PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
GRADUATE PROGRAM IN COMPUTER SCIENCE

DECISION SUPPORT IOT FRAMEWORK: DEVICE DISCOVERY AND STREAM ANALYTICS

WILLIAN TESSARO LUNARDI

Dissertation presented as partial requirement for obtaining the degree of Master in Computer Science at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Dr. Sabrina Marczak
Co-Advisor: Dr. Leonardo Amaral

Porto Alegre
2016

Dados Internacionais de Catalogação na Publicação (CIP)

L961d Lunardi, Willian Tessaro

Decision support IoT framework : device discovery and stream analytics / Willian Tessaro Lunardi. – 2016.
71 p.

Dissertação (Mestrado) – Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul.
Orientador: Prof. Dr. Sabrina Marczak
Co-orientador: Dr. Leonardo Amaral

1. Internet das Coisas. 2. Sistemas de Suporte de Decisão. 3. Processamento de Eventos (Informática). 4. Informática. I. Marczak, Sabrina. II. Amaral, Leonardo. III. Título.

CDD 23 ed. 006.22

Salete Maria Sartori CRB 10/1363
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "*Decision Support IoT Framework: Device Discovery and Stream Analytics*" apresentada por Willian Tessaro Lunardi como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 23 de março de 2016 pela Comissão Examinadora:

Profa. Dra. Sabrina dos Santos Marczak– Orientadora — PPGCC/PUCRS

Prof. Dr. Leonardo Albernaz Amaral – Coorientador — PPGEE/PUCRS

Prof. Dr. Fabiano Passuelo Hessel – — PPGCC/PUCRS

Prof. Dr. Eric Rohmer - — UNICAMP

Homologada em 23/06/2016, conforme Ata No. 013 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

AGRADECIMENTOS

Gostaria de agradecer especialmente a minha família. Palavras não podem expressar o quão grato sou a vocês, Milton e Márcia, por todos os sacrifícios em meu nome, por terem buscado despertar a minha curiosidade pelo conhecimento. Agradeço a minha irmã Stefani pelo companheirismo, por todos os momentos de descontração e felicidade. Meus sinceros agradecimentos à minha amada esposa Luciana, que me acompanhou durante toda a jornada, por ter estado ao meu lado nos momentos em que não havia ninguém para responder as minhas perguntas. Também agradeço-lhes por sempre terem me incentivado a estudar e correr atrás dos meus sonhos, sempre apoiando as minhas decisões.

Gostaria de expressar o meu apreço especial à minha orientadora Prof. Dr. Sabrina Marczak, por ter sido uma tremenda mentora para mim. Eu gostaria de agradecer-lhe por incentivar a pesquisa e por me permitir a crescer como pesquisador. Agradeço também ao meu co-orientador, Leonardo Amaral.

Agradeço aos meus amigos e colegas de mestrado, em especial Ramão e Everton pelo companheirismo e parceria em todas as etapas do curso.

DECISION SUPPORT IOT FRAMEWORK: DEVICE DISCOVERY AND STREAM ANALYTICS

RESUMO

Durante os últimos anos, com o rápido desenvolvimento e proliferação da Internet das Coisas (IoT), muitas áreas de aplicação começaram a explorar este novo paradigma de computação. O número de dispositivos computacionais ativos tem crescido em um ritmo acelerado ao redor do mundo. Consequentemente, um mecanismo para lidar com estes diferentes dispositivos tornou-se necessário. Middlewares para a IoT têm sido desenvolvidos tanto em ambientes industriais como de pesquisa para suprir esta necessidade, no entanto, a descoberta e a seleção de dispositivos, bem como o suporte a tomada de decisão baseada no fluxo de dados destes dispositivos continuam sendo um desafio crítico. Neste trabalho apresentamos o Decision Support IoT Framework, composto pelo sistema COBASEN, um motor de busca de dispositivos da IoT, e o sistema DMS, o qual atua sobre dados de dispositivo em movimento, extraindo informações valiosas para dar suporte a tomada de decisões. O sistema COBASEN opera com base nas características textuais dos perfis dos dispositivos. O sistema DMS utiliza processamento de eventos complexos para analisar e reagir sobre os dados de fluxo contínuo, permitindo, por exemplo, disparar um alerta quando um erro ou condição específica aparece no fluxo de dados do dispositivo. O objetivo principal deste trabalho é destacar a importância de um motor de busca de dispositivos para a Internet das Coisas e um sistema de apoio à tomada de decisão baseado na análise de fluxo contínuo dos dispositivos IoT. Foi desenvolvido dois sistemas que implementam conceitos COBASEN e DMS. No entanto, em testes preliminares, realizado uma avaliação funcional de ambos os sistemas em termos de desempenho. Resultados iniciais sugerem que o Decision Support IoT Framework fornece abordagens importantes que facilitam o desenvolvimento de aplicações da Internet das Coisas, podendo executar funções essenciais para melhorar os processos de ambientes que fazem uso deste paradigma.

Palavras Chave: Internet das Coisas, Descobrimento de devices, Suporte à decisão, Processamento de Eventos.

DECISION SUPPORT IOT FRAMEWORK: DEVICE DISCOVERY AND STREAM ANALYTICS

ABSTRACT

During the past few years, with the fast development and proliferation of the Internet of Things (IoT), many application areas have started to exploit this new computing paradigm. The number of active computing devices has been growing at a rapid pace in IoT environments around the world. Consequently, a mechanism to deal with this different devices has become necessary. Middleware systems solutions for IoT have been developed in both research and industrial environments to supply this need. However, device discovery and selection, as well decision analytics remain a critical challenge. In this work we present the Decision Support IoT Framework composed of COBASEN, an IoT search engine to address the research challenge regarding the discovery and selection of IoT devices when large number of devices with overlapping and sometimes redundant functionality are available in IoT middleware systems, and DMS, which allows to set up analytic computations on device data when it is still in motion, extracting valuable information from it for decision management. COBASEN operates based on textual characteristics of devices. The DMS uses Complex Event Processing to analyze and react over streaming data, allowing for example, to triggers an alert when a specific error or condition appears in the stream. The main goal of this work is to highlight the importance of an IoT search engine for devices and a decision support system for stream analytics in the IoT paradigm. We developed two systems that implements COBASEN and DMS concepts. However, for preliminarily tests, we made a functional evaluation of both systems in terms of performance. Our initial findings suggest that the Decision Support IoT Framework provides important approaches that facilitate the development of IoT applications, which may perform essential roles to improve IoT processes.

Keywords: Internet of Things, Device Discovery, Decision Analytics, Stream Processing, Event Processing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

API – Application Programming Interface

CEP – Complex Event Processing

IDL – Interface Description Language

IOT – Internet of Things

M2M – Machine to Machine

PAAS – Platform as a Service

PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul

REST – Representational State Transfer

RFID – Radio-frequency Identification System

SAAS – Software-as-a-Service

SE – Search Engine

SN – Sensor Networks

SOA – Service-Oriented Architecture

SOAP – Simple Object Access Protocol

URI – Uniform Resource Identifier

WOT – Web of Things

WS – Web Services

XML – eXtensible Markup Language

CONTENTS

1. INTRODUCTION

The term Internet of Things (IoT) was coined in 1998 [6] and defined as the computing paradigm that allows people and things to be connected Anytime, Anyplace, with Anything and with Anyone, ideally using Any path/network and Any service [54]. In this sense, there are current market statistics and predictions that demonstrate a rapid growth in computing device deployments related to IoT environments. By 2020, it is estimated that there will be 50 to 100 billion IoT devices connected to the Internet [51].

Based on these statistics, the old frontiers between the digital and physical world will be torn down. According to experts from industry and research, the introduction of the IoT into the manufacturing environment is ushering in a fourth industrial revolution called Industrial Internet of Things (IIoT or Industry 4.0). The adoption of several computing devices (RFID, sensors, and actuators) in industrial environments has improved manufacturing processes and their outcomes (e.g. cost savings, greater efficiency and speed, smarter products and services).

These statistics and facts imply that we will be faced with a vast amount of IoT devices, which, when properly used, will add more value to environments. In this sense, the challenge and time-consuming task is to select and use the appropriate devices when there are a large amount of available devices to choose from. Another critical challenge is derived from the usage of devices, which represents vast amounts of data flowing (streaming data) from the perception layer to the application layer. Therefore it is not only to find and use devices, but is also what we do and how long it take to react to it that truly matters.

IoT Middleware systems solutions have been developed in both research and industrial environments to supply this need. However, device discovery and selection, as well stream analytics remains a critical challenge [43].

1.1 Motivation

According to the study provided by the GNS Team [22], there are over 12,000 working and useful Web services on the Web. Even in such condition, making a choice among the available alternatives (depending on context characteristics) has become a challenging issue. Perera et al. [42] reinforce that "similarities" strengthen the argument that device selection is an important challenge at the same level of complexity as Web services. On the other hand, "differences" demonstrate that device selection will become a much more complex challenge over the coming decade due to the scale of IoT environments.

That is, end users, such as software engineers that are in charge of select heterogeneous devices in order to contextualize virtual environments and define rules among them, are not aware of domain modeling and middleware specification patterns (to define what and how they want to interact with things/devices). Besides, end users may not have enough knowledge to perform such

task without devoting time to understand the middleware patterns or to engage others which were responsible for modeling the domain and register the devices.

Let's consider a scenario to reinforce our arguments and to strength the necessity to provide an easily understandable manner for end users and IoT applications software engineers to search, select, and use IoT devices through middlewares. It will also help to understand the related challenges more clearly.

Spontaneous heating of carbonaceous materials is a common phenomenon, occurring in particular when large quantities of materials are stored for extended periods. In large-scale silo storage of wood fuel pellets self heating has become a serious problem, sometimes causing self-ignition [11]. The challenge can be addressed by deploying devices as sensors and actuator in the environment. Moreover, to gather data of these devices is essential to understand what is happening in the silo. For example, monitoring the temperature of a silo can be useful for identifying outbreaks fermentation. However, a manual temperature monitoring a silo is very difficult and sometimes impossible due to its possible physical size. The utilization of smart monitoring devices (e.g. temperature, air humidity) can be useful to solve this problem. As well, to make this possible, the data stream provided by the devices need to be constantly monitored by algorithms that enable detection and combination of events and patterns making it possible to take action.

To monitor the temperature of a single silo, an average of 124 monitoring devices are needed [29]. Another perspective can be a scenario of an industrial environment that contains 10 storage silos with several devices monitoring the temperature of each silo. The number of possible devices to cover this environment can easily reach more than 1200. The challenge in this case is that after the deployment of the infrastructure of the IoT middleware and physical and logical arrangement of the devices, end users or IoT application software engineers (who are not aware of the environment disposition of the devices) face several difficulties regarding how to search, select, and use the data provided by the monitoring devices.

The challenge is to develop a system that is able to search devices available in IoT middleware and provide to software engineers an easy and transparent method to search, select and use them. Another critical challenge is using device data when it is still in motion, extracting valuable information from it through Complex Event Processing (CEP), which is better explained in the Section 2.3.3, and allow decision making.

1.2 Research Goal

By analyzing others IoT middleware systems (e.g., [15][22] and those described in Chapter 3), we found that they do not provide search methods for device discovery and selection and/or do not provide methods to allow decision making based on device streaming. Trying to fill this gap, we propose in this work a software framework composed of a search engine for device discovery and a decision management system. The framework allows IoT middleware users engineers who are

not aware of devices domain and middleware patterns, to be able to discover, search, select, and define business rules for decision management, and interact with devices in order to use it in their applications.

1.3 Text Organization

The remainder of this work is organized as follows: Chapter 2 consists of relevant literature concepts and definitions. Chapter 3 presents studies that have relevance related to the proposed work. Chapter 4 presents in details the objectives and the system framework. Chapter 5 provides implementation and evaluation details including tools, software platforms, data sets used in this work and our preliminary results. Chapter 6 presents the final considerations and concludes this work with prospects of future work.

2. THEORETICAL BACKGROUND

In this Chapter, Section 2.1 presents the IoT paradigm and its architecture overview. Section 2.2 presents the concepts regarding middleware technology for IoT systems. The reference middleware platform – COMPaaS [3] is presented in the Section 2.2.2. Section 2.3 presents IoT middlewares systems challenges and perspectives.

2.1 Internet of Things (IoT)

Internet of Things is an emerging computing paradigm that combines aspects and technologies coming from different areas of human knowledge in order to provide a computing environment able to autonomously cope with users needs and requirements. Ubiquitous and pervasive computing, sensing and communication technologies, operating systems, mobile computing, big data management, and embedded systems are examples of research fields that have been merged together to form the IoT ecosystem wherein representations of real and digital worlds meet each other in order to keep a ubiquitous environment in constant interaction [12].

In IoT ecosystems, computing interactions are driven by smart objects, which are system entities considered the main building blocks of the IoT environment. By putting intelligence into everyday objects (i.e., dedicated embedded systems into everyday physical devices), these devices are turned into smart objects able not only to collect information from the environment and interact/control the things of the physical world, but also to be interconnected to each other through the Internet to autonomously exchange pervasive data and information. The expected huge number of interconnected devices and the significant amount of available data, open new opportunities to create smart services that will bring tangible benefits to the society, environment, economy, and individual citizens [12].

IoT considers the pervasive presence of a variety of smart objects interacting and cooperating in the physical environment through available ubiquitous services. Thus, the goal of IoT is to enable things to be interconnected at anytime, anyplace, with anything and anyone, ideally using any path/network and any source [44].

To make the idea of IoT more clear, let's consider the "city ecosystem" as an example of how the city of the future (i.e., the Smart City) will look like in the coming years [12]. Indeed, a smart city is a kind of city that should be able to operate simultaneously on two representation levels, physical and virtual, respectively. These abstractions should imply in the provision of intelligent solutions that ensure efficiency at multiple levels, aiming basically to: (i) a more aware and optimized usage of the resources of the city, (ii) a minimization of environmental impact (e.g., by reducing CO2 emissions), and (iii) an increase in the life quality of citizens in terms of safety, health, and wellness. This smart capability is desired due to the fact that, today, half of the global population is concentrated in the cities, and hence, is increasingly consuming the city's resources (e.g., light, water)

everyday. Besides, quality, sustainability, and security are crucial requirements and unavoidable issues for the city.

A smart city should provide autonomous management of its public services (e.g., transport, energy, lighting, waste management, health, and entertainment) through the widespread adoption of information and communication technologies. Such technologies are the basis for the provision of a logical/virtual infrastructure that should be able to control and coordinate the physical infrastructure of the city in order to adapt the city services to the actual citizen needs, while reducing waste and making the city more sustainable [53]. In this way, IoT is essential in this transition process since it has helped conventional cities to be turned into smart cities where traditional and more emerging sectors such as mobility, buildings, energy, living, and governance will also benefit from this transition. For example, smart mobility services can be created to provide effective tools to the citizens to accurately plan their journeys with public/private transportation.

