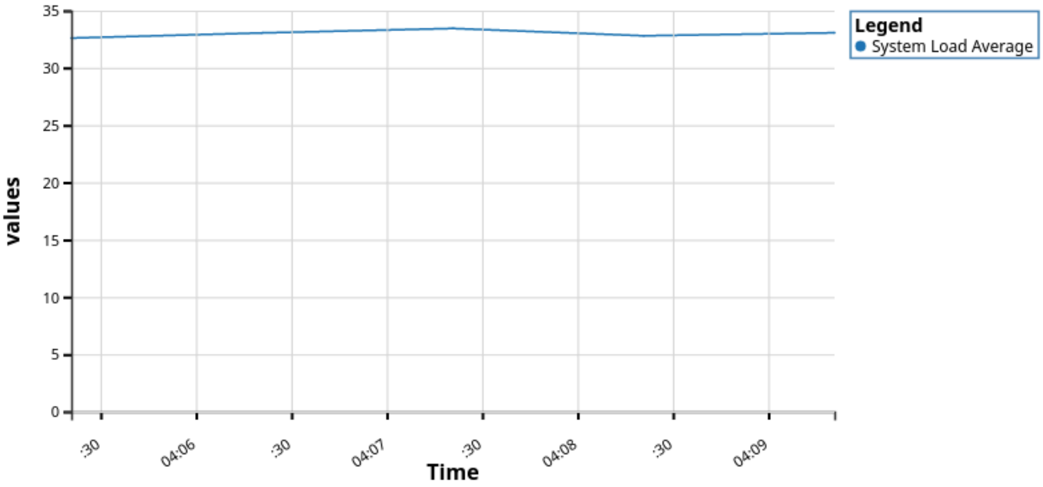


Figure 5.4: Configuration A - Average System Load

The average system load is, however, quite high. Figure 5.4 shows its values measured with configuration A applied, which are between 30 and 35, and then increase to values above 100 in configuration B, as seen in figure 5.4. While this can be seen as very bad in a first impression, it can be explained by the existence of a huge amount of tasks simultaneously. In fact, since there are a lot of rules with a high number of publishers and receivers, it is acceptable the existence of thousands of threads to process them all, which are seen as tasks by the system and are thus considered in the calculation of the system load. Yet, most of them can be blocked in I/O operations without using CPU and thus not affecting the system performance. The load can be, though, lightened by spreading the tasks by a cluster of servers, which is allowed in WSO2 CEP through the use of Apache Storm as already explained in chapter 2.

In summary, WSO2 CEP proves to be able to handle a huge amount of work, being capable of processing hundreds of events per second with an efficient management of the server

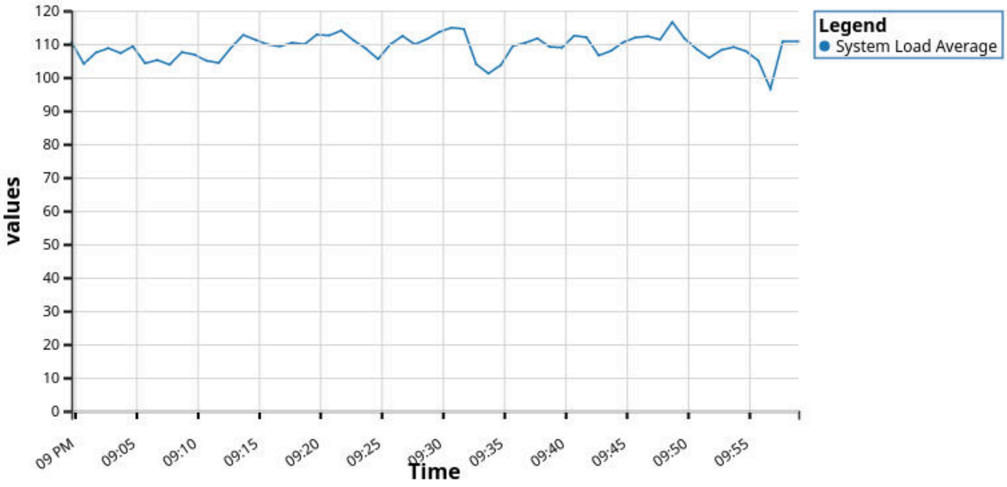


Figure 5.5: Configuration B - Average System Load

resources. Nonetheless, latency must be measured in order to check its fast responsiveness, which is addressed in the next subsection.

5.2.3 SYSTEM LATENCY

The measurement of latency allows for an analysis of the system performance. As previously stated, not all of the building services supported by the platform will require a small latency. However, services like lighting are very sensitive to delays in the chain between input (at the sensor) and output (at the luminaire actuator). Monitoring this parameter and its variation interval over a different usage scenarios, allows to determine whether the system will be able to cope with strict time requirements for near real-time functionality or not.

In order to include not only the processing time of the platform but also the network latencies and the processing times at the devices, the measurement of the total end-to-end latency was performed with the real-life devices. In cooperation with the project colleagues, time was measured between an event generated by a device and the processing of its correspondent reply event. In further detail, the time interval began with the triggering of a sensor, then sent via BLE to the gateway, processed by the CEP engine where a reply was generated. This reply event was passed back through the gateway and communicated to the end node also by BLE. After being processed and the output action executed, the time interval was completed.

Considering both defined configurations, for this test scenario the time between a motion detection and the corresponding action on the luminaire was measured. A routine was implemented on the physical devices and the simulator in order to repeatedly trigger a specific motion sensor which would then communicate to the CEP where a predefined rule would make it send an event to activate a specific luminaire. In the meanwhile, the remaining devices in the simulator operated in a random fashion in order to simulate a real environment, stressing the platform.

For configuration A, 115 latency measurements were taken using simulated devices only, and results are shown in table 5.1. In a second turn, 124 latency measurements were taken using the physical devices, and the results are shown in table 5.2. Additionally, the project colleagues were able to estimate the BLE connection delay to be around 79 ms.

Average Latency	53,92 ms
Minimum Latency	5,88 ms
Maximum Latency	233,33 ms
Standard Deviation	39,32 ms

Table 5.1: Configuration A - End-to-end latency when using a simulated device

Average Latency	149,27 ms
Minimum Latency	68,12 ms
Maximum Latency	379,52 ms
Standard Deviation	55,95 ms

Table 5.2: Configuration A - End-to-end latency when using a physical device

Regarding the simulated device results, since it does not include the BLE connection and thus its delay, we can verify that it has a marginal delay of about 50ms. Considering the delay from the BLE connection, an average latency of around 150 milliseconds is observed at the physical device, which is quite acceptable. In fact, even the slowest response is still under half a second. These results validate the system against the latency requirements previously proposed. Taking into account the most time-critical application considered, lighting, where sub-second (preferably bellow 500ms) latencies are expected, the measured values deliver well within expected.

