

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL  
FACULTY OF INFORMATICS  
GRADUATE PROGRAM IN COMPUTER SCIENCE**

**CONTEXT-AWARE  
INFORMATION SERVICES  
PROVISION FOR IOT  
ENVIRONMENTS**

**EVERTON DE MATOS**

Dissertation presented as partial requirement  
for obtaining the degree of Master in  
Computer Science at Pontifícia Universidade  
Católica do Rio Grande do Sul.

Advisor: Prof. Fabiano Passuelo Hessel  
Co-Advisor: Prof. Leonardo Albernaz Amaral

**Porto Alegre  
2016**



## **Dados Internacionais de Catalogação na Publicação (CIP)**

M433c Matos, Everton de

Context-aware information services provision for IoT environments / Everton de Matos. – 2016.

78 f.

Dissertação (Mestrado) – Faculdade de Informática, PUCRS.

Orientador: Prof. Fabiano Passuelo Hessel

Coorientador: Prof. Leonardo Albernaz Amaral

1. Middleware - Sistemas Distribuídos. 2. Internet das Coisas. 3. Informática. I. Hessel, Fabiano Passuelo. II. Amaral, Leonardo Albernaz. III. Título.

CDD 23 ed. 004.678

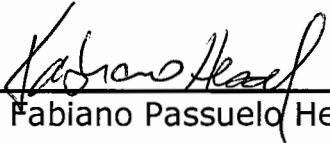
**Loiva Duarte Novak CRB 10/2079**  
**Setor de Tratamento da Informação da BC-PUCRS**



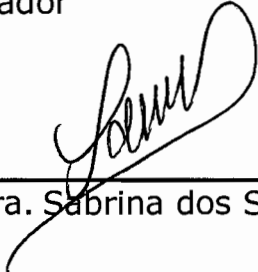



## TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "*Context-Aware Information Services Provision for IoT Environments*" apresentada por Everton de Matos como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 29 de março de 2016 pela Comissão Examinadora:

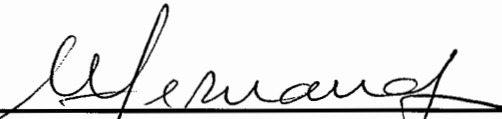
  
Prof. Dr. Fabiano Passuelo Hessel - PPGCC/PUCRS  
Orientador

  
Dr. Leonardo Albernaz Amaral - PPGEE/PUCRS  
Coorientador

  
Profa. Dra. Sabrina dos Santos Marczak - PPGCC/PUCRS

  
Prof. Dr. Jorge Luis Victória Barbosa - UNISINOS

Homologada em 02/06/2016, conforme Ata No. 011... pela Comissão Coordenadora.

  
Prof. Dr. Luiz Gustavo Leão Fernandes  
Coordenador.



Dedico este trabalho a meus pais.





“I reject your reality and substitute my own.”  
(Adam Savage)



## ACKNOWLEDGMENTS

Agradeço primeiramente aos meus pais, Alberto e Rosemari, e a minha namorada, Luana, pelo apoio, paciência e compreensão durante minha pesquisa. Também os agradeço por sempre me incentivarem a estudar e correr atrás de meus sonhos, apoiando todas as minhas decisões.

Agradeço aos meus amigos, e colegas de mestrado, Ramão e Willian pelo companherismo e parceria em todas as etapas do curso.

Agradeço aos meus amigos, Jean, Rafael Maroso e Rodrigo, por estarem presente em momentos difíceis, encurtando as distâncias entre Porto Alegre, Passo Fundo e Belo Horizonte.

Agradeço ao meu orientador, Fabiano Hessel, pelo incentivo e confiança no meu trabalho, além de me motivar a buscar sempre o melhor de mim. Agradeço também ao meu co-orientador, Leonardo Amaral, pelos conselhos, conversas e pelo conhecimento compartilhado, fazendo com que eu evoluísse como pesquisador.



# PROVIMENTO DE SERVIÇOS DE INFORMAÇÃO CIENTES DE CONTEXTO PARA AMBIENTES IOT

## RESUMO

O paradigma da computação Internet das Coisas (IoT) irá conectar bilhões de dispositivos ao redor do mundo em um futuro próximo. Nos últimos anos, a IoT vem ganhando mais atenção. Esse paradigma tornou-se popular por embarcar redes móveis e pelo seu poder de processamento em uma vasta gama de dispositivos computadorizados utilizados na vida cotidiana de muitas pessoas.

Um