Figure 2.1 provides a schematic representation of a smart city ecosystem. In this perspective, the city will be equipped with physical devices or things (e.g., network of sensors, cameras, speakers, smart meters, and thermostats) that will collect information of the environment. The gathered information, the so-called "Big Data" (the name refers to its large volume and its heterogeneity in terms of content and data representation), will not only be used for the improvement of just a single city service/application, but it will also be shared among different services into the smart city ecosystem [19]. In this sense, a common platform for the operational management of city elements – a sort of City Operating System [12], will be responsible for managing, storing, analyzing, processing, and forwarding the city data anywhere and anytime, to anyone and anything, helping the city to improve and adapting the city services to users needs. This management layer, no longer vertical but horizontal, will ensure interoperability, coordination, and optimization of individual services/applications through the analysis of heterogeneous information flows. Citizens/authorities will access the services offered by the platform through their applications, will consume them and will actively participate by creating additional content (i.e., new data or applications) that will be provided as further input to the City Operating System.

From the above description of the future Smart City, it is clear that, for the IoT vision to successfully emerge, a number of different technical challenges need to be faced and solved. These challenges range from hardware devices, systems and platform architectures, communication technology, devices and data discovery, data processing and network management, security and privacy, just to mention a few. Furthermore, there is also a significant lack of a "standard middleware system platform" able to cope with the horizontal abstraction of the common management layer described above. This middleware layer should not only take care of the elements and services of the city, but also be aware of most technical challenges identified above.

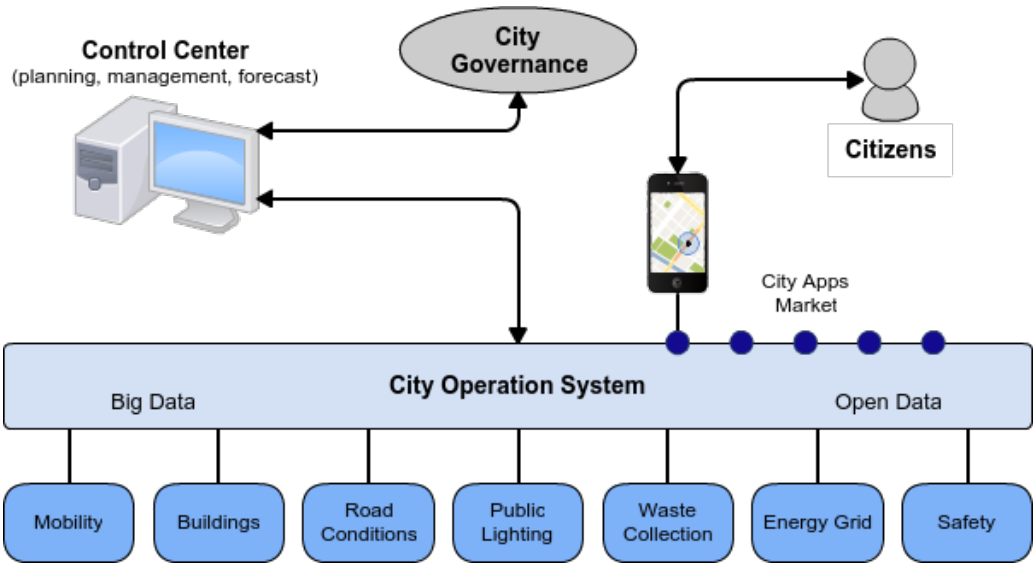

Figure 2.1 – Schematic representation of a Smart City ecosystem [24].

2.1.1 Horizontal Architecture Approach for IoT Systems

A systemic implementation of IoT ecosystems is usually based on a layered architecture style [10], which can range from data acquisition layer (i.e., Perception layer) at the bottom, to application layer at the top (see Figure 2.2). In this kind of architecture, layers from the bottom usually contribute to devices integration and data capturing, while layers from the top are responsible for data distribution and utilization by IoT applications.

The common architectural approach largely used by current IoT systems is a vertical strategy where each application is built on its proprietary infrastructure and dedicated physical devices. In this approach, similar applications do not share any feature of the IoT infrastructure (e.g., managing services and network), resulting in unnecessary redundancy and increase of costs (i.e., financial and computational costs). As explained in the smart city example, this totally vertical approach should be overtaken by a more flexible and horizontal approach, where a common operational platform is able to manage the network and the application services, and abstracts across a diverse range of data sources to enable applications to work properly.

As shown by Figure 2.2, through a horizontal IoT middleware approach, applications no longer work in isolation, and share infrastructure, environment, and network elements by means of a common service platform (i.e., the IoT middleware platform) that orchestrates on their behalf. Figure 2.2 also shows the three different layers (or interaction phases) in which the cyber-physical world interactions should take place. Specifically, these are: (i) perception layer, (ii) transportation layer, and (iii) application layer (i.e., process, management and utilization phases). Each layer is characterized by different interacting technologies and protocols and has different purposes and functions as discussed below.

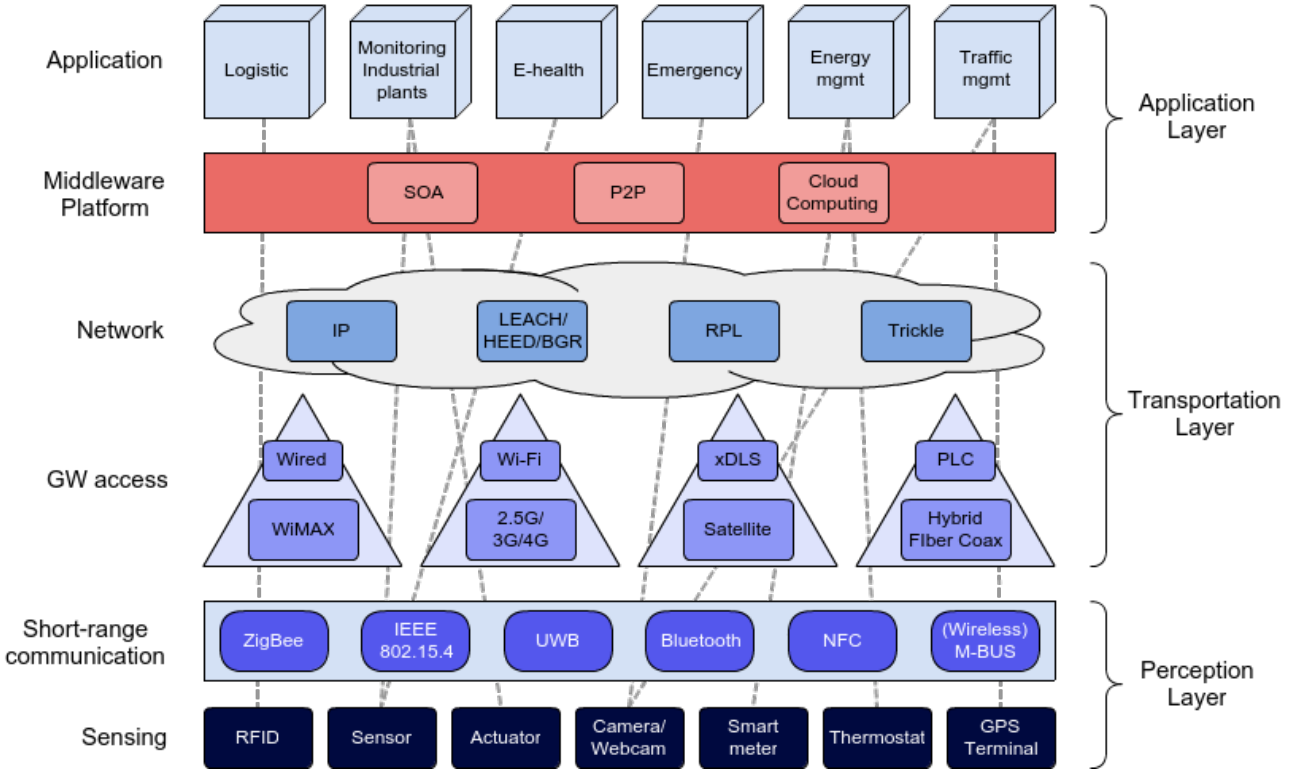

Figure 2.2 – IoT systems architecture overview.

- **Perception layer:** it refers to procedures for sensing the physical environment, collecting real-time data, and reconstructing a general perception of it in the virtual world (i.e., in the system logical domain). Technologies such as Radio-Frequency IDentification (RFID) and sensors provide identification of physical objects and sensing of physical parameters. While technologies such as IEEE 802.15.4 and Bluetooth are responsible for data collecting.

- **Transportation layer:** it includes mechanisms to deliver the collected data to applications and to different external servers. Methods are therefore required for accessing the network through gateways and heterogeneous technologies (e.g., wired, wireless, satellite), as well as for addressing and/or routing.

- **Applications layer:** it deals with processing and analyzing information flows, forwarding data to applications and services, and providing feedbacks to control applications. In addition, it is responsible for critical functions such as device discovery, device management, data filtering, data aggregation, semantic analysis, and information utilization/distribution. Indeed, these functions are essential for IoT ecosystems, and as such, must be handled by an IoT middleware platform.

The first step towards IoT is the collection of information about the physical environment (e.g., temperature, humidity, brightness) or about objects (e.g., identity, state, energy level). Data acquisition is encompassed by using different sensing technologies attached to sensors, cameras, GPS terminals, while data collection is generally accomplished by short range communications,

which could be open source standard solutions (e.g., Bluetooth, ZigBee, Dash7, Wireless M-BUS) as well as proprietary solutions (e.g., Z-Wave, ANT).

Once data is gathered through sensing technologies, it needs to be transmitted across the network in order applications can be able to consume the data. Heterogeneous communication technologies form the backbone to access the network.

When data arrives in the application layer, information flows are processed and then forwarded to applications. The IoT middleware layer covers a fundamental role for managing the above operations, being crucial for hiding the heterogeneity of hardware, software, data formats, technologies, and communication protocols that characterize an IoT ecosystem [13]. Besides, it is responsible for abstracting all the features of objects, network, and services, and for offering a loose coupling of components. Additional features of this layer are service discovery and service composition.

2.2 IoT Middleware

IoT ecosystems are based on a layered architecture style and use this view to abstract the integration of objects and to provide services solutions to applications [27]. In IoT, high-level system layers as the application layer are composed of IoT applications and middleware system, which is a software layer interposed between the infrastructure of devices and applications, and is responsible for providing services according to devices functionality [7]. IoT middleware systems have evolved from hiding network details to applications into more sophisticated systems to handle many important requirements, providing support for heterogeneity and interoperability of devices, data management and analysis methods, security, etc.

Many of the system architectures proposed for IoT middleware comply with Service-Oriented Architecture (SOA). This approach allows each device to offer its functionality as standard services. Moreover, SOA architecture refers to enabling interoperation in various domains, like industry, environment, and society, and supports open and standardized communication through all layers of web services.

The IoT can use a SOA-based IoT middleware in order to meet applications needs, providing infrastructure abstractions to applications and offering multiple services [10]. In this sense, Cooperative Middleware Platform as a Service for Internet of Things Applications (COMPaaS) [3] is a SOA-based IoT middleware that provides to users a simple and well-defined infrastructure of services. Behind these services there is a set of system layers that deals with the users and applications requirements, for example, request and notification of data, management of computing devices, communication issues, and data management. Section 2.2.2 presents COMPaaS in details.

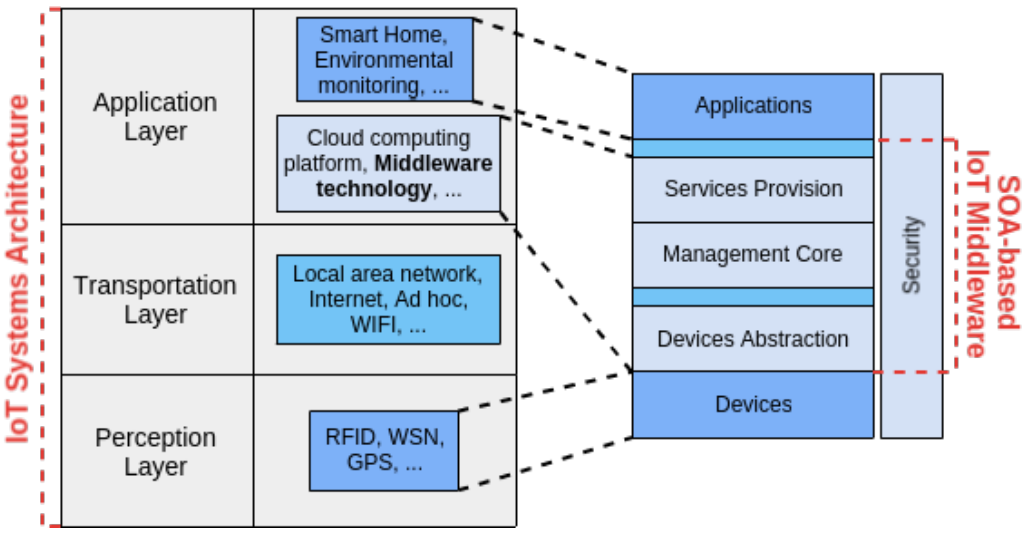

Figure 2.3 – SOA-based IoT middleware architecture.

2.2.1 Service-Oriented Architecture (SOA)

In IoT ecosystems, computation, storage, data management and analysis methods, and communication services are intended to be highly ubiquitous and distributed. Furthermore, the entities of the environment (people, things/objects, platforms, and surrounding spaces) are a highly decentralized pool of resources, which must be interconnected by a dynamic network of networks.

The smart integration of both intended services and entities represents the real ecosystem of the IoT. In this context, middleware for IoT is considered an important building block for the provision of IoT services, which are extremely desired to be highly pervasive and distributed. Indeed, middleware is an IoT platform intended to be a service of services to IoT ecosystems.

The notion of service-based IoT systems has been realized according to the principles of SOA and ROA (Resource-Oriented Architecture) architecture styles, which increasingly coexist in the IoT ecosystem since ROA allows the deployment of lightweight SOA-based communication mechanisms embedded into resource-constrained IoT devices [24]. SOA-based techniques provide to IoT applications a uniform and structured abstraction of services for communication with IoT devices. On the other hand, ROA-based approaches realize the necessary requirements to make the devices (things) addressable, searchable, controllable, and accessible to IoT applications through the Web.

According to Figure 2.3 (which is an extension of Figure 2.2), IoT middleware is a software layer or a set of sub-layers interposed between technological (perception and transportation layers) and application layers. The middleware's ability to hide the details of different technologies is fundamental to exempt the programmer from issues that are not directly pertinent to his focus, which is the development of specific applications enabled by IoT infrastructures [7]. In this way, IoT middleware has received much attention in the last years due to its major role of simplify the development of application and the integration of devices.

The devices layer can be composed of any IoT device which can connect to the middleware to provide services based on its features/resources. The devices abstraction layer can be embedded into both devices and middleware. Each service in the services provision layer is composed of one or more services from the devices. The devices function is abstracted into services by the devices abstraction layer and provided by the middleware through the services provision layer.

The applications should use an Application Programming Interface (API) from the services provision layer to consume the provided services. All the processing activity is generated in the management core layer also called middleware core. The security layer must ensure security in all exchanged and stored data, since the middleware architecture enables some vulnerability points that can be explored by security threats.