After configuring all the devices in the simulator and deploying additional and more complex rules, as defined for configuration B, new measurements were taken in order to observe how a significant increase in new processing tasks would affect the platform's performance. For this scenario, since the device simulator was hosted in a server with a single core and a small amount of memory (1 gigabyte), the latency measurements at the simulator did not take place. Behind this decision is the fact that for such a high number of threads, the number of processing tasks is also increased at the simulator. Thus additional delays could take place, which were related to the time each thread must wait to acquire the processor in order to perform the measurement. Hence, the results would not be reliable. The results for using physical devices are shown in table 5.3, corresponding to 86 total measurements.

From these results we can conclude that, the addition of new devices and rules does not have a significant impact in performance. In fact, and surprisingly, even with a higher

Average Latency	108,63 ms
Minimum Latency	68,35 ms
Maximum Latency	446,89 ms
Standard Deviation	45,04 ms

Table 5.3: Configuration B - End-to-end latency when using a physical device

Maximum latency value, the average is lower than before. This difference is tolerable since it is in the error range defined by the standard deviation. Moreover, these measurements were taken in separate days, which could also mean that the surrounding interference in 2.4 GHz frequency could have changed between test, thus slightly influencing the results, with a stronger impact for configuration A.

To summarize, given these results, WSO2 CEP is proved to be a fast and capable CEP engine, not only to support SmartLighting project, but an endless number of other scenarios where fast responsiveness is needed. It is capable of processing hundreds of events per second with a sub-second latency, and thus enables the presented solution and its implementation to meet the defined requirements and objectives.

Chapter 6

Conclusions and Future Work

The emergence of the Internet of Things and its exponential evolution allows us to assume that, in a near future, interconnections between devices and even between people and devices will experience a strong change, eventually changing human behaviours. However, there is still a long way to go to explore the full potential. A topic where IoT principles are already being exploited is automation for smart environments. Here intelligent and dynamic automation can be used to help individuals, businesses, and societies on a daily basis. In particular, considering the weight of buildings in the worldwide energy usage, IoT can definitely be used to reduce resource wastage. Furthermore, the same efficient means of control and automation can also be used to provide higher levels of comfort and convenience to building occupants.

Hence, this document presents an overview along with a critical analysis about the applicability of some of IoT's most relevant principles, towards a solution aiming to support the automation of buildings. Furthermore, and as a consequence of the exponential growth the IoT creates, support must be provided for multiple heterogeneous sources of data. The proposed solution addresses this issue with principles from the area of Complex Event Processing, in order to extract the most important information, in a fast and efficient manner.

Using the scenario of the SmartLighting project, the proposed solution and its implementation, was proven to be capable of responding to thousands of events with a sub-second latency. In chapter 5, a performance evaluation of the implementation was undertaken, testing the performance of the CEP engine and measuring latencies in the devices. It was possible to conclude that the solution has very low latencies (around 150 milliseconds), being even the worst case below half a second, and thus still acceptable for time-critical applications like lighting.

Regarding the performance of the CEP engine (WSO2 CEP), it has been shown that it performs well with an efficient management of memory, even when processing hundreds of events per second. Yet, the system load was shown to reach high values. Therefore, it is worth pointing out that as future work an implementation into a cluster of servers should be considered, which the current solution is flexible enough to support.

Another objective of this work consisted in developing a platform that could be easily handled by a generic building manager. This was achieved by deploying a platform with a user-friendly web interface where everything can be managed and configured, acting as an abstraction layer able to hide hundreds of lines of complex code on a CEP engine.

The solution was also designed to be scalable and extensible. Any functionality that does not currently exist, can be appended to the system as a new and pluggable module. In fact, due to the chosen approach, where every rule is represented as a JSON object, even new CEP engines can be integrated. Towards this, an engine controller capable of parsing the rules and adapt them to the new engine must be implemented.

On the opposite side, some issues on the chosen implementation were also found, relating to WSO2 CEP. First, the use of SOAP interface for deploying rules is slow, especially when the system is already too busy handling a large number of events. The problem is that each rule can have dozens or hundreds of elements associated with it, such as receivers, publishers and event streams, and each of them is deployed individually. However, it is the only interface available to manage the system. Furthermore, WSO2 CEP does not handle well exceptions, specifically the ones related to the MQTT clients. When the connections to the MQTT broker fails, an exception occurs but a retry mechanism does not exist and thus the whole system stops working until a reboot of WSO2 CEP is performed.

Hence, and as a target of future work, some optimizations should be made to make a better management of WSO2 CEP using its SOAP interface or, considering it is an open-source platform, implement faster modules to do so. In addition to that, the MQTT client must also be optimized, and means for reconnecting after a defined time, when the connection drops, should also be implemented.

Moreover, due to time constraints, the complete solution could not be implemented, and a module that could provide interesting features, regarding user interactions, was not implemented. The same happened to the deployment of processing engines on the gateways, which is mostly relevant for failure handling. Both features are valuable work for future iterations, even more since the current implementation already enables easy integration of both functionalities.

Security aspects were also not considered in this implementation, even though WSO2 CEP provides means for user authentication, as does the SCoT platform for authentication of each message. Yet, it is an important aspect to have in consideration for this platform, and should be approached in the future, adding encryption of the payload of each MQTT message, and user authentication to the Building Management platform.

Finally, the integration of new systems and provision of new modules to the current implementation is also a good target for future work. In fact, the core aspect of this work was not to provide multiple functionalities, but the means for adding them over time, and thus enrich the platform.

References

[1] A. J. Nelson, O. Rakau, and P. Dörrenberg, “Green buildings - a niche becomes mainstream”, Deutsche Bank Research, 2010.

[2] B. Atanasiu, C. Despret, M. Economidou, J. Maio, I. Nolte, and O. Rapf, “Europe’s buildings under the microscope”, Buildings Performance Institute Europe, 2011.

[3] “Desigo building automation”, Siemens, Tech. Rep., 2014.

[4] University of Aveiro. (2016). Research Day, [Online]. Available: `http://www.ua.pt/researchday/2016/` (accessed on 07/02/2016).

[5] H. Moreira, G. Correia, M. Silva, A. Marques1, J. Barraca, L. Alves, P. Fonseca, and N. Lourenço, “SmartLighting - A platform for intelligent building management”, *INForum*, 2016.

[6] (2016). Leadership in Energy and Environmental Design (LEED), [Online]. Available: `http://www.usgbc.org/leed` (accessed on 07/06/2016).

[7] (2016). Building Research Establishment Environmental Assessment Method (BREEAM), [Online]. Available: `http://www.breeam.com`.