2.2.2 Cooperative Middleware Platform as a Service

COMPaaS [3] is an IoT middleware system that provides to users a simple and well-defined infrastructure of services. The goals of the COMPaaS can be summarized as follows:

- Abstraction of the integration and interoperation between applications and physical devices through the provision of hierarchical system services according to device profiles (i.e., a set of functional characteristics describing each physical device);
- Abstraction of the collection and management of data provided by physical devices through the provision of application-level services;
- Provision of high-level services to facilitate the development and integration of IoT applications; and
- Provision of a software architecture based on IoT/M2M and WoT (Web of Things) standards.

The COMPaaS architecture is based on the SOA approach [7] and is composed of two main systems according to Figure 2.4: Middleware and Logical Device. Logical Device is the system responsible for hiding all the complexity of physical devices and abstracts the functionality of these devices to the upper layer. Both systems are explained next.

- **Middleware**: Middleware is the system responsible for abstracting the interaction between applications and devices and also for hiding all the complexity involved in these activities. It provides an API to be used by applications in order to use the services of COMPaaS. The main functions of the middleware range from data management to device integration and address the provision of high-level services to applications.
- **Logical Device**: This is the middleware abstraction for the physical devices that are relevant to the applications and that must be accessible to provide some benefit. Logical Devices are

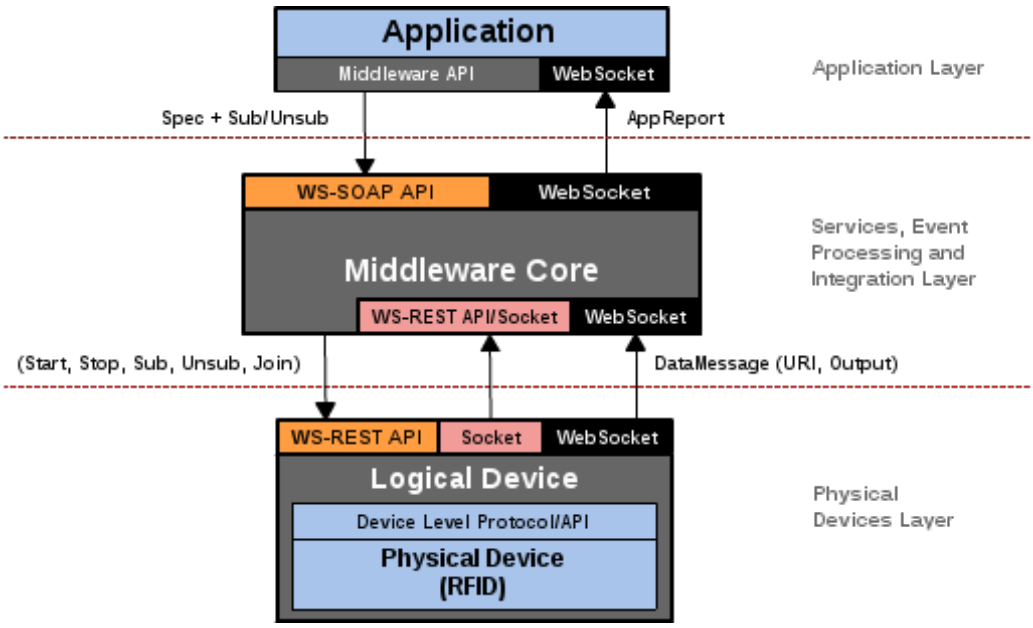

Figure 2.4 – COMPaaS architecture.

described and registered through system profiles. Each system profile contains attributes to characterize the physical device, such as: name, manufacturer, function, model, data type, URI, etc. These attributes are used by applications to find the desired devices. Besides the profile, Logical Device also contains two more system elements: Communication Module and Service Module. The Communication Module is not only responsible for the registration of the resource in the middleware, but also for the provision of the features for data communication and for notification of the operational status of the resource to the middleware. On the other hand, the Service Module is responsible to expose the interfaces and features of the resource to the middleware.

2.3 IoT Middleware Challenges

IoT has an immense potentiality for developing new intelligent applications in nearly every field. This perspective is possible, mainly, due to its double ability to perform situated sensing (e.g., collect information about natural phenomena, medical parameters, or user habits), and to offer tailored services. Regardless of the application field, such IoT applications aim to enhance the quality of people in everyday life, and will have a profound impact on the economy and society.

Although the rapid increase of worldwide mobile data traffic is mainly driven by the video sharing related content, it is expected that many innovative and unconventional IoT services and applications will materialize in the near future due to the maturity of the 5G technology. Otherwise, at least those applications that are infeasible due to technological limitations can benefit with 5G in the future.

In Section 2.2 we emphasized the importance of having a "standard platform for middleware system" able to cope with the desired "horizontal abstraction approach" (i.e., a common service/application management layer), which is an important requirement that has been lacking in current IoT ecosystems architectures around the world in the last years. Although this horizontal approach aims to provide a better interoperation, coordination, and optimization of services and applications from different domains (e.g., through the analysis of heterogeneous information flows), this desired middleware layer should not only merely take care of data management and interoperability issues. Conversely, it should provide a reliable and smart support for advanced challenges that go beyond classic IoT middleware requirements (such as interoperability, scalability, data management, device discovery, security, etc), and that should imply in rethinking some middleware functionality, mainly if we consider that, in the near future, IoT applications will be increasingly more pervasive and centered on users life experiences thanks to a ubiquitous coexistence of IoT and 5G.

Most of the estimated technological impact in the middleware design will be sourced by features like: (1) potential data processing in cloud (since advanced mobile/broadband networks will be able to transmit such data in a reliable and fast manner), (2) high scalability and interoperability, (3) support for managing virtualized networks, (4) support for managing next-generation networks and transport technologies, and also (5) support for flexible and powerful nodes at the edge of the network.

Based on these potential impacts there are several important challenges as: (1) cloud-based big data management, (2) interoperability and scalability, (3) context-based smart services provision, (4) security and privacy, (5) device discovery, and (7) stream analytics.

The following Sections 2.3.1 and 2.3.2 present important challenges regarding the device discovery and stream analytics, which are topics addressed in this work.

2.3.1 Device Discovery

The major idea of IoT is the pervasive presence around us of a variety of things or devices such as RFID tags, sensors, actuators, mobile phones, etc., which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals [20]. To provide value added services to the end users through IoT platforms, these devices must interact with the environment and among themselves. An inherent problem is that a lot of these devices will be “headless”, meaning they lack the screens, buttons and other features that would normally let a person interact with the technology directly.

That is, to realize the vision of IoT, there must be mechanisms available for automatic discovery of devices, their properties and capabilities as well as the means to access them. Furthermore, such discovery mechanisms also depend on other services like configuration management, registration and un-registration of smart objects. This makes discovery a fundamental requirement for any IoT framework and platform [23].

Devices discovery can be made by various different approaches, as follows [20]: (1) Distributed and P2P discovery services, where the philosophy behind such system is peer-to-peer (P2P) distributed systems; (2) Centralized architecture for resource discovery, generally have a central registry as backbone of the architecture; (3) Semantic based discovery, most ontology based systems; and (4) Search engine for resource discovery, systems that provide indexing, look-up and ranking facilities.

2.3.2 Stream Analytics

Data produced by devices contain information as "where they are", "how they are", etc. But it is what we do with these data that really matters. In this way, an important requirement for IoT frameworks is to allow domain experts to adjust system behavior by defining and refining business rules, and, as consequence, trigger events at the same time as the data stream comes from the periphery to the center of the system.

Examples of business rules can be: (1) If the temperature (from some device) is equal to 30 F (-1 C) and humidity (from some other device) is less than 30 percent, set forecast status as "snowstorm approaches"; (1) If a "snowstorm approaches", crank up the furnace; (2) If the temperature is out of the last 10-day norm, alert the operator; (3) If 2 devices get in range, send a collision alert.

Due the exponential explosion of events due to IoT, companies are already leveraging the combination of event stream processing, business rules management, and real-time analytics to capitalize on these opportunities. A few examples include the use of decision analytics to deliver better, faster customer service; decision analytics to deliver context-aware marketing sending offers to mobile phone subscribers; and decision analytics for fraud detection.

Based on [38], six distinct types of applications (as listed in Figure 2.5) are emerging in two broad categories: information and analysis and automation and control.

Information and analysis: as the new networks link data from products, company assets, or the operating environment, they will generate better information and analysis, which can enhance decision making significantly. Some organizations are starting to deploy these applications in targeted areas, while more radical and demanding uses are still in the conceptual or experimental stages.

Automation and control: making data the basis for automation and control means converting the data and analysis collected through the IoT into instructions that feed back through the network to actuators that in turn modify processes. Closing the loop from data to automated applications can raise productivity, as systems that adjust automatically to complex situations make many human interventions unnecessary. Early adopters are ushering in relatively basic applications that provide a fairly immediate payoff. Advanced automated systems will be adopted by organizations as these technologies develop further.

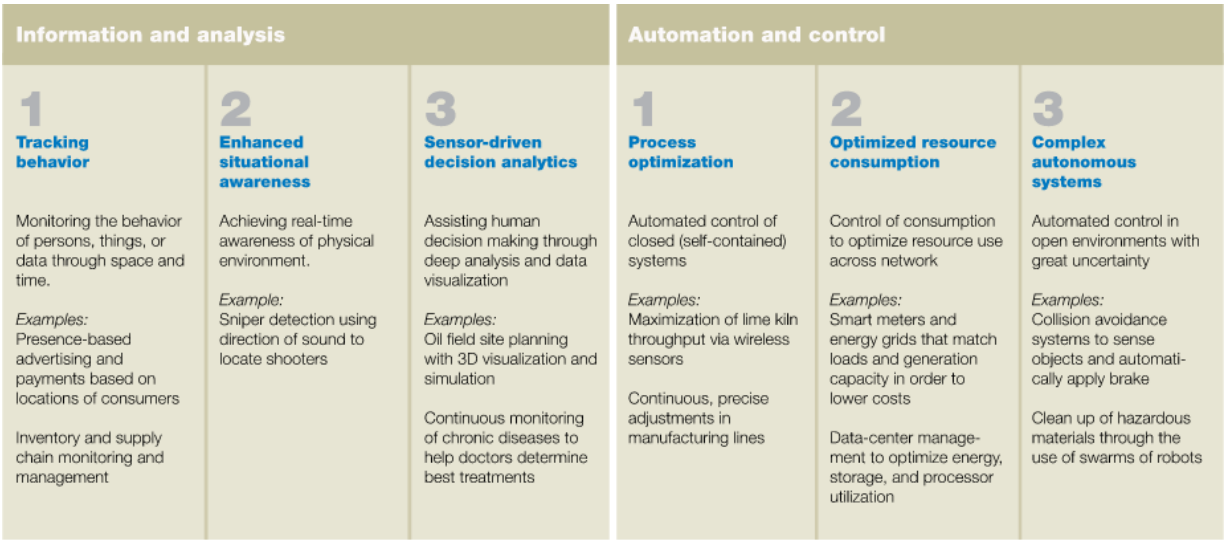

Figure 2.5 – Decision analytics and distinct types of applications that are emerging [38].

2.3.3 Complex Event Processing

Complex Event Processing (CEP) [9] is an technology for building and managing information systems. The goal of CEP is to enable the information contained in the events flowing through all of the layers of the enterprise IT infrastructure to be discovered, understood in terms of its impact on high level management goals and business processes. CEP is a new field that deals with the task of processing multiple streams of simple events with the goal of identifying the meaningful events within those streams. CEP employs techniques such as detection of complex patterns of many events, event correlation and abstraction, event hierarchies, and relationships between events such as causality, membership, and timing, and event-driven processes [32].

There is no broadly accepted definition on the term Complex Event Processing. Luckham [32] defines that events are related to other events by time, causality, and aggregation. Time is a property of an event in the form of a timestamp (although timestamps from different clocks may not be compatible depending on synchronization). A $\rightarrow$ B means that event A caused B, i.e., A *had to happen* before B, or B depends on A. Luckham's Cause-Time Axiom states that if event A caused event B in system S, then no clock in S gives B an earlier timestamp than it gives A. A higher-level event A may be created from a set of events and signifies a *complex event* that consisting of all the activities that the aggregated events signify. These three relationships are transitive and asymmetric.

From this viewpoint, CEP can be regarded as a service that receives and matches lower-level events and generates higher-level events, for example, a pair of buy and sell orders may be aggregated into a transaction [47].

3. RELATED WORK

In this chapter, we present studies that have relevance related to the proposed work. The aim for this presentation is to improve the understanding of current research related to both IoT middleware requirements addressed in this work (device discovery, stream analytics) to establish a theoretical and scientific basis for defining the strategies, components and features of the framework.

3.1 IoT Search Engines

An IoT directory must support search in order to discover heterogeneous devices and data [44]. Searches enables the retrieval of derived information based on syntactic, semantic, and structural information contained in data.

A search engine is a system that aims to create an index of objects and use this index to respond queries from users. For device discovery, the role of the search engines is to act as a meeting point for IoT context producers to register the availability of their things and sensor devices, and IoT context consumers to discover them. In this sense, the next related works are related to device discovery.

Today, most of the leading midddleware solutions provide only limited sensor search and selection functionality. As shown in the Figure 3.1,

Linked Sensor Middleware (LSM) [15] [30] provides limited functionality such as selecting devices based on location and device types. Nevertheless, all the searching capability uses SPARQL query language.

GSN [22] is another IoT middleware similar to LSM that provides a middleware to address the challenges of device data integration and distributed query processing. In short, GSN lists all devices available in a combo-box, which is used by the user to select the desired device. Xively [31] (formely known as COSM) is another approach that connects and collects data from devices to provide real-time control and data storage. Xively offers only keyword search. Microsoft SensorMap [39] only allows users to select sensors by using a location map, by sensor type and by keywords.

Related to sensor data search, the research in the IoT search engine area is still very limited [16]. There are already some pioneers works in real-time retrieval of sensor data with some prototype systems developed, like Snoogle [55], Microsearch [52], and MAX [58]. All these systems require that the sensors to be searched have a textual description attached so that they can be searched through keyword matches.

In location-based searches of sensor data, the authors of Senseweb [21] proposed a method to express sensor locations in meta-data and to support current-location-based searches. However, the work mainly supports static locations while for moving sensors, only the latest locations can be retrieved.

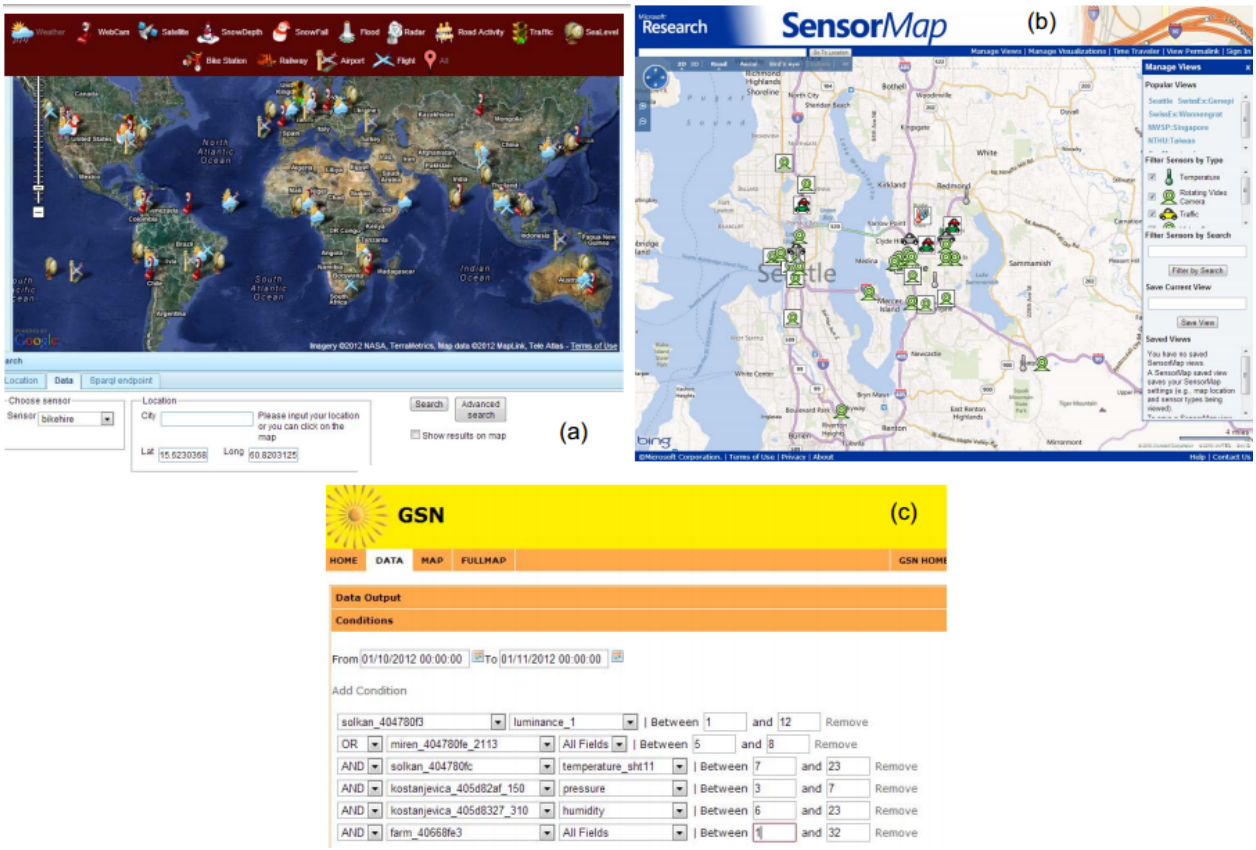

Figure 3.1 – (a) Linked Sensor Middleware (LSM) by Digital Enterprise Research Institute [15], [30], (b) SensorMap by Microsoft [39], (c) Global Sensor Network (GSN) initiated by EPFL [22].

Dyser is a search engine proposed by Ostermaier et al. [40] for real-time IoT, which uses statistical models to make predictions about the state of its registered objects (sensors). When a user submits a query, Dyser pulls latest data to identify the actual current state to decide whether it matches the user query. Prediction models help to find matching sensors with minimum number of sensors data retrievals. It support constraints based on the latest states of sensors, but historical states can not be retrieved.

A hybrid search engine is proposed in [16]. The authors also have noted that there is very limited work on search engine for IoT. It uses an index layer on top of the storage. The main goal here has been developing a search engine for effective multimodal query processing to obtain data generated by things in real time.

3.2 Decision Analytics Support Systems

We identified that there is a lack of dedicated IoT frameworks and middlewares supporting stream analytics. Therefore, we decided to expand and address IoT frameworks or middlewares that makes use of reasoning algorithms and analytics for other purposes. These frameworks and middleware are described below.

SAS [49] allows to get value out of streaming data from operational transactions, sensors, devices, transmissions, etc. The software assesses data streams using a suite of pre-built operators, functions, routines and advanced analytics. It also allows to derive insight into events and take appropriate actions.

IBM SPSS [25] enables to optimize high-volume transactional decisions before deployment. SPSS Analytical Decision Management use a combination of predictive models, business rules, optimization and scoring, the solution guides employees and systems to instantly make the best possible data-driven decisions.

Both SAS and SPSS generic approach is to select a subset of data from the data warehouse, perform sophisticated data analysis on the selected subset of data to identify key statistical characteristics, and to then build predictive models. Finally, these predictive models are deployed in the operational database. For example, once a robust model to offer a room upgrade to a customer has been identified, the model (such as a decision tree) must be integrated back in the operational database to be actionable.

Pentaho IoT Platform [41] allows to integrate, transform, and orchestrate machine and sensor data in a Big Data environment, as well as blend big data with data from traditional information systems. SenaaS [1], is an Internet of Things (IoT) virtualization framework to support connected objects sensor event processing and reasoning by providing a semantic overlay of underlying IoT cloud. The use of EDSOA paradigm assists the framework to leverage the monitoring process by dynamically sensing and responding to different connected objects sensor events.

Wang et al. [56] presents a design of middleware for EPC Sensor Network (ESN) architecture. It collects RFID and sensor data from distributed readers and processes this large volume of data in real time. After filtering, grouping and aggregating, well formatted data reports are sent to the upper layer. The middleware adopt CEP technology to discover information among multiple events.

Wu et al. [57] presents a system that executes complex event queries over real-time streams of RFID readings encoded as events. These complex event queries filter and correlate events to match specific patterns, and transform the relevant events into new composite events for the use of external monitoring applications. Dong et al. [17], discusses core principles for middleware to apply for the real time actions and the implementation of these principles through technologies such as Event Processing Language (EPL), caching strategy and active database.

3.2.1 Rule-based Languages

Many of the rule-based languages are bounded to a specific inference engine [48]. This link can be good or bad, depending on the quality of the engine. Expressiveness and easiness of use are two other important criteria for evaluation. The rule-based language must be quite generic, versatile and capable of large expressiveness, going from simple to complex rules. Knowledge engineers and

operators must be able to dynamically add rules to the system. The language must thus be easy to use to allow anyone (with a minimal training) to create and add rules on the fly. Community support is also an important asset to consider.

Appropriate toolkits and inference engines already exist for complex event processing, such as Jess and Drools. It play useful roles in formulating machine-readable rules for determining the trigger sequences of events for a particular activity or process. In this way, the candidate inference engines that have been identified as potential for DMS system are: OO jDrew, Jboss Rules (Drools), Mandarax, CLIPS and Jess. Each of these engines has benefits and drawbacks, and choosing the best one depends on the application being developed.

Based on a survey [48], we found that the best three rule-based engine candidates for our project are JBoss Rules (Drools), Jess, and OO jDrew. All three engines have a good user/community support, they have been there for many years and are considered as mature rule-based engines, they have been applied to different applications successfully and are quite versatile, and they have good reviews in general (no major issues). Bellow is a summary of the main characteristics of these three candidates:

- OO jDrew: Open-source, easy to implement, language POSL easy to use, not JSR-94 compliant, performance issues to be evaluated.
- JBoss Rules (Drools): Open-source, tool suite to support the engine (many development tools for creating/supporting/testing rules), language complicated but understandable to Java programmer and supported by a mapping tool (DSL), JSR-94 compliant, support the RETE algorithm, allows prioritization of the rules, already implemented for the JBoss environment (used for the project), and has good performance review.
- Jess: Non-free license, language complicated (similar to CLIPS, with overuse of parenthesis), JSR-94 compliant, support the RETE algorithm and has good performance review.

The main drawbacks of OO jDrew are that it is not compliant with JSR-94, it does not support the RETE, ReteOO or Phreak algorithms, its performance remains to be evaluated, and the lack of support tools.

The features of Drools and Jess have been compared. Drools uses natural-language-based rule set, editing, and development (*drl* and *dsl* files), whereas Jess requires specific syntax to represent rules (JessML or CLIPS). Drools and Jess both support facts that can be expressed in standard Java class and methods. They both follow the Java Bean approach to insert Java objects in working memory. Java Beans properties are similar to the list of slots in the facts of Jess and Drools. With respect to Drools, facts are objects (Java beans) from the application that are being asserted into the working memory. The fact that Drools and Jess both support the JSR-94 standard is also an asset that provide the potential for more easily interchanging these tools in the future, for research purposes.

4. SYSTEM APPROACH

The goals of our system framework are two-fold: (1) to allow software engineers to search, select, and interact with devices that best suit the application requirements, and as consequence, make IoT middleware patterns transparent to users (e.g., users do not have to understand the middleware patterns to define and subscribe specification reports); and (2) to allow software engineers to define business rules, and use data from devices when it is still in motion, extracting valuable information from it through CEP, allowing near real time decision making through event-based reports.

In order to make this possible, IoT middleware systems should follow some standardized architecture. We believe SOA is a consolidated and well-defined system architecture pattern that provides an architecture able to support the interactions needed to make our system goals possible. In this sense, we decide to choose the COMPaaS middleware as the reference platform for this proposed work.

The Decision Suport IoT Framework is complied with any SOA-based middleware solution that supports the Subscribe/Notify communication pattern, as well as data requesting mode using XML and Web Services. The *goal 1* is addressed in Section 4.1, and *goal 2* is addressed in Section 4.2.

4.1 COBASEN: Context-based Search Engine

COBASEN [34] is composed of two main components (see Figure 4.3). The first is the Context Module that is responsible for gather the device context and related data from the middleware and send it to the Search Engine. The second component is the Search Engine that is responsible for indexing the device information and answer queries using the index. The Search Engine also provides a graphical interface that allows users to select one or more devices, as well as to set the specification through specific parameters. Finally, the Search Engine sends the specification to the middleware layer and gives a feedback to the user.

The steps proposed for the COBASEN are presented in Figure 4.1. COBASEN allows users to search and select devices according to the users needs (steps 1 and 2). After the aggregation (selecting a set of devices), the user fills in a specification form (step 3). Next, the Search Engine creates the aggregation specification according to the middleware patterns and submits it to the middleware (steps 4 and 5).

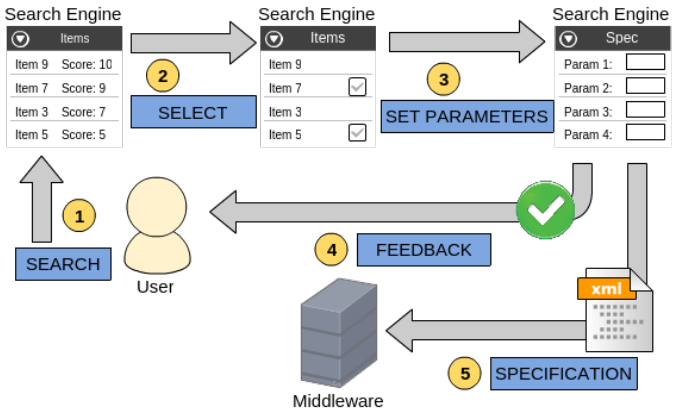

Figure 4.1 – Overview of COBASEN

4.1.1 Context Module

As previously stated, in the IoT paradigm, there will be a large number of sensors attached to everyday objects. These objects will produce large volumes of sensor data that has to be collected, analysed, fused and interpreted [46]. In order to accomplish the sensor data fusion task, context needs to be collected. Therefore, the aim of the Context Module is to assemble an formal structure to the informations of each device connected to the middleware and send that to the Search Engine allowing indexing and searching of this information.

The main goal of the Context Module is to gather primary context (any information retrieved without using existing context and without performing any kind of sensor data fusion operations) of all devices that are connected to the middleware. Context Module monitors the middleware, and each time a device is registered or updated, the Context Module send it to the Search Engine to updates the device list. An example of COMPaaS device data structure that are used in COBASEN can be seen in Figure 4.2.

We are not generating and indexing secondary context. Secondary context is any information that can be computed using primary context [44] (e.g., identify the distance between two sensors by applying sensor data fusion operations on two raw GPS sensor values).

4.1.2 Search Engine

Search engines are structurally similar to database systems. Documents are stored in a repository, and an index of the documents is maintained. An index is a data structure that makes the engine efficient to retrieve objects given the value of one or more elements of the objects. Queries are evaluated by processing the index in order to identify similarities, which are then returned to the user. However, there are also many differences: database systems must contend with arbitrarily complex queries, whereas the vast majority of queries to search engines are lists of terms and phrases [45].

```
<profileInformation xsi:type="deviceProfile"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <uri>
        http://192.168.0.10:8080/LogicalResource/resources
    </uri>
    <name>Termometer 15</name>
    <location>
        <name>Silo 4 - level 1 - pendulum 2</name>
        <latitude>-27.817244 S</latitude>
        <longitude>-51.697457 W</longitude>
    </location>
    <purpose>Temperature monitor</purpose>
    <owner>GSE</owner>
    <keywords>
        <keyword>Temperature</keyword>
        <keyword>Degrees</keyword>
        <keyword>Celsius</keyword>
    </keywords>
    <services>
        <service>
            <responseType>Double</responseType>
            <generationTime>1000</generationTime>
            <availableService>
                Temperature generation
            </availableService>
        </service>
    </services>
    <model>
        <deviceClass>Termometer</deviceClass>
        <number>1903</number>
        <name>Digital Termometer</name>
    </model>
    <information>
        <status>ONLINE</status>
    </information>
</profileInformation>
```

Figure 4.2 – Example of a XML file with information about a middleware device

There are many ways to implement indexes, and we shall not attempt to discuss the matter here. The most common situation is where the objects are recorded, and the index is on one of the fields of that record (e.g. books are objects recorded and their titles are the index field).

A popular method of indexing that is used in modern information retrieval systems is called Inverted Index [59]. Inverted File Index is a variation of Inverted Index that contains a list of references to documents for each word. For example, given two devices with the subsequent characteristics: D[1]: "The device is a temperature sensor"; and D[2]: "This sensor measures the humidity" – we have the following Inverted File Index (where the integers in the set notation brackets refer to the indexes of the text symbols, D[1], and D[2]): "a":{1}, "device":{1}, "humidity":{2}, "is":{1}, "measure":{2}, "sensor":{1, 2}, "temperature":{1}, "the":{1, 2}, "this":{2}. Another Inverted Index variation is the Full Inverted Index. This method contains a list of references to documents for each word and additionally contains the positions of each word within a document.

Search engines also have procedures to analyze variations in documents and queries. The verification of these variations can improve the searching process. These procedures can play functions such as check synonyms to search on words with equivalent meanings, stopwords (keywords that are not considered relevant), and so on [59].