[8] Prompt Automation. (2016). Building Automation, [Online]. Available: `http://www.promptautomation.com/building-automation.html`.

[9] W. Kastner, G. Neugschwandtner, S. Soucek, *et al.*, “Communication systems for building automation and control”, *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, Jun. 2005.

[10] ASME. (2008). Multi-zone automatic temperature control system, [Online]. Available: `https://www.asme.org/about-asme/who-we-are/engineering-history/landmarks/244-multi-zone-automatic-temperature-control` (accessed on 02/12/2016).

[11] “Communication in building automation”, Siemens, Tech. Rep., 2014.

[12] (2016). BACnet, [Online]. Available: `http://www.bacnet.org` (accessed on 07/01/2016).

[13] “Introduction to the LonWorks platform”, Echelon, Tech. Rep., 2009.

[14] (2016). KNX, [Online]. Available: `http://www.knx.org/` (accessed on 07/01/2016).

[15] (2016). DALI, [Online]. Available: http://www.dali-ag.org/ (accessed on 07/01/2016).

[16] (2016). EnOcean Alliance, [Online]. Available: https://www.enocean-alliance.org (accessed on 07/01/2016).

[17] American Society of Heating, Refrigerating and Air-Conditioning Engineers, Ed. (2009). Addendum q to ANSI/ASHRAE Standard 135-2008, [Online]. Available: http://www.bacnet.org/ (accessed on 07/01/2016).

[18] American Society of Heating, Refrigerating and Air-Conditioning Engineers, Ed. (2006). Addendum c to ANSI/ASHRAE Standard 135-2004, [Online]. Available: http://www.bacnet.org/ (accessed on 07/01/2016).

[19] J. M. F. Calado, F. Ferreira, A. L. Osório, and C. S. Pedro, “Building automation interoperability – a review”, *IWSSIP 2010 - 17th International Conference on Systems, Signals and Image Processing*, 2010.

[20] A. Davis. (2011). Open systems - is an open protocol enough?, [Online]. Available: http://www.automatedbuildings.com/news/oct11/articles/andydavis/110925014909andydavis.html (accessed on 02/12/2016).

[21] LonMark International. (2016). What is the LonWorks platform?, [Online]. Available: http://www.lonmark.org/connection/what_is_lon (accessed on 02/12/2016).

[22] H. Merz, T. Hansemann, and C. Hübner, *Building Automation: Communication systems with EIB/KNX, LON und BACnet*, Springer, Ed. 2009.

[23] Echelon Corporation & LONMARK International, “LonMark - SNVT Master List”, Tech. Rep., 2012.

[24] KNX. (2014). What is KNX?, [Online]. Available: http://www.knx.org/knx-en/knx/association/what-is-knx/index.php (accessed on 02/13/2016).

[25] ——, “The worldwide STANDARD for home and building control”, Tech. Rep., 2014.

[26] H. Li, M. Wu, and Y. Zhong, “Development and research of lighting system based on dali”, *ICIEA 2008. 3rd IEEE Conference on Industrial Electronics and Applications*, pp. 1302–1307, 2008.

[27] Y. Zhang, P. Zhou, and M. Wu, “Research on dali and development of master-slave module”, *ICNSC ’06. Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control*, pp. 1106–1110, 2006.

[28] Siemens. (2016). Building Automation Systems, [Online]. Available: http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/building-automation/Pages/building-automation-system.aspx (accessed on 06/02/2016).

[29] Johnson Controls. (2016). Metasys Building Automation System, [Online]. Available: http://www.johnsoncontrols.com/buildings/building-management/building-automation-systems-bas (accessed on 06/02/2016).

[30] Honeywell. (2016). Building Automation Systems, [Online]. Available: http://www.honeywell.com/industries/buildings/building-automation-systems (accessed on 06/02/2016).

[31] (2016). OpenHAB, [Online]. Available: http://www.openhab.org/ (accessed on 06/02/2016).

[32] (2016). Home Assistant, [Online]. Available: https://home-assistant.io/ (accessed on 06/02/2016).

[33] (2015). DomotiGa, [Online]. Available: https://domotiga.nl/projects/domotiga/wiki/Home (accessed on 06/02/2016).

[34] (2016). Domoticz, [Online]. Available: https://domoticz.com/ (accessed on 06/02/2016).

[35] Philips, "Case study: Philips helps create a comfortable, productive and sustainable environment at the edge", Tech. Rep., 2015.

[36] BREEAM, Ed. (2016). The Edge, Amsterdam, [Online]. Available: http://www.breeam.com/index.jsp?id=804 (accessed on 06/02/2016).

[37] D. Evans, "The Internet of Things - how the next evolution of the internet is changing everything", Cisco, Tech. Rep., 2011.

[38] J. Camhi. (2015). BI Intelligence projects 34 billion devices will be connected by 2020. BI Intelligence, Ed., [Online]. Available: http://www.businessinsider.com/bi-intelligence-34-billion-connected-devices-2020-2015-11 (accessed on 05/16/2016).

[39] U.S. Census Bureau. (2015). World population: 1950-2050. International Data Base, Ed., [Online]. Available: http://www.census.gov/population/international/data/idb/worldpopgraph.php (accessed on 05/18/2016).

[40] "The state of broadband 2015", ITU and UNESCO, 2015.

[41] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges", *Ad Hoc Networks*, vol. 10, pp. 1497–1516, 7 Sep. 2012.

[42] K. Ashton. (2009). That 'Internet of Things' thing. RFID Journal, Ed., [Online]. Available: http://www.rfidjournal.com/articles/view?4986 (accessed on 05/16/2016).

[43] "An Introduction to the Internet of Things (IoT)", Lopez Research, 2013.

[44] M. Weiser, "The computer for the 21st century", *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[45] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions", *Future Generation Computer Systems*, vol. 29, pp. 1645–1660, 2013.

[46] J.-P. Vasseur and A. Dunkels, *Interconnecting smart objects with ip*, M. K. Publishers, Ed. 2010.

[47] TinyOS Alliance, "TinyOS 2.1 Adding Threads and Memory Protection to TinyOS", in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, 2008, pp. 413–414.

[48] (2016). Contiki OS, [Online]. Available: `http://www.contiki-os.org/` (accessed on 07/02/2016).

[49] S. Yinbiao, K. Lee, P. Lanctot, F. Jianbin, *et al.*, "Internet of things: Wireless sensor networks", International Electrotechnical Commission, Tech. Rep., 2014.

[50] Z. Chang, O. Alanen, T. Huovinen, T. Nihtila, E. H. Ong, J. Kneckt, and T. Ristaniemi, "Performance analysis of ieee 802.11ac dcf with hidden nodes", *Vehicular Technology Conference (VTC Spring)*, pp. 1–5, 2012.