In COBASEN, the Search Engine is responsible for performing the following tasks:

1. To index the list of devices that are received from the Context Module using an Inverted Index strategy;

2. To enable the search of devices through a query with the utilization of analyzer functions;

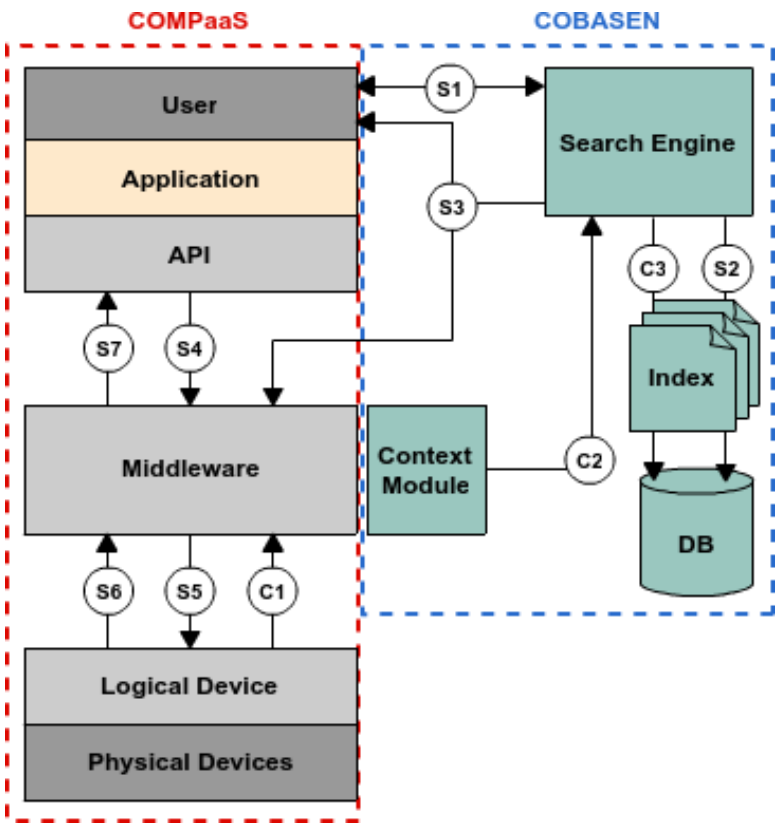

Figure 4.3 – Overview of the COBASEN architecture

3. To enable the selection of one or more devices;
4. To generate the specification of the selected devices;
5. To send the specification to the middleware; and
6. To send a feedback to the user containing the necessary information to make use of the specification.

4.1.3 COBASEN Architecture

The architecture of COBASEN is an extension of COMPaaS architecture (see Section 2.2.2). The system interactions (steps) are presented in Figure 4.3. The steps with the letter "C" correspond to the process of acquiring and indexing device context and related data. The steps with the letter "S" correspond to process related to the search engine and internal middleware functionalities. These steps may occur independently. Next, we will explain the flows of activities shown in Figure 4.3.

About the steps regarding to the letter "C": (C1) in this step the context module gather information related to devices that are connected to the middleware. This information contains characteristics of each device. (C2) The context information of devices is sent to the search engine; (C3) the search engine receives and indexes this information.

About the steps regarding to the letter "S": (S1) The user searches for device using a query; (S2) after the processing of user's request, the search engine returns the result list by relevance. The user select the desirable devices (set the aggregation) and define parameters; (S3) the specification is sent to middleware. At the same time, the essential data is sent to user (specification name) to make a subscription request; (S4) the user makes a subscription request to middleware using the specification name. In this moment the middleware event cycle is started; (S5) middleware starts the devices included in the specification report; (S6) middleware receives data from devices and makes the processing according to the specification report; (S7) finally, middleware sends a report to user containing the data from devices.

Note that after the creation of the middleware specification, it is stored in the middleware and the user application can access this specification as many times as necessary without the need to perform the steps C1, C2, C3, S1, S2, and S3 again.

4.2 DMS: Decision Management System

Streaming analytics aims to enable decision making. That is, streaming analytics become the key point for IoT applications. CEP could help in this task enabling action based on an analysis of a series of events that have just happened.

A huge number of applications requires continuous and timely processing of information as it flows from the perception layer. Examples include intrusion detection systems [28] which analyze network traffic to identify possible attacks; environmental monitoring applications which process raw data coming from sensor networks to identify critical situations; or applications performing online analysis of stock prices to identify trends and forecast future values.

Today, existing IoT support systems do not necessarily have event detection or processing capabilities. For many people big data is, erroneously, synonymous with the Hadoop framework [4]. But Hadoop does not have the ability to deal with streaming data, as is the case with IoT data.

While IoT data has similar characteristics as big data, IoT data is much more complex. One of the characteristics of IoT data is that it is "dynamic", in terms of "data in motion" as opposed to the traditional "data at rest". In this sense, DMS use device data when it is still in motion, extracting valuable information from it through CEP, allowing near real time decision making.

DMS architecture is an extension of COMPaaS (Section 2.2.2) and COBASEN (Section 4.1) architectures. DMS uses complex event processing to meet the needs of advanced users. It enable to trigger sophisticated reports and enhance automation and efficiency in ubiquitous environment. To build an efficient and scalable system, a rule engine Drools [8] is used to quickly and reliably to send notifications, based on the predefined users rules.

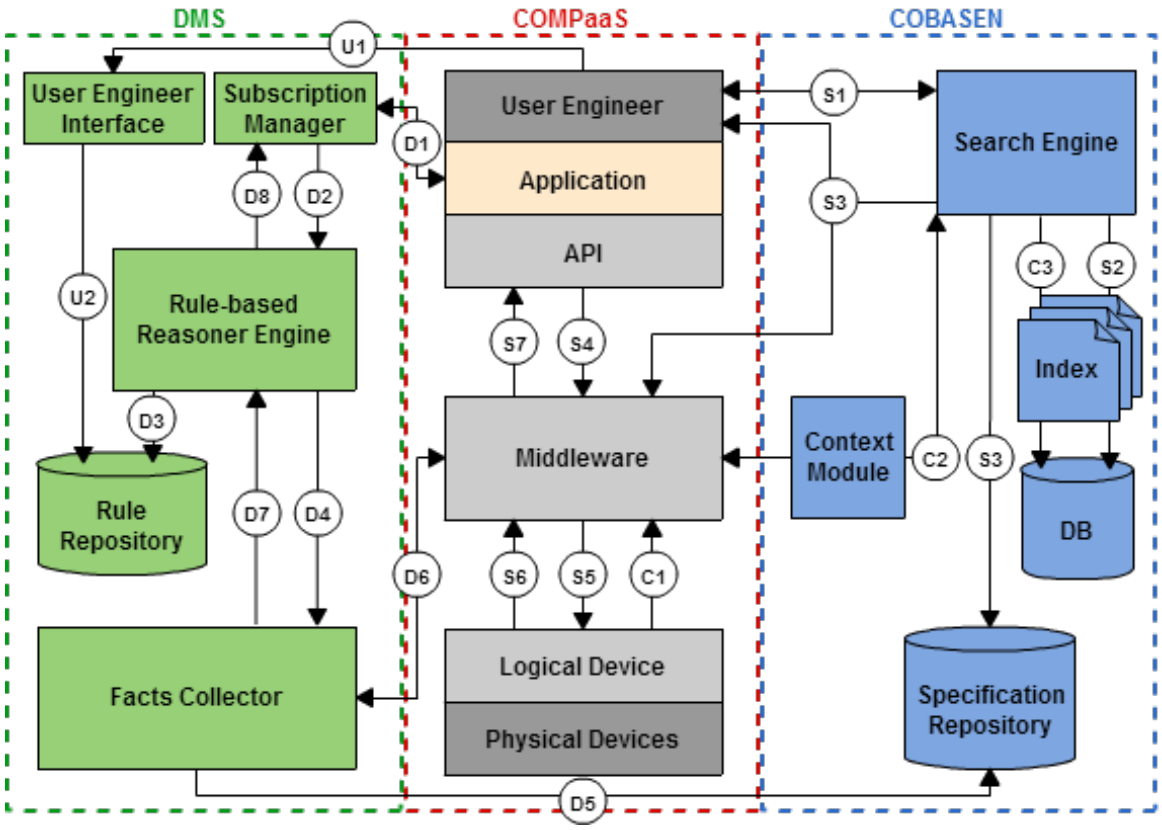

Figure 4.4 – Decision Managment System (DMS)

4.2.1 DMS Architecture

The DMS architecture is an extension of COMPaaS (Section 2.2.2) and COBASEN (Section 4.1) architectures. DMS uses the pre-defined and stored specifications that is created by users through COBASEN and data retrieved by COMPaaS middleware.

The heart of the system as shown in Figure 4.4 is the *Rule-based Reasoner Engine (RARE)*. *RARE* is responsible to load rules that are stored on the *Rule Repository*. The *Rule Name* and the *specifications IDs* are extracted from the rule base because they are essential information that will be used to gather the facts. The *Facts Collector* is used by the RARE to load the specifications on the COBASEN *Specification Repository* and gather all the facts using COMPaaS. When all facts are collected, *RARE* match the IF part of each rule against each asserted fact, executing the THEN part when a positive match occurs. When a positive match occurs, *Subscription Manager* notify all apps that are subscribers of that rule.

The system interactions are presented in the Figure 4.4 and 4.8. Each part of the DMS is described and explained in the Sections 4.2.2 to 4.2.7.

```
rule "Temperature Room 1 higher than 30"
        when
                //if the specification id is 1 and the value is higher than 30
                $s : Specification ( id == 1, value > 30 )
                //declare a Subscription Manager
                $submanager :   SubscriptionManager ( );
        then
                //use the Subscription Manager
                //notify the application sending the event description
                $submanager.notifyApp("Room 1 temperature is higher than 30");
end
```

Figure 4.5 – Specification attributes and subscription manager method

4.2.2 Rule Construction and Methods

As previously mentioned, we use Drools Engine [8] in *RARE*. Thus, the rule needs to follow its rule language. All information about the Drools Rule Language can be found in the Drools Expert User Guider [26]. Moreover, in this section we want to described how the *User Engineer* can use the specifications previously created in the COBASEN and some particular methods as the responsible to send the notification to the application.

The specification attributes that can be used in the rule is the specification *ID* and the specification *value*. The attribute *ID* is related to the ID given in the creation of the specification through *COBASEN*. The attribute *value* is the value that the *Facts Collector* module assigns to this specification *ID*. An example of the utilization of the specification attributes can be seen in the Figure 4.5.

The name of the rule is very important and it will be used by the application to subscribe it (e.g., in the Figure 4.6 the name of the rule is *Temperature Room 01 is higher than Room 02*). When the *User Engineer* understands that the application needs to be notified when the rule is executed, the *Subscription Manager* needs to be used calling the method *notifyApp* in the THEN part of the rule. This method takes as parameter a string, and it can be used to describe the event or data related to the specifications. An example of its utilization can be seen in the Figure 4.5.

Let's try to explain all these concepts in a scenario where the *User Engineer* would like to his application be notified when the temperature of the *Room 1* is higher than the *Room 2*. For that, consider thats he already selected all temperature devices of the *Room 1* and *Room 2*, and generated the specifications through COBASEN. In this case, let us consider that the COBASEN defined the specification ID for the *Room 1* as 1, and for the *Room 2* as 2. The resulted rule for this scenario is shown in Figure 4.6.

```
rule "Temperature Room 02 is higher than Room 01"
    when
        $spec1  :   Specification ( )
        $spec2  :   Specification ( $spec1.id == 1, id == 2, value > $spec1.value )
        $submanager :   SubscriptionManager ( );
    then
        $submanager.notifyApp("Room 02 temperature is higher than the Room 01");
end
```

Figure 4.6 – DMS rule example

4.2.3 User Engineer Interface

Users Engineer Interface is a User Interface (UI) that allow *Users Engineers* to upload rules. As previously mentioned, we use Drools rule engine in the system. In this way, only Drools rules extensions (.drl files) are allowed to be uploaded. As shown in Figure 4.4, the rules are stored in the *Rule Repository*. This module also allow to manage the uploaded rules (update, delete, and check details).

It is not the intend of this work to provide any kind of rule editor, debugger or validator. The Drools Expert User Guide [26] includes all information needed to develop rules. It includes all background related to the Drools Rule Language and also presents some IDE's, API's and Guided Editors that can be used to edit, debug and to validate the rules.

Prior to storing the rules is performed a validation to verify if the specifications IDs which were used in the rule are found in the *Specification Respository*. In this way, the user can not deploy rules which uses a specification which are not previously created (in the COBASEN).

4.2.4 Subscription Manager

Subscription Manager is a interface that allow the application to subscribe (to receive notifications) and to unsubscribe the DMS. It also is responsible to notify the user when *RARE* requests it.

To subscribe to DMS the application needs to sends a XML file through a RESTful Web-service containing a collection of names of the rules that it wants to subscribe (nameRules), and the address of the Websocket that will be used to send back the notification (notificationURI). A rule can be subscribed by more than one application. In this way, when the method *notifyApp* is executed, all subscribers of the specific rule are notified.

To unsubscribe the application needs to send a XML containing a collection of *nameRules* that it wants to unsubscribe and the *notificationURI* of the application.

Subscription Manager also is responsible to create in RARE the *Rule Lifecycle* and manage the observers (subscribers). It is better explained in Section 4.2.6.

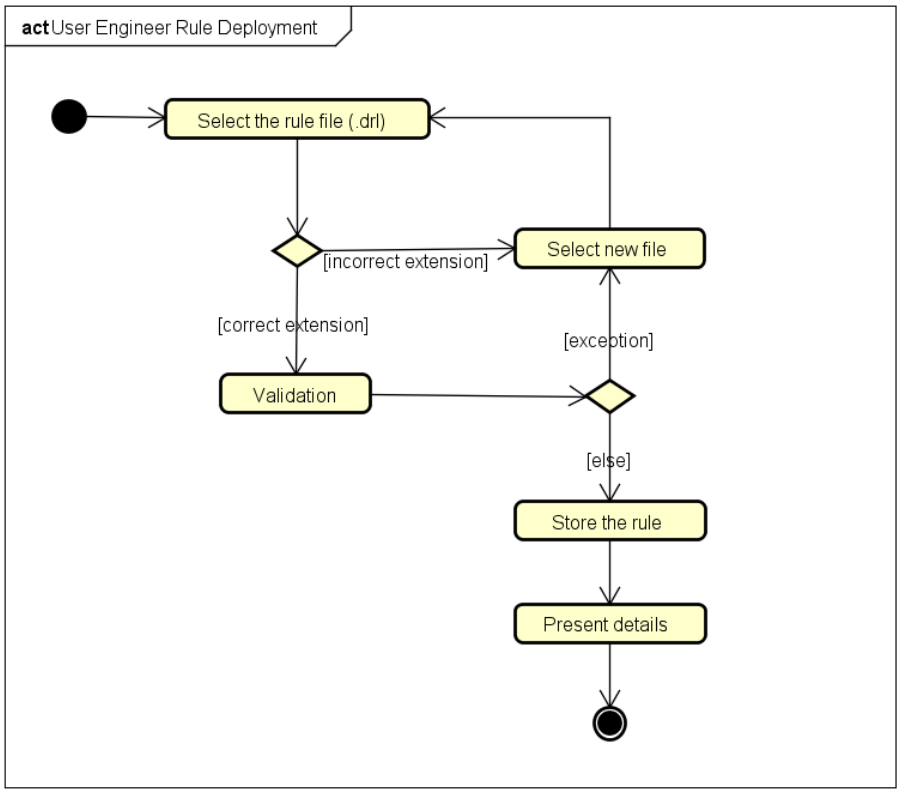

Figure 4.7 – User Engineer rule deployment activity diagram

4.2.5 Facts Collector

Facts Collector is used by *RARE* to gather facts that will be inserted into the working memory. For that, *RARE* sends a collection of specifications ID that compose a rule. After that, the *Facts Collector* creates an *lifecycle* that we called *Facts Lifecycle*. In this way, for each *Rule Lifecycle* (explained in Sections 4.2.6 and 4.2.7), there are multiples *Facts Lifecycles* (according to the number of specifications used in the rule).