[51] Bluetooth SIG. (2016). Our history, [Online]. Available: `https://www.bluetooth.com/media/our-history` (accessed on 05/22/2016).

[52] T. I. of Electrical and E. Engineers, "Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans)", *IEEE Standards*, 2003.

[53] N. Salman, I. Rasool, and A. H. Kemp, "Overview of the ieee 802.15.4 standards family for low rate wireless personal area networks", *Wireless Communication Systems (ISWCS)*, pp. 701–705, 2010.

[54] K. Devadiga, "IEEE 802.15.4 and the Internet of Things", *ESG Seminar - Aalto University*, 2011.

[55] M. Salayma, A. Al-Dubai, I. Romdhani, and M. B. Yassein, "Battery aware beacon enabled ieee 802.15.4: An adaptive and cross-layer approach", *COMPUTER SCIENCE AND INFORMATION SYSTEMS (FEDCSIS)*, pp. 1267–1272, 2015.

[56] J. Olsson, "6LoWPAN demystified", Texas Instruments, Tech. Rep., 2014.

[57] TEXAS INSTRUMENTS, Ed. (2016). Overview for 6LoWPAN, [Online]. Available: `http://www.ti.com/lsds/ti/wireless_connectivity/6lowpan/overview.page`.

[58] R. Tadayoni and A. Henten, "From ipv4 to ipv6: Lost in translation?", *Telematics and Informatics*, vol. 33, pp. 650–659, 2016.

[59] D. Airehrour, J. Gutierrez, and S. K. Ray, "Secure routing for internet of things: A survey", *Journal of Network and Computer Applications*, vol. 66, pp. 198–213, 2016.

[60] L. Wadhwa, R. S. Deshpande, and V. Priye, "Extended shortcut tree routing for zigbee based wireless sensor network", *Ad Hoc Networks*, vol. 32, pp. 295–300, 2015.

[61] ZigBee Alliance. (2016). ZigBee IP and 920IP, [Online]. Available: `http://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeeip/` (accessed on 05/23/2016).

[62] Thread Group, "Thread stack fundamentals", Tech. Rep., 2015.

[63] Wi-Fi Alliance. (2016). Wi-Fi HaLow : Low power, long range Wi-Fi, [Online]. Available: `http://www.wi-fi.org/discover-wi-fi/wi-fi-halow` (accessed on 05/23/2016).

[64] HiveMQ. (2016). Enterprise MQTT Broker, [Online]. Available: `http://www.hivemq.com/` (accessed on 05/18/2016).

[65] V. Lampkin, W. T. Leong, L. Olivera, S. Rawat, N. Subrahmanyam, and R. Xiang, "Building smarter planet solutions with mqtt and ibm websphere mq telemetry", in *IBM Redbooks*. 2012.

[66] HiveMQ. (2016). Building a two node high availability mqtt cluster, [Online]. Available: `http://www.hivemq.com/blog/building-a-high-availability-mqtt-cluster` (accessed on 05/19/2016).

[67] M. H. Amaran, N. A. M. Noh, M. S. Rohmad, and H. Hashim, "A comparison of lightweight communication protocols in robotic applications", *Procedia Computer Science*, vol. 76, pp. 400–405, 2015.

[68] D. Thangavel, X. Ma, A. Valera, and H.-X. Tan, "Performance evaluation of MQTT and CoAP via a common middleware", *INTELLIGENT SENSORS, SENSOR NETWORKS AND INFORMATION PROCESSING (ISSNIP)*, pp. 1–6, 2014.

[69] CoAP. (2016). Rfc 7252 constrained application protocol, [Online]. Available: `http://coap.technology/` (accessed on 02/18/2016).

[70] M. Koster, A. Keranen, and J. Jimenez, "Publish-subscribe broker for the constrained application protocol (coap)", IETF Secretariat, Internet-Draft, 2015. [Online]. Available: `https://tools.ietf.org/html/draft-koster-core-coap-pubsub-04`.

[71] (2016). XMPP, [Online]. Available: `https://xmpp.org/` (accessed on 07/04/2016).

[72] P. Waher and Y. DOI, "XEP-0322: Efficient XML Interchange (EXI) Format", XMPP, Tech. Rep., 2015.

[73] (2016). AMQP, [Online]. Available: `https://www.amqp.org/` (accessed on 07/04/2016).

[74] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols and applications", *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 2347–2376, 2015.

[75] (2016). Apache Software Foundation, [Online]. Available: `http://www.apache.org/` (accessed on 07/04/2016).

[76] Apache Software Foundation. (2016). Apache Hadoop, [Online]. Available: `http://hadoop.apache.org/` (accessed on 05/27/2016).

[77] B. Qureshi, Y. Javed, A. Koubâ, M.-F. Sriti, and M. Alajlan, "Performance of a low cost hadoop cluster for image analysis in cloud robotics environment", *Procedia Computer Science*, vol. 82, pp. 90–98, 2016.

[78] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters", Google Research Publications, Tech. Rep., 2004.

[79] Apache Software Foundation. (2016). Apache Spark, [Online]. Available: `http://spark.apache.org` (accessed on 05/27/2016).

[80] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets", in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10, Boston, MA: USENIX Association, 2010, pp. 10–10. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1863103.1863113`.

[81] Sally. (2014). The apache software foundation announces Apache Spark as a Top-Level project. The Apache Software Foundation Blog, Ed., [Online]. Available: https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50 (accessed on 07/04/2016).

[82] ——, (2014). The apache software foundation announces Apache Storm as a Top-Level project. The Apache Software Foundation Blog, Ed., [Online]. Available: https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces64 (accessed on 07/04/2016).

[83] Apache Software Foundation. (2016). Apache Storm, [Online]. Available: http://storm.apache.org (accessed on 05/27/2016).

[84] (2016). Apache Thrift, [Online]. Available: https://thrift.apache.org/ (accessed on 07/04/2016).

[85] Business-Software. (2016). Storm architecture, [Online]. Available: http://www.business-software.com/wp-content/uploads/2014/09/Storm.png (accessed on 07/02/2016).

[86] Apache Software Foundation. (2016). Apache Samza, [Online]. Available: http://samza.apache.org/ (accessed on 07/04/2016).

[87] C. Riccomini. (2013). Apache samza: Linkedin's real-time stream processing framework. LinkedIn, Ed., [Online]. Available: https://engineering.linkedin.com/data-streams/apache-samza-linkedins-real-time-stream-processing-framework (accessed on 07/04/2016).

[88] Sally. (2015). The apache software foundation announces Apache Samza as a Top-Level project. The Apache Software Foundation Blog, Ed., [Online]. Available: https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces71 (accessed on 07/04/2016).

[89] Apache Software Foundation. (2016). Apache Flink, [Online]. Available: http://flink.apache.org (accessed on 05/27/2016).

[90] D. Warneke and O. Kao, "Nephele: Efficient parallel data processing in the cloud", in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, ser. MTAGS '09, Portland, Oregon: ACM, 2009, 8:1–8:10, ISBN: 978-1-60558-714-1. DOI: http://doi.acm.org/10.1145/1646468.1646476.

[91] Sally. (2015). The apache software foundation announces Apache Flink as a Top-Level project. The Apache Software Foundation Blog, Ed., [Online]. Available: https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces69 (accessed on 07/04/2016).

[92] ——, (2016). The apache software foundation announces Apache Apex as a Top-Level project. T. A. S. F. Blog, Ed., [Online]. Available: https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces90 (accessed on 07/04/2016).

[93] (2016). Apache Apex, [Online]. Available: https://apex.apache.org/ (accessed on 07/04/2016).

[94] J. Fanelli. (2015). Datatorrent open sources datatorrent rts, industry's only enterprise-grade unified stream and batch processing platform. DataTorrent, Ed., [Online]. Available: https://www.datatorrent.com/press-releases/datatorrent-open-sources-datatorrent-rts-industrys-only-enterprise-grade-unified-stream-and-batch-processing-platform/ (accessed on 07/04/2016).

[95] (2016). Pulsar, [Online]. Available: http://gopulsar.io/ (accessed on 07/04/2016).

[96] S. Murthy, T. Ng, B. Avalani, X. Wang, K. Wang, and A. Gangadharan, "Pulsar - Real-time Analytics at Scale", eBay, Inc, Tech. Rep., 2015.

[97] N. Marz. (2011). A storm is coming: More details and plans for release. Twitter, Ed., [Online]. Available: https://blog.twitter.com/2011/a-storhttps://blog.twitter.com/2011/a-storm-is-coming-more-details-and-plans-for-releasem-is-coming-more-details-and-plans-for-release (accessed on 05/27/2016).

[98] (2016). Heron, [Online]. Available: http://twitter.github.io/heron/ (accessed on 07/02/2016).

[99] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale", *SIGMOD '15*, pp. 239–250, 2015.

[100] K. Ramasamy. (2016). Open sourcing twitter heron. Twitter, Ed., [Online]. Available: https://blog.twitter.com/2016/open-sourcing-twitter-heron (accessed on 05/27/2016).

[101] D. C. Luckham, *The power of events: An introduction to complex event processing in distributed enterprise systems*, A.-W. Professional, Ed. 2001.

[102] EsperTech, Ed. (2016). Esper History, [Online]. Available: http://www.espertech.com/esper (accessed on 05/29/2016).

[103] (2016). Drools, [Online]. Available: http://www.drools.org/ (accessed on 07/02/2016).

[104] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, and V. Nanayakkara, "Siddhi: A second look at complex event processing architectures", in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, ACM, 2011, pp. 43–50.

[105] WSO2. (2016). WSO2 Complex Event Processor, [Online]. Available: http://wso2.com/products/complex-event-processor/ (accessed on 05/29/2016).

[106] S. Eshan. (2015). WSO2 CEP 4.0.0 in distributed mode. Sajith Ravindra's Blog, Ed., [Online]. Available: http://sajithr.blogspot.pt/2015/09/wso2-cep-400-in-distributed-mode.html (accessed on 05/29/2016).

[107] J. Green, CTO Data Virtualization, "IoT Reference Model", Cisco, Tech. Rep., 2014.

[108] M. Antunes, J. P. Barraca, D. Gomes, P. Oliveira, and R. L. Aguiar, "Smart Cloud of Things: An evolved IoT platform for telco providers", *Journal of Ambientcom*, vol. 1, pp. 1–24, 2015.

[109] (2016). JBoss BRMS, [Online]. Available: http://www.jboss.org/products/brms/overview/ (accessed on 07/03/2016).

[110] Buildings Datasets, Ed. (2015). Start a project with real datasets from the department of energy, [Online]. Available: https://trynthink.github.io/buildingsdatasets/ (accessed on 06/28/2016).

[111] Emparador. (2016). Motion detector, [Online]. Available: https://play.google.com/store/apps/details?id=org.motion.detector (accessed on 06/28/2016).

[112] Crafty Apps EU. (2016). Tasker, [Online]. Available: https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm (accessed on 06/28/2016).

[113] I. Harvey. (2016). Bluepy, [Online]. Available: https://github.com/IanHarvey/bluepy (accessed on 06/28/2016).

Appendix A: Example modules implementation

```
class timeGT(Filter):
        _type = ’time_gt’

        time = models.CharField(max_length=8)

        def __str__(self):
                return ’Time Greater than %s’% self.time

        def get_data(self):
                d = super(timeGT, self).get_data()
                d[’value’] = self.time
                return d

        def get_dict(self):
                d = dict()
                d[’type’] = timeGT._type
                d[’time’] = self.time
                return d

        @staticmethod
        def descript():
                d = dict()
                d[’name’] = ’Time Greater than’
                d[’description’] = ’Filters events if current time’ \
                                ’is Greater than specified time’
                d[’type’] = timeGT._type
                d[’parameters’] = {
                        ’time’: {
                                ’label’: ’Time’,
                                ’type’: ’charfield’,
                                ’max_length’: 8,
                                ’required’ : True,
                                ’placeholder’: ’HH:MM:SS’
                        }
                }
                return d
```

Snippet 9: “Time Greater Than” module for Building Management platform

```
class timeGT(Filter):
        _type = 'time_gt'
        def __init__(self, **kwargs):
                super(timeGT, self).__init__()
                self.time = kwargs['time']

        def get_filter(self):
                return 'time_gt(time:currentTimestamp(), "%s")'%self.time

        def has_function(self):
                return True

        def get_function(self):
                return 'define function time_gt[JavaScript] return bool {\n' \
                                '\tvar str1 = data[0];\n' \
                                '\tvar str2 = str1.substring(0, 11) + data[1];\n' \
                                '\tvar dt1 = new Date( str1.split(\' \').join(\'T\') );\n' \
                                '\tvar dt2 = new Date( str2.split(\' \').join(\'T\') );\n' \
                                '\treturn dt1>dt2;\n' \
                                '};\n\n'
```