When a *Facts Lifecycle* is started, it loads the related stored specification in the COBASEN and starts the *Gathering Process*. The gathering process use the specification XML and behave as an ordinal application that wants to subscribe the COMPaaS middleware to receive data (facts). The entire duration of the gathering process is assigned based on the highest specification duration. For example, if the gathering process is composed of two specifications, one with a duration of 1000 milliseconds and the other of 500 milliseconds, the entire gathering duration will be 1000 milliseconds. The entire process is invalidated if any *Fact Lifecycle* was not successfully gathered its respective specification. In this case, RARE sends a notification to application reporting that some of its devices related of a specification are inaccessible.

The following steps compose the *Gathering Process* using COMPaaS middleware: (1) create a connection (Web Socket) to receive back the data; (2) observe the connection to detect

when receives new data; (3) send the specification XML defining which (devices) and how is desired the data; (4) makes the subscribe to starts receiving data.

When all *Facts Lifecycle's* have completed the gathering process, then all facts is sent back to *RARE*.

4.2.6 Rule-based Reasoner Engine (RARE)

The Rule-based inference engines are composed of two main parts [48]: the production memory and the working memory. The production memory contains the rule-set (or rule-base) and the working memory contains facts (or data).

In *RARE*, when all facts related to a rule are collected, these facts are inserted into the working memory. After that, *RARE* tries to match the IF part of each rule against each asserted fact, executing the THEN part when a positive match occurs for all premises. The inference process continues until no more rules can fire. For this purpose, RARE use the Phreak algorithmn, which is possible through the Drools Engine.

A important case to reinforce, as explained in Section 4.2.2, is that the *User Engineer* needs to declare the method *notifyApp* in the THEN part of the rule if, in this cases, he wants his application to be notified.

In *RARE* is found the core of the system, the main *Lifecycle*. Each subscribed rule is related to a *lifecycle* that we called *Rule Lifecycle*. When an app subscribes a rule, the *Subscription Manager* creates a *Rule Lifecycle*, adds the application as observer, and start it. In this way, there are a list of *Rule Lifecycle's* and for each which is contained therein, there are a list of observers (used to know who needs to be notified when the *notifyApp* is fired). The *Rule Lifecycle* is responsible to coordinate all activities related to the inference process. That includes the cooperation with the *Facts Lifecycle* and others that are better explained in Section 4.2.7.

4.2.7 DMS Modules and Lifecycle

In this section we present and explain the DMS modules interactions and activities and *RARE Rule Lifecycle*. Also, it is presented the DMS modules activity diagrams shown in Figure 4.4.

As previously mentioned, DMS architecture is an extension of COMPaaS (Section 2.2.2) and COBASEN (Section 4.1) architectures. The systems interaction are presented in Figure 4.4. The steps (arrows) with the letters "U" and "D" correspond to the DMS system. The steps with the letter "C" and "S" are explained in Section 4.1.3. Next, we will explain the DMS flows of activities.

The U1 step, shown in Figure 4.4, represents the user selecting the rule file. Only ".drl" files are allowed to be selected and uploaded. The deployment of the rule through the UI User Engineer Interface is presented in Figure 4.7. In U2 step, prior to storing the rules a rule validation

is performed to verify if all the specifications IDs used in the rule file are found in the *Specification Respository*. In this way, the user can not deploy rules which uses specifications IDs which are not previously created (in the COBASEN).

Figure 4.8 presents the activity diagram of the DMS lifecycle, starting from the subscription moment, following to the abstraction of the rules, specification loading on the COBASEN, facts gathering through *Facts Collector* (which use COMPaaS middleware), inference process and the application notification.

The next steps are presented in Figure 4.4: (D1) The application makes the subscription; (D2) *Subscription Manager* creates the *Rule Lifecycle*, adds the application as observer, and start it; (D3) *RARE* extract the specifications ID which are used in the rule; (D4) *RARE* sends a collection of specifications ID to the *Facts Collector*; (D5) *Facts Collector* creates the *Facts Lifecycle's* and each of it loads the related specification XML at the COBASEN; (D6) *Facts Lifecycle* gather the facts; (D7) When all facts are collected, *Facts Collector* sends it back to *RARE*; (D8) *RARE* insert these facts into the working memory and fire the rules. If the THEN part of some rule are executed and the *notifyApp* method are declared in it, the *Subscription Manager* is used to notify the observers (applications) of that *Rule Lifecycle*.

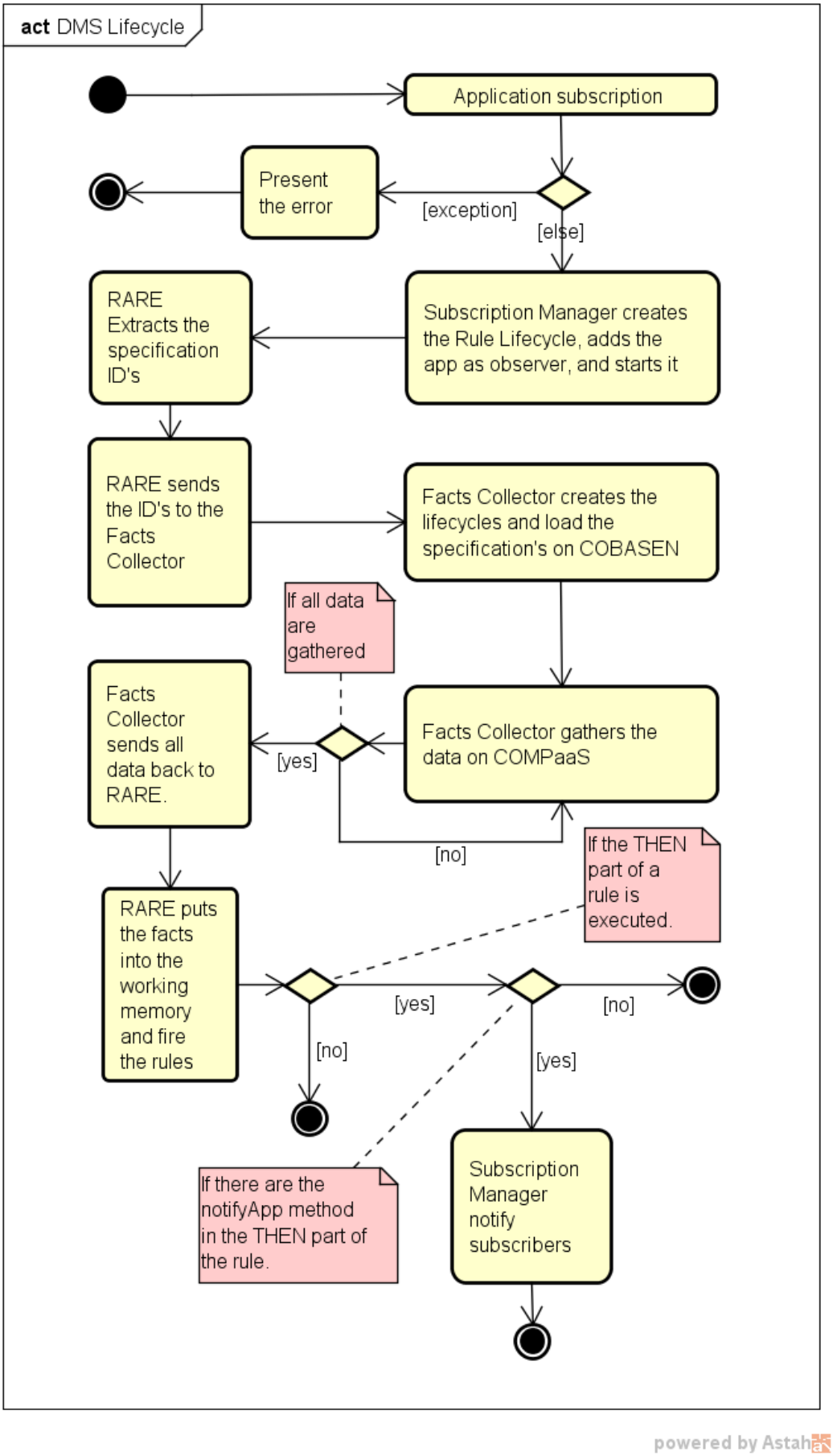

Figure 4.8 – DMS Lifecycle

5. EVALUATION

5.1 COBASEN

In terms of evaluation, we performed performance experiments in order to verify that the platform that implements the framework is functional. In this way, the results of the performance test are demonstrated at the end of this section.

We developed an application prototype that stimulates the indexing and searching features of COBASEN. Basically, we made experiments and analyzed two main requirements: (1) time elapsed while processing a query from user; and (2) time elapsed during an indexing activity. We conducted each experiment 20 times and averages were considered in the results.

5.1.1 Implementation

The COBASEN application prototype was developed using Java. The data used was stored in PostgreSQL database. The Figures 5.1, 5.2, and 5.3 were marked on key parts of the interface, and these are explained below.

The search interface of the prototype tool are shown in Figure 5.1: (1) users express their needs through a query, (2) present a list of devices ranked according to index, (3) allows to select the device, and (4) add the selected devices to the aggregation list. In addition, this step provides features to enhance user interaction: table pagination, number of devices per page, among others.

The aggregation list interface of the prototype tool are shown in Figure 5.2: (1) allows to select the device, (2) removes the selected devices from the aggregation list, and (3) displays a window that contains all device characteristics.

In Figure 5.3 we can see the parameters of the middleware specification definition interface. This interface allows user to choose the desired parameters for setting the middleware specification. *Repeat Period* is the time interval between event cycles without collecting data from devices. *Duration* is the total time of a data collection cycle in the middleware. *Data set* allows user to choose which collected data will form the data set. There are three options: Additions: it adds data of the current cycle but not from previous cycle, Deletions: it adds data that are present only in the previous cycle but not from current cycle, and Current: it adds only data presented in the current cycle. *Group By* allows aggregation of the resulting data by: Resource: aggregates data by URI, Location: aggregates data by location, and None: data does not have a aggregation parameter. *Operation* allows the user to choose the type of processing on the aggregate data, which can be: average, maximum, minimum, sum and none. In this interface, users also can see details and examples of each of the fields, and creates the specification which is sent to the middleware. These parameters compose the XML specification can be seen in Figure 5.4.

Figure 5.1 – Search interface of the prototype tool.

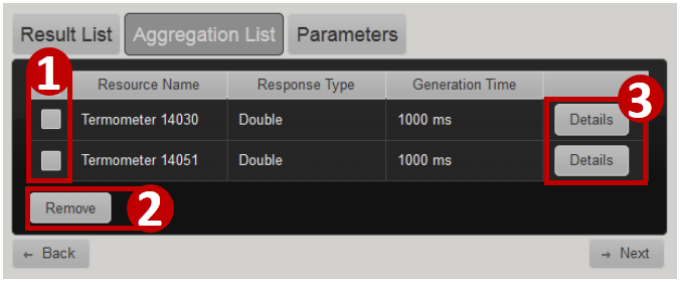

Figure 5.2 – Aggregation List interface of the prototype tool.

5.1.2 Evaluation

We used a computer with AMD FX-8350 (8 core) 4.0GHz and 16GB RAM to evaluate COBASEN. For the search engine (to index device context and related data), we employed the open source Apache Lucene API [5] to index data. Lucene is a high-performance search engine library written entirely in Java. This API provides consistent full-text indexing across both database objects and documents since it utilizes Inverted Index strategies. Salesforce, LinkedIn, and Twitter are examples of large-scale Lucene deployments.

We made the following assumptions regarding the evaluation. We assume that device context and related data are already retrieved by the context module. Besides, we collected data sets from COMPaaS and generated a synthetic data set. In other words, we have used a synthetic method of data generation in our evaluation, which provided device descriptions and context information for one million of devices.

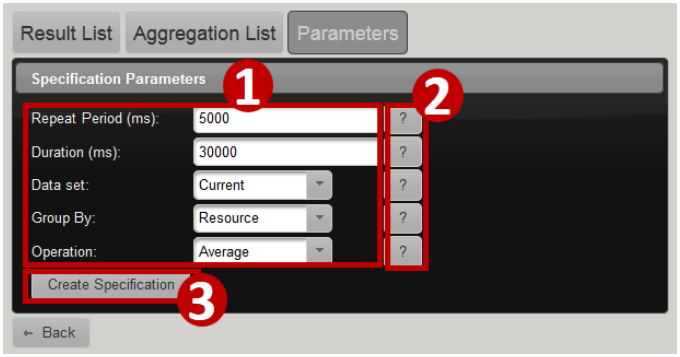

Figure 5.3 – Parameters definition interface of the prototype tool.

```
<ns3:ReportSpec
    xmlns:ns2="http://service.middleware.compaas/"
    xmlns:ns3="http://service.gather.middleware.compaas/">
    <resources>
        <resource>
            <uri>
                http://192.168.0.10:8080/Termometer14030/resource
            </uri>
        <resource>
    </resources>
    <constraintSpec>
        <repeatPeriod unit="MS">5000</repeatPeriod>
        <duration unit="MS">30000</duration>
    </constraintSpec>
    <outputSpecs>
        <outputSpec outputName="Spec14030">
            <setSpec set="CURRENT"/>
            <groupSpec groupBy="RESOURCE"/>
            <dataOutputSpec operation="AVERAGE"/>
        </outputSpec>
    </outputSpecs>
</ns3:ReportSepc>
```

Figure 5.4 – Example of specification report for the utilization of one device.

As depicted in Figure 5.5, the graph shows the searching processing time taken when the number of devices and the number of results per page gets increased. In this case, the number of context properties kept at 10. In the searching process were not used any pagination method to restrict the number of returned devices per page. Even in an extreme situation with 1 million of devices and a user making queries that returns up to 20000 results, the processing time did not exceed 500 milliseconds.

As depicted in Figure 5.6, the graph shows the processing time taken by the indexing process when the number of context characteristics and numbers of device get increased. Synthetic characteristics properties are used for that experiment. It is noticeable that reducing the number of context characteristics per each device (less information) improve the performance of the indexing method. The processing time starts to get increased after 100000 devices.

During the test round that used 10 context characteristics and one million of devices, the index size has reached 79.1MB. Table 5.1 presentes the index size values related to the indexing process shown in Figure 5.6.

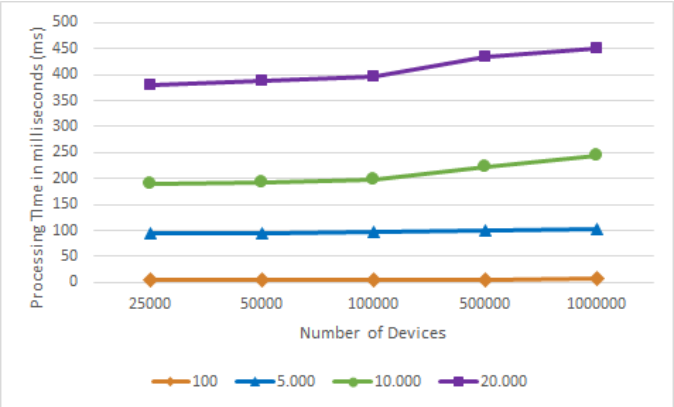

Figure 5.5 – Searching processing time.