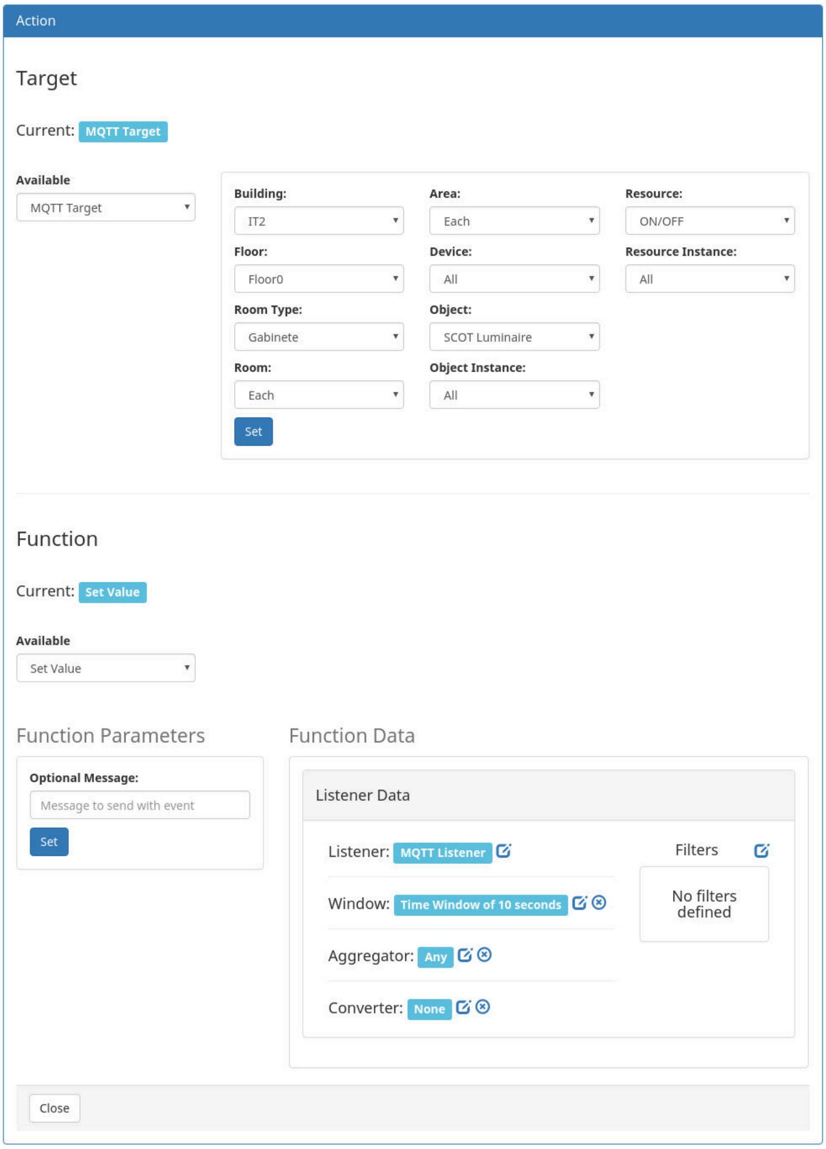
Snippet 10: "Time Greater Than" implementation on WSO2 CEP engine controller.

Appendix B: Example Rule

This appendix aims to show the complexity hidden by the implemented platform. First, it presents two figures, along with a brief description, of an example action defined in the platform's web interface, thus showing how it is organized. Second, it presents two flux diagrams of rules defined in the WSO2 CEP engine for the test scenario considered in chapter 5.

Figure 1 shows an example of an action defined in the user-friendly interface, which is organized in two main parts: target definition on top, and function definition below it. A target can be chosen using the drop-down box at the left where an MQTT target is selected, and then in the right side its parameters must also be chosen. Similarly, the function definition is also selected using a drop-down box, and its parameters defined below it. Yet, there is an additional part regarding the function data, where a listener, a window, an aggregator and a converter may be defined, along with filters for making selection of data coming from the defined listener. All the elements that can be defined in the function data part follow an organization similar to the target definition, except for filters which have more complex combinations, and thus are specified as shown in figure 2.

WSO2 CEP automatically generates flux diagrams for each rule that is defined. From these, it is possible to understand the correlation of data from multiple sources, by checking the streams from which each query reads data from. Figure 3 shows the flux diagram for a complete rule used in configuration A of the test scenario considered, while figure 4 shows only a portion of a much bigger and complex rule, used in configuration B.



Action

Target

Current: MQTT Target

Available: MQTT Target

Building: IT2

Area: Each

Resource: ON/OFF

Floor: Floor0

Device: All

Resource Instance: All

Room Type: Gabinete

Object: SCOT Luminaire

Room: Each

Object Instance: All

Set

Function

Current: Set Value

Available: Set Value

Function Parameters

Optional Message: Message to send with event

Set

Function Data

Listener Data

Listener: MQTT Listener

Window: Time Window of 10 seconds

Aggregator: Any

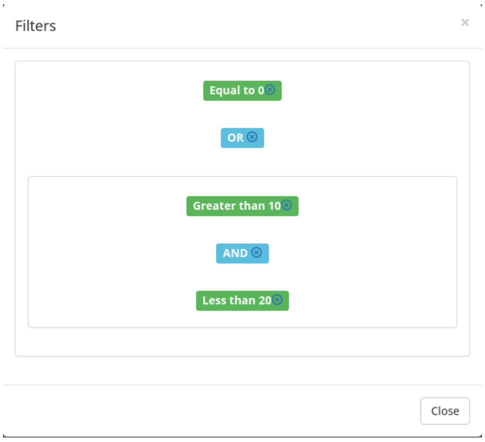
Converter: None

Filters

No filters defined

Close

Figure 1: Example action configured at Web interface.



Filters

Equal to 0

OR

Greater than 10

AND

Less than 20

Close

Figure 2: Example filtering configured at Web interface.