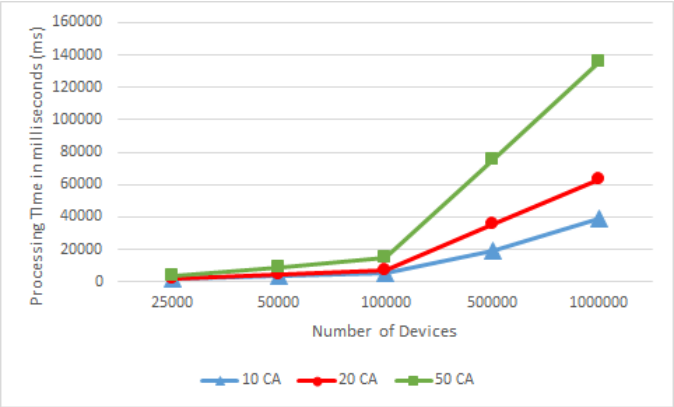

Figure 5.6 – Processing time taken by the indexing process

5.1.3 Discussion

With the constant growth of the IoT and the provision of a huge number of IoT devices in the near future, it is unfeasible that application programmers need to find the desired devices manually and without a standard middleware support. This limitation strengthens the fact that the adoption of a context-based IoT system support like COBASEN in an industrial environment has becoming even more necessary.

COBASEN is able to provide improvements in the IoT application development process that are coupled with the process of finding and usage of devices, since we provide a transparent method for users that do not know the domain and/or the middleware configuration patterns. The transparency provided by the COBASEN is due to the system interface (see Figures 5.1, 5.2, and 5.3) that hides the middleware layer complexity and guides the user in the creation of the middleware specification. In addition, an industrial environment that uses our framework will be able to develop different applications to improve their process. The discussed scenario in Section 1.1 is only one among the various categories of the industry that can take advantage of IoT technologies. Some application examples are: monitoring, automation, cost reduction, accident prevention, among others.

Table 5.1 – Index size (megabytes).

Devices/Characteristics	10	20	50
25000	18.7	37.4	93.5
50000	39.3	78.6	196.7
100000	79.1	158.2	395.2
500000	395.3	790.6	1976.3
1000000	791.7	1583.4	3958.5

These improvements are possible due to the system components of the COBASEN framework. The acquisition of the devices context characteristics provides the necessary information that enables the use of techniques for the retrieving of heterogeneous devices (in this case the search engine). Based on Figure 5.6, we can say that the indexing time increases with the increasing of amount of devices context characteristics.

As shown in Figures 5.5 and 5.6, with the use of Inverted Indexing strategies we can perform precise and faster full text searches in exchange of greater difficulty in the act of inserting and updating documents. It increases the efficiency of search and makes the system scalable. We must point out that the search method currently implemented in the prototype tool uses pagination only in its interface to improve user interaction with the system. In other words, the search method returns the entire list of results sorted by relevance and then the interface uses pagination to show this list. Therefore, Figure 5.5 shows that is evident that the use of techniques which allow restrict the number of devices per page can improve the searching process and the efficiency of COBASEN. For example, if we return 100 devices per page, the time would not exceed 7ms.

COBASEN presents itself as a essential tool to improve the IoT applications development. In IoT environments is very usual to come across the fact that people that deploy middleware are not the same that develop the applications. In this way, with the support of COMPaaS and COBASEN, users from IoT environments (IIoT, smart cities, etc.) will be enabled to develop applications for their environments without having to learn the middleware patterns and/or understand the devices modeling. It is a substantial improvement in terms of business processes and can be widely used to help users in large scale environments.

5.2 DMS

This section presents the evaluation tests performed in DMS system. We developed an application prototype and it was used to perform the DMS system validation. We made experiments and analyzed the following mainly points of the system: (1) time taken to validate and extract the rules specifications IDs; (2) time taken to start and load the system lifecycles in scenarios with different numbers of rules and specifications; and (3) time taken to process different number of rules. We conducted each experiment 20 times and averages were considered in the results.

5.2.1 Implementation

We have developed a Java-based implementation of DMS system. DMS architecture is SOA and the communication between the components takes place by passing XML-defined messages. XML Schema is an expressive language with a rich type system and user-defined data types. The current implementation supports publish/subscribe approach.

An open-source Java rule management system (BRMS) based-rules engine (Drools [8]) version 6.3.0.Final was selected as the rule-based reasoning component for this initial DMS prototype. In its current state, Drools tool suite supports the engine (many development tools for creating/supporting/testing rules), language complicated but understandable to Java programmers and is supported by a mapping tool (DSL), JSR-94 compliant, support the RETE algorithm, allows prioritization of the rules, and has good performance review.

A survey of open-source and commercial rule-based languages and tools performed by Roy, J. [48] was used to guide the selection of the inference engine. The main advantage of Drools over a non-free license similar tool (Jess [18]) would be its rule language (DRL), closest to Java and not over-using parenthesis. In addition, the use of Domain Specific Language (DSL) gives another advantage to Drools over Jess by allowing the user to customize its own rule language. Finally, the fact that Drools is free and Jess is not provides another advantage.

The DMS system modules and interactions are explained in Section 4.2. The system interactions are also presented in Figure 4.4, and the system flow and lifecycles are presented in Figure 4.8.

5.2.2 Evaluation

We used a computer with AMD FX-8350 (8 core) 4.0GHz and 16GB RAM to evaluate DMS. We made experiments and analyzed the main functions of major modules in order to analyze the DMS performance.

The goals of the tests were to evaluate the processing capability of the DMS system in order to verify if it is functional and suitable to IoT environments. In order to reproduce a real-world IoT scenario (huge number of rules), we created multiple synthetic Drools rule files in which each contains different rules and quantity of it. We conducted each experiment 20 times and averages were considered in the results.

In this way, we performed an experiment which aims to verify how much DMS decreases the amount of messages transmitted over the network. Thus, we defined an IIoT scenario in which physical sensors were responsible to gather data readings from the field. The collected device data would allow the optimization of an industry process.

For that, an IoT software engineer has defined two data stream specifications containing six thermometer devices each. Thus, each middleware report was composed of the average of all six devices and it had about 1,2KB. A new report was sent to the application every 1000 milliseconds.

There are two different scenarios in this experiment:

- Scenario 1: The user engineer uses COBASEN to search and select the desirable devices and create the data stream specification. The user engineer uses these specifications to gather the data directly on COMPaaS, and locally perform analytics.
- Scenario 2: The user engineer uses COBASEN as in the case 1, but in addition, he also defines a rule for this situation and subscribe to it in DMS. The rule is defined as follows: ***WHEN*** *the resulted value of the specification one is higher than 35 C°,* ***AND*** *the resulted value of the specification two is higher than 30 C°,* ***THEN notify*** *the application.*

To be able to evaluate the difference between these two scenarios, we define three temperature variation ranges related to industrial field: (1) Small variation: temperature ranges from 20 C° to 40 C°; (2) Medium variation: temperature ranges from 10 C° to 50 C°; (3) High variation: temperature ranges from 0 C° to 60 C°.

We have used an algorithm that simulates the temperature changes based on the predefined variations. Each 30 seconds it randomly defines its posture (i.e., if the temperature should increase or decrease). Then, for each second it starts to increase or decrease 1 C° based on its posture. However, it does not exceed the maximum and minimum variations values defined above.

For each experiment execution we have monitored the messages flow for 600 seconds. Table 5.2 presents the number of messages needed to fulfill the scenario described above. The scenario 1 (2nd row) presents the number of messages towards the three field temperature variation using COMPaaS directly communication. The scenario 2 (3th row) presents the number of messages towards the three field temperature variation using DMS.

Table 5.2 – Network Exhaust Evaluation

Scenario	Less Range	Medium Range	High Range
1	600	600	600
2	211	125	98

An experiment has been performed in order to validate the rule validation process (step U2 of Figure 4.4), which occurs after the upload of rule file (mentioned in Section 4.2.3 and shown in the Figure 4.7). To perform this test we have defined a test plan with different scales of rule files. That is, the validation function was tested with different rule files and each having distinct scales of number of rules and number of IDs used in the conditions of these rules. The results of the tests are presented in Table 5.3. The first column shows the number of rules contained in the rule file and the first row shows the number of IDs used in the conditions of the rule file.

In order to verify and validate the steps D2, D3, D4, D5, and D7 (related steps of Figure 4.4), we performed a test to stimulate all these steps of the system. The time taken to gathering

Table 5.3 – Rule validation function execution time (milliseconds).

Rule/IDs	5	10	20	30
3000	44	46	51	56
6000	77	79	94	106
12000	116	124	147	152
24000	129	133	158	190
48000	190	203	246	298
94000	351	348	434	485

data though COMPaaS (step D6) is validated in [3], in this way, the gathering time is not taken into account. To perform this test we have used the same rule files mentioned above. For each test round, we synthetically subscribed to all rules contained in one of these rule files. For example, in a scenario where the rule file contains 3000 rules and each rule uses 5 IDs per condition, the *Subscription Manager* creates 3000 *Rule Lifecycles* (step D2) and *RARE* extracts specifications IDs which are used in all of these 3000 rules. The time taken to load the *rule base* are not taken into account, and it is presented in Table 5.7. The results of this tests are presented in Table 5.4. The first column shows the number of rules contained in the rule file and the first row shows the number of IDs used in the conditions of the rule file.

Table 5.4 – DMS system execution time (milliseconds).

Rule/IDs	5	10	20	30
3000	1336	1485	1728	1890
6000	2065	2295	2794	3172
12000	3292	3780	4171	4482
24000	4945	5872	7492	9004
48000	8347	9868	12001	14512
94000	15321	17428	21681	24246

Drools 6.0+ comes with the new Phreak algorithm (enabled by default), which is a drop-in replacement for the ReteOO algorithm (which you can still use instead too). The usage has not changed and it still executes the same DRL syntax. Phreak algorithm open many doors for future optimizations, such as exploiting multi-core machines more efficiently [36].

We decided to test which algorithm would be more efficient (ReteOO or Phreak) for the DMS system taking account its unique requirements (large number of rules and large number of facts). Both algorithm was tested towards six different scenarios. ReteOO algorithm execution time is presented in Table 5.5. Phreak algorithm execution time is presented in Table 5.6. In both tables the first column represents the number of rules contained in the rule file and the first row represents the number of facts in the working memory. A comparison between the algorithms is displayed in a bar chart in Figure 5.7.

The rule base reading time (time to load the rule base) which occurs only when the rules file is updated, and the rule base size (bytes) are presented in Table 5.7.

Table 5.5 – Time taken (milliseconds) to determine possibles rules to fire using ReteOO algorithm.

Rule/Facts	500	1000
3000	4697	26438
6000	7755	32312
12000	12795	40390

Table 5.6 – Time taken (milliseconds) to determine possibles rules to fire using Phreak algorithm.

Rule/Facts	500	1000
3000	3663	21203
6000	6126	25946
12000	10236	32473

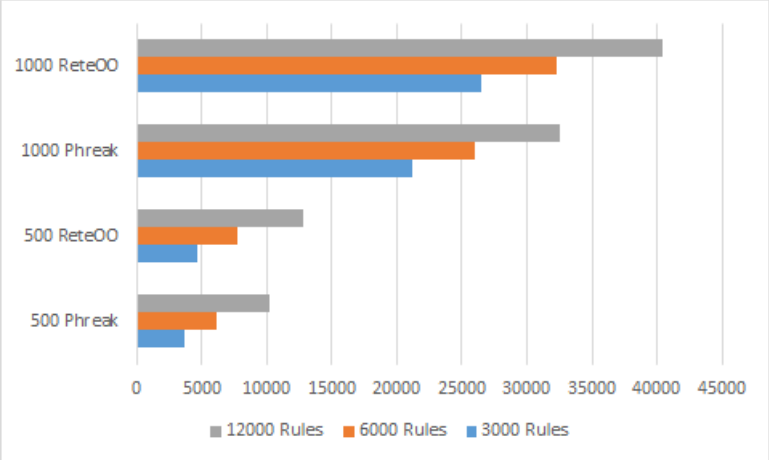

Figure 5.7 – ReteOO and Phreak comparison.

5.2.3 Discussion

Today's device-driven world is forcing analytics to occur as fast as the data is generated. In this way, an important requirement for stream processing systems is to process messages "in-stream" as they fly by, without any requirement to store them to perform any operation or sequence of operations. Storage operations adds a great deal of unnecessary latency to the process (e.g., committing a database record requires a disk write of a log record).

DMS follows the *straight-through processing paradigm*, and incorporate event-driven processing capabilities. It avoid applications to continuously *poll* for conditions of interest, removing additional latency to the process, because (on average) half the polling interval is added to the processing delay [50].

By making use of validated architectures as COBASEN and COMPaaS, DMS provides improvements in the IoT application development process. The utilization of COBASEN, allows to inherits all its benefits related to discovery and selection of devices. On the other hand, COMPaaS is who are directly connected and interacting with devices, and also is responsible to collect data.

Table 5.7 – Rule base (RB) reading time, and rule base size

Rules	RB Reading Time	RB Size (bytes)
3000	16514	762,589
6000	35657	1,530,471
12000	77145	3,076,478

More than that, DMS allows to apply immediate analytics insights from streams of data of devices into IoT applications.

As shown in Table 5.3, DMS can operate with an inordinate number of rules. We verified that the system time grows more by the increase of rules than the increase of the number of conditions. In this way, in cases where the number of *Fact Lifecycles* are the same (e.g., 12000 rules each of it containing 5 conditions, and 6000 rules each of it containing 10 conditions, both requires 60000 *Fact Lifecycles*). In these cases, time has increased more for the scenario that has more rules, since higher number of rules require more processing to manage subscribers, among other functions.

Regarding results presented in Table 5.2, DMS can reduces application processing overhead (i.e., application does not need to locally manage events) and makes it receives only relevant data. Furthermore, even in a scenario wherein the temperature is kept varying, and close to the rule condition, DMS reduces networking exhaust. The number of events and messages increase when the variation range occurs near the rule condition values. In this experiment, we have observed that DMS decreases 73,9% on average the amount of messages that are sent over the network from middleware to the application. DMS also prevents to store data that ultimately have no real value for future operations.