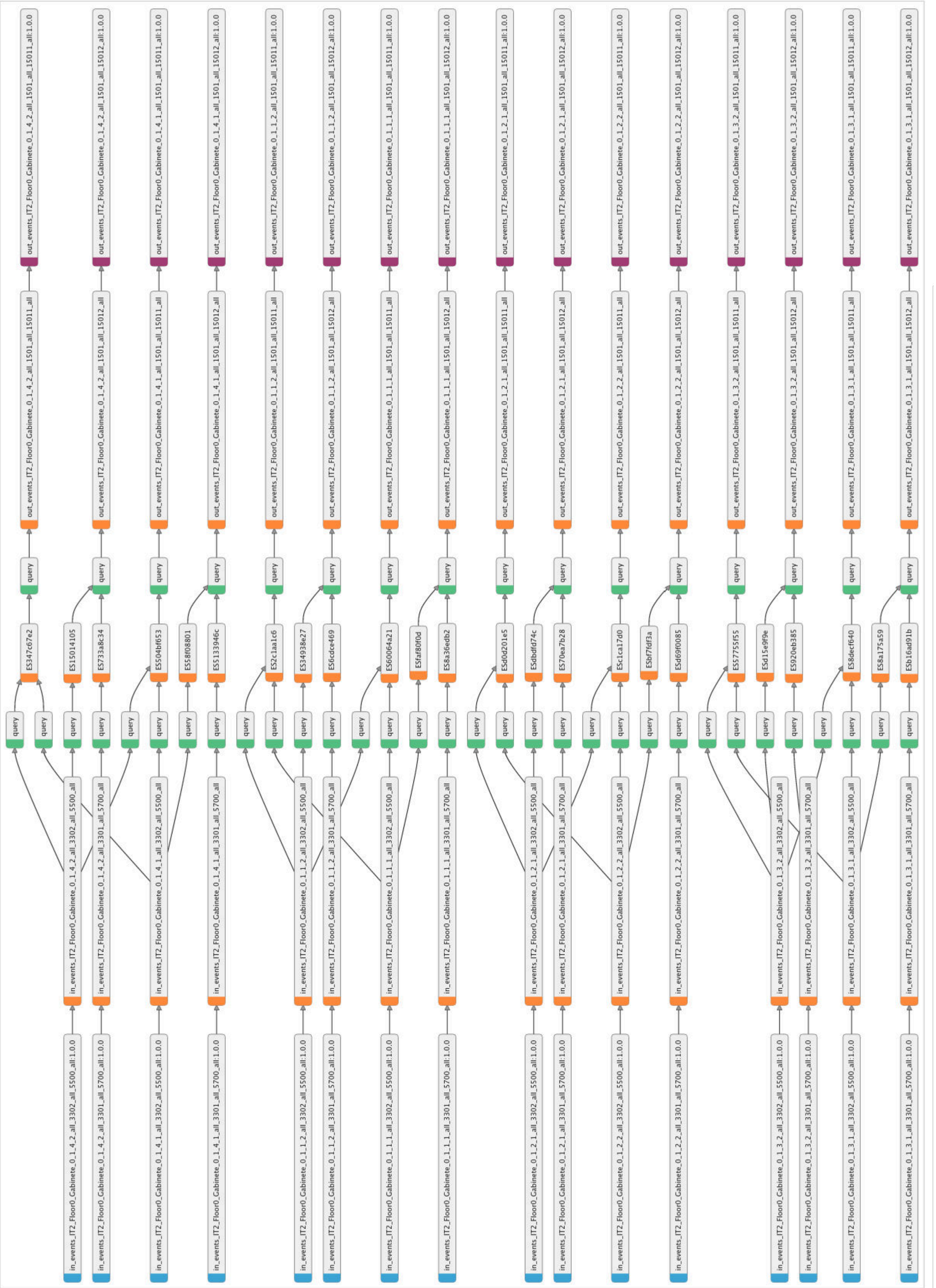

out_events_IT2_Floor0_Gabinete_0_1_4_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_4_2_all_1501_all_15011_all
query
ES347c67e2
query
query
ES15014105
query
in_events_IT2_Floor0_Gabinete_0_1_4_2_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_4_2_all_3302_all_5500_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_4_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_4_2_all_1501_all_15012_all
query
ES733a8c34
query
in_events_IT2_Floor0_Gabinete_0_1_4_2_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_4_2_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_4_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_4_1_all_1501_all_15011_all
query
ES504bf653
query
in_events_IT2_Floor0_Gabinete_0_1_4_1_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_4_1_all_3302_all_5500_all:1.0.0
ES58f08801
query
out_events_IT2_Floor0_Gabinete_0_1_4_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_4_1_all_1501_all_15012_all
query
ES5133946c
query
in_events_IT2_Floor0_Gabinete_0_1_4_1_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_4_1_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_1_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_1_2_all_1501_all_15011_all
query
ES2c1aa1c6
query
ES34938e27
query
in_events_IT2_Floor0_Gabinete_0_1_1_2_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_1_2_all_3302_all_5500_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_1_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_1_2_all_1501_all_15012_all
query
ES6cdce469
query
in_events_IT2_Floor0_Gabinete_0_1_1_2_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_1_2_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_1_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_1_1_all_1501_all_15011_all
query
ES60064a21
query
in_events_IT2_Floor0_Gabinete_0_1_1_1_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_1_1_all_3302_all_5500_all:1.0.0
ESfaf80f0d
query
out_events_IT2_Floor0_Gabinete_0_1_1_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_1_1_all_1501_all_15012_all
query
ES8a36edb2
query
in_events_IT2_Floor0_Gabinete_0_1_1_1_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_1_1_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_2_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_2_1_all_1501_all_15011_all
query
ESd0d201e5
query
ESdbdfd74c
query
in_events_IT2_Floor0_Gabinete_0_1_2_1_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_2_1_all_3302_all_5500_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_2_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_2_1_all_1501_all_15012_all
query
ES70ea7b28
query
in_events_IT2_Floor0_Gabinete_0_1_2_1_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_2_1_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_2_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_2_2_all_1501_all_15011_all
query
ESc1ca17d0
query
in_events_IT2_Floor0_Gabinete_0_1_2_2_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_2_2_all_3302_all_5500_all:1.0.0
ESbf7fdf3a
query
out_events_IT2_Floor0_Gabinete_0_1_2_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_2_2_all_1501_all_15012_all
query
ESd69f0085
query
in_events_IT2_Floor0_Gabinete_0_1_2_2_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_2_2_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_3_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_3_2_all_1501_all_15011_all
query
ES57755f55
query
ESd15e9f9e
query
in_events_IT2_Floor0_Gabinete_0_1_3_2_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_3_2_all_3302_all_5500_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_3_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_3_2_all_1501_all_15012_all
query
ES920eb385
query
in_events_IT2_Floor0_Gabinete_0_1_3_2_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_3_2_all_3301_all_5700_all:1.0.0
query
out_events_IT2_Floor0_Gabinete_0_1_3_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_3_1_all_1501_all_15011_all
query
ES8decf640
query
in_events_IT2_Floor0_Gabinete_0_1_3_1_all_3302_all_5500_all
in_events_IT2_Floor0_Gabinete_0_1_3_1_all_3302_all_5500_all:1.0.0
ES8a175a59
query
out_events_IT2_Floor0_Gabinete_0_1_3_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Gabinete_0_1_3_1_all_1501_all_15012_all
query
ESb16ad91b
query
in_events_IT2_Floor0_Gabinete_0_1_3_1_all_3301_all_5700_all
in_events_IT2_Floor0_Gabinete_0_1_3_1_all_3301_all_5700_all:1.0.0

Figure 3: Full Flux Diagram of a rule used in configuration A.