6. FINAL CONSIDERATIONS

6.1 Publications

In addition of the research and development of the reported work, some papers were written in order to expose and validate our proposal. The following list shows the published works:

- Lunardi, W.T.; Amaral, L.A.; Marczak, S.; Hessel, F.; Voos, H.: "Automated Decision Support IoT Framework", in Emerging Technology and Factory Automation (ETFA 2016) [33];
- Lunardi, W.T.; de Matos, E.; Tiburski, R.; Amaral, L.A.; Marczak, S.; Hessel, F.: "Context-based Search Engine for Industrial IoT: Discovery, Search, Selection, and Usage of Devices", in Emerging Technology and Factory Automation (ETFA 2015) [34];
- de Matos, E.; Lunardi, W.T.; Tiburski, R.; Amaral, L.A.; Marczak, S.; Hessel, F.: "Context-based Framework to Discovery, Search, and Selection of Computing Devices in the Internet of Things", in Congresso da Sociedade Brasileira de Computação - Seminário Integrado de Software e Hardware (CSBC 2015 - SEMISH) [14];
- Lunardi, W.T.; Marczak, S.; Amaral, L.A.; Hessel, F.: Discovery and Usage of Computing Devices in IoT Environments, in 2nd Latin-American School on Software Engineering (ELAS-ES 2015) [35];
- de Matos, E.; Amaral, L.A.; Tiburski, R.; Lunardi, W.T.; Hessel, F.; Marczak, S.: "Context-Aware System for Information Services Provision in the Internet of Things", in Emerging Technology and Factory Automation (ETFA 2015) [37];
- Amaral, L.A.; de Matos, E.; Tiburski, R.; Hessel, F.; Lunardi, W.T.; Marczak, S.: "Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G, in Internet of Things (IoT) in 5G Mobile Technologies (Springer Book) [2].

6.2 Conclusion

Things that seemed like science fiction 20 years ago (Self-driving cars, Refrigerators that send a text when you are out of milk) are rapidly approaching the market. It means that the world around us will become filled with devices. We identified through literature that there are significant amount of middleware systems solutions for device data management, integration, context management fields. However, IoT becomes unfeasible without a support system to help us in the matter of how will we discover, identify, and interact with the objects and devices. The same occurs to stream analytics, to make the IoT useful, we need a system that support Analytics of Things. This will mean new ways to analyze streaming data continuously.

For device discovery challenge, we presented the Context-based Search Engine (COBASEN). COBASEN provides a repository for IoT software engineers to search and select their "Things", whether they be Sensor/Actuator Devices, virtual computational elements. It allow software engineers to discover them using a set of search techniques. An interface helps in the selection and creation of additional output files (specification file in this case), for the use of "Things" that best suited for the application requirements. COBASEN experimental results demonstrated that it is efficient and can help users in the development of their IoT applications. Besides, IoT environments can benefit from the use of COBASEN since is difficult for an outsider of the IoT environment to understand the complex characteristics of the domain, which can be facilitated by the context-based and integration-driven characteristics of the systems of COBASEN.

Due the stream analytics challenge, we presented the Decision Management System (DMS). DMS use device data when it is still in motion, extracting valuable information from it through Complex Event Processing (CEP), allowing software engineers to deploy business rules and make near real time decision. DMS experimental results demonstrated that it is able to be used as an decision support system for streaming analytics towards scenarios with large numbers of devices, rules and facts, as the IoT scenarios. Further to this, it can reduces application processing overhead and networking exhaust.

We also presented similar work in order to identify how they manage the near real-time processing for decision management. We could see that none of those mentioned related works allow streaming analytics and intelligence automated action on fast-moving data towards the pre-selection of "Things" that generate it. The framework characteristics of pre-selection of "things" that will provide the data streams, to further develop the business rules provides a new way to see how the business rules and decision-making will be made towards the Internet of Things.

6.3 Future Work

In future, we are planning to improve COBASEN context module and search engine. In the context module we will explore techniques to analyze and categorize devices using ontologies and secondary context. In the search engine must be develop functions to improve the querying (e.g. pagination, and filtering) and the security (authentication and access control).

As we are not using pagination pagination technique in the COBASEN search engine, we believe that it will enhance the search engine response time. Further, a filtering technique will allow to filter query results according to a custom filtering process, which is a way to apply additional data restrictions. Some interesting use cases of filters are: security (e.g. returns only devices where the logged user has credentials), temporal data (e.g. view only last month's data), and population filter (e.g. search limited to a given category).

In DMS we are planning to expand COMPaaS and DMs capabilities, going beyond send sophisticated reports to applications and start to work through writing cycles in actuators devices.

In the construction of the rules must be expanded the support and access to the system environment and classes, allowing software engineers to better describe the environments and relate more types of events. It will allow the construction of rules and conditions that take into account more information, beyond the specification values. Also related to the rule construction, there is the possibility to verify new ways to facilitate the construction of rules for persons that does not know how to build rules (e.g. a drag and drop interface).

BIBLIOGRAPHY

[1] Alam, S.; Chowdhury, M. M.; Noll, J. "Senaas: An event-driven sensor virtualization approach for internet of things cloud". In: International Conference on Networked Embedded Systems for Enterprise Applications, 2010, pp. 1–6.

[2] Amaral, L. A.; de Matos, E.; Tiburski, R. T.; Hessel, F.; Lunardi, W. T.; Marczak, S. "Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G". Springer International Publishing, 2016, chap. 3, pp. 333–367.

[3] Amaral, L. A.; Tiburski, R. T.; de Matos, E.; Hessel, F. "Cooperative middleware platform as a service for internet of things applications". In: ACM Symposium on Applied Computing, 2015, pp. 488–493.

[4] Apache Software Fundation. "Apache Hadoop". Accessed: 2016-01-05, In: https://hadoop.apache.org/.

[5] Apache Software Fundation. "Apache Lucene". Accessed: 2016-01-03, In: https://lucene.apache.org/.

[6] Ashton, K. "That 'internet of things' thing", *RFiD Journal*, vol. 22–7, 2009, pp. 97–114.

[7] Atzori, L.; Iera, A.; Morabito, G. "The internet of things: A survey", *Computer Networks*, vol. 54–15, 2010, pp. 2787 – 2805.

[8] Bali, M. "Drools JBoss Rules 5.0 Developer's Guide". Packt Publishing, 2009, 320p.

[9] Bandyopadhyay, D.; Sen, J. "Internet of things: Applications and challenges in technology and standardization", *Wireless Personal Communications*, vol. 58–1, 2011, pp. 49–69.

[10] Bandyopadhyay, S.; Sengupta, M.; Maiti, S.; Dutta, S. "Role of middleware for internet of things: A study", *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 2–3, 2011, pp. 94–105.

[11] Blomqvist, P.; Persson, H.; Vinterbäck, J.; et al.. "Self-heating in storages of wood pellets." In: World Bioenergy Conference and Exhibition on Biomass for Energy, Jönköping, Sweden, 27-29 May 2008., 2008, pp. 172–176.

[12] Borgia, E. "The internet of things vision: Key features, applications and open issues", *Computer Communications*, vol. 54, 2014, pp. 1–31.

[13] Chen, M.; Leung, V. C. M.; Hjelsvold, R.; Huang, X. "Smart and interactive ubiquitous multimedia services", *Computer Communications*, vol. 35–15, 2012, pp. 1769 – 1771.

[14] de Matos, E.; Lunardi, W.; Tiburski, R.; Amaral, L.; Marczak, S.; Hessel, F. "Context-based framework to discovery, search, and selection of computing devices in the internet of things". In: Congresso da Sociedade Brasileira de Computação 2015 - SEMISH, 2015, pp. 1–8.

[15] Digital Enterprise Research Institute. "Linked sensor middleware (LSM)". Accessed: 2016-01-03, In: https://www.deri.ie/.

[16] Ding, Z.; Gao, X.; Guo, L.; Yang, Q. "A hybrid search engine framework for the internet of things based on spatial-temporal, value-based, and keyword-based conditions". In: IEEE International Conference on Green Computing and Communications (GreenCom), 2012, pp. 17–25.

[17] Dong, L.; Wang, D.; Sheng, H. "Design of rfid middleware based on complex event processing". In: IEEE Conference on Cybernetics and Intelligent Systems, 2006, pp. 1–6.

[18] Friedman-Hill, E. "Jess in Action: Rule-based Systems in Java". Manning, 2003, 480p.

[19] Gama, K.; Touseau, L.; Donsez, D. "Combining heterogeneous service technologies for building an Internet of Things middleware", *Computer Communications*, vol. 35–4, 2012, pp. 405 – 417.

[20] Giusto, D.; Iera, A.; Morabito, G.; Atzori, L. "The Internet of Things". Springer New York, 2010, 442p.

[21] Grosky, W.; Kansal, A.; Nath, S.; Liu, J.; Zhao, F. "Senseweb: An infrastructure for shared sensing", *IEEE MultiMedia*, vol. 14–4, Oct 2007, pp. 8–13.

[22] GSN Team. "Global Sensor Network". Accessed: 2016-01-03, In: http://sourceforge.net/apps/trac/gsn/.

[23] Guinard, D.; Trifa, V.; Karnouskos, S.; Spiess, P.; Savio, D. "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services", *IEEE Transactions on Services Computing*, vol. 3–3, 2010, pp. 223–235.

[24] Guinard, D.; Trifa, V.; Wilde, E. "A resource oriented architecture for the web of things". In: Internet of Things (IOT), 2010, pp. 1–8.

[25] IBM. "SPSS Analytical Decision Management". Accessed: 2016-02-16, In: http://www-03.ibm.com/software/products/en/decision-managment-linux-system-z.

[26] jBoss Developer. "Drools Expert User Guide". Accessed: 2016-01-10, In: https://docs.jboss.org/drools/release/6.3.0.Final/drools-docs/html_single/.

[27] Jing, Q.; Vasilakos, A.; Wan, J.; Lu, J.; Qiu, D. "Security of the internet of things: perspectives and challenges", *Wireless Networks*, vol. 20–8, 2014, pp. 2481–2501.

[28] Jun, C.; Chi, C. "Design of complex event-processing ids in internet of things". In: Sixth International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), 2014, pp. 226–229.

[29] Larsson, S. H.; Lestander, T. A.; Crompton, D.; Melin, S.; Sokhansanj, S. "Temperature patterns in large scale wood pellet silo storage", *Applied Energy*, vol. 92–0, 2012, pp. 322 – 327.

[30] Le-Phuoc, D.; Quoc, H. N. M.; Parreira, J. X.; Hauswirth, M. "The linked sensor middleware– connecting the real world and the semantic web", *Semantic Web Challenge*, vol. 152, 2011.

[31] LogMeIn. "Xively". Accessed: 2016-01-03, In: http://xively.com/.

[32] Luckham, D. C. "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems". Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, 400p.

[33] Lunardi, W.; Amaral, L.; Marczak, S.; Hessel, F.; Voos, H. "Automated decision support iot framework". In: IEEE Conference on Emerging Technologies Factory Automation (ETFA), 2016, pp. 1–8.

[34] Lunardi, W.; de Matos, E.; Tiburski, R.; Amaral, L.; Marczak, S.; Hessel, F. "Context-based search engine for industrial iot: Discovery, search, selection, and usage of devices". In: IEEE Conference on Emerging Technologies Factory Automation (ETFA), 2015, pp. 1–8.

[35] Lunardi, W.; Marczak, S.; Amaral, L.; Hessel, F. "Discovery and usage of computing devices in iot environments". In: 2nd Latin-American School on Software Engineering, 2015, pp. 1–8.

[36] Mark Proctor. "R.I.P. RETE time to get PHREAKY". Accessed: 2016-02-11, In: http://blog.athico.com/2013/11/rip-rete-time-to-get-phreaky.html.

[37] Matos, E. D.; Amaral, L. A.; Tiburski, R.; Lunardi, W.; Hessel, F.; Marczak, S. "Context-aware system for information services provision in the internet of things". In: IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), 2015, pp. 1–4.

[38] Michael Chui, Markus Löffler, and Roger Roberts. "The Internet of Things". Accessed: 2016-02-15, In: http://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things.

[39] Nath, S.; Liu, J.; Zhao, F. "Sensormap for wide-area sensor webs", *Computer*, vol. 40–7, 2007, pp. 90–93.

[40] Ostermaier, B.; Romer, K.; Mattern, F.; Fahrmair, M.; Kellerer, W. "A real-time search engine for the web of things". In: Internet of Things (IOT), 2010, 2010, pp. 1–8.

[41] Pentaho Corporation. "Pentaho IoT Platform". Accessed: 2016-02-16, In: http://www.pentaho.com/internet-of-things-analytics.

[42] Perera, C.; Zaslavsky, A.; Christen, P.; Compton, M.; Georgakopoulos, D. "Context-aware sensor search, selection and ranking model for internet of things middleware". In: IEEE International Conference on Mobile Data Management (MDM), 2013, pp. 314–322.

[43] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. "Ca4iot: Context awareness for internet of things". In: IEEE International Conference on Green Computing and Communications (GreenCom), 2012, pp. 775–782.

[44] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. "Context aware computing for the internet of things: A survey", *IEEE Communications Surveys & Tutorials*, vol. 16–1, 2014, pp. 414–454.

[45] Rajaraman, A.; Ullman, J. D. "Mining of Massive Datasets". New York, NY, USA: Cambridge University Press, 2011, 513p.

[46] Raskino, M.; Fenn, J.; Linden, A. "Extracting value from the massively connected world of 2015", *Gartner Res*, vol. 1, 2005, pp. 4.

[47] Robins, D. "Complex event processing". In: Second International Workshop on Education Technology and Computer Science. Wuhan, 2010, pp. 1–10.

[48] Roy, J. "Rule-based expert system for maritime anomaly detection". In: SPIE Defense, Security, and Sensing, 2010, pp. 76662N–76662N.

[49] SAS Institute. "SAS: Business Analytics and Business Intelligence Software". Accessed: 2016-02-16, In: http://www.sas.com.

[50] Stonebraker, M.; Çetintemel, U.; Zdonik, S. "The 8 requirements of real-time stream processing", *ACM SIGMOD Record*, vol. 34–4, 2005, pp. 42–47.

[51] Sundmaeker, H.; Guillemin, P.; Friess, P.; Woelfflé, S.; for the Information Society, E. C. D.-G.; Media. "Vision and Challenges for Realising the Internet of Things". Publications Office of the European Union, 2010, 229p.

[52] Tan, C. C.; Sheng, B.; Wang, H.; Li, Q. "Microsearch: A search engine for embedded devices used in pervasive computing", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9–4, 2010, pp. 43.

[53] Theodoridis, E.; Mylonas, G.; Chatzigiannakis, I. "Developing an iot smart city framework". In: Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on, 2013, pp. 1–6.

[54] Vermesan, O.; Friess, P.; Guillemin, P.; Gusmeroli, S.; Sundmaeker, H.; Bassi, A.; Jubert, I. S.; Mazura, M.; Harrison, M.; Eisenhauer, M.; et al.. "Internet of things strategic research roadmap", *Internet of Things-Global Technological and Societal Trends*, vol. 1, 2011, pp. 9–52.

[55] Wang, H.; Tan, C.; Li, Q. "Snoogle: A search engine for pervasive environments", *IEEE Transactions on Parallel and Distributed Systems*, vol. 21–8, Aug 2010, pp. 1188–1202.

[56] Wang, W.; Sung, J.; Kim, D. "Complex event processing in epc sensor network middleware for both rfid and wsn". In: IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008, pp. 165–169.

[57] Wu, E.; Diao, Y.; Rizvi, S. "High-performance complex event processing over streams". In: International Conference on Management of Data, 2006, pp. 407–418.

[58] Yap, K.-K.; Srinivasan, V.; Motani, M. "Max: Human-centric search of the physical world". In: 3rd International Conference on Embedded Networked Sensor Systems, 2005, pp. 166–179.

[59] Zobel, J.; Moffat, A. "Inverted files for text search engines", *ACM Computing Surveys (CSUR)*, vol. 38–2, 2006, pp. 6.