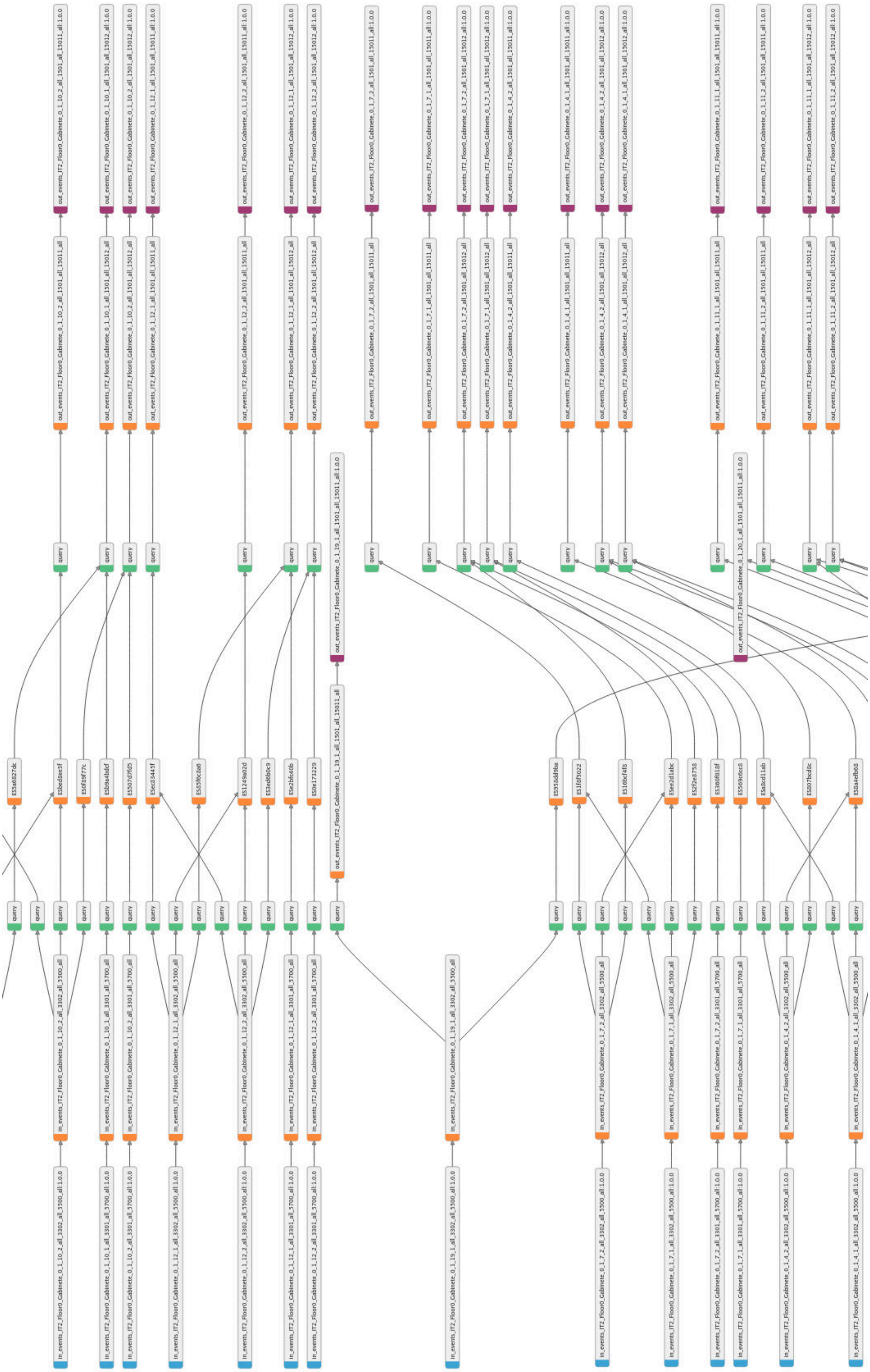

in_events_IT2_Floor0_Cabinete_0_1_10_2_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_10_1_all_3301_all_5700_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_10_2_all_3301_all_5700_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_12_1_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_12_2_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_12_1_all_3301_all_5700_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_12_2_all_3301_all_5700_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_19_1_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_7_2_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_7_1_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_7_2_all_3301_all_5700_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_7_1_all_3301_all_5700_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_4_2_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_4_1_all_3302_all_5500_all:1.0.0
in_events_IT2_Floor0_Cabinete_0_1_10_2_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_10_1_all_3301_all_5700_all
in_events_IT2_Floor0_Cabinete_0_1_10_2_all_3301_all_5700_all
in_events_IT2_Floor0_Cabinete_0_1_12_1_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_12_2_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_12_1_all_3301_all_5700_all
in_events_IT2_Floor0_Cabinete_0_1_12_2_all_3301_all_5700_all
in_events_IT2_Floor0_Cabinete_0_1_19_1_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_7_2_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_7_1_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_7_2_all_3301_all_5700_all
in_events_IT2_Floor0_Cabinete_0_1_7_1_all_3301_all_5700_all
in_events_IT2_Floor0_Cabinete_0_1_4_2_all_3302_all_5500_all
in_events_IT2_Floor0_Cabinete_0_1_4_1_all_3302_all_5500_all
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
ES5a6827dc
ESbed8ee5f
ES0f89f77c
ESb9a4bdcf
ES507d7fd5
ESec83445f
ES85f0c8a6
ES1249a02d
ES3ed6b0c9
ESe2bfc40b
ES0e173229
ES958dd9ba
ES1f8f5022
ES16bcf4f8
ESee2d1abc
ES2f2e8758
ES360f018f
ES569c6cc8
ESa5cd11ab
ES807bcd8c
ES8a4effb68
out_events_IT2_Floor0_Cabinete_0_1_19_1_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_19_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_20_1_all_1501_all_15011_all:1.0.0
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
query
out_events_IT2_Floor0_Cabinete_0_1_10_2_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_10_1_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_10_2_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_12_1_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_12_2_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_12_1_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_12_2_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_7_2_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_7_1_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_7_2_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_7_1_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_4_2_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_4_1_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_4_2_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_4_1_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_11_1_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_11_2_all_1501_all_15011_all
out_events_IT2_Floor0_Cabinete_0_1_11_1_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_11_2_all_1501_all_15012_all
out_events_IT2_Floor0_Cabinete_0_1_10_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_10_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_10_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_12_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_12_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_12_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_12_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_7_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_7_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_7_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_7_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_4_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_4_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_4_2_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_4_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_11_1_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_11_2_all_1501_all_15011_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_11_1_all_1501_all_15012_all:1.0.0
out_events_IT2_Floor0_Cabinete_0_1_11_2_all_1501_all_15012_all:1.0.0

Figure 4: Part of Flux Diagram of a rule used in configuration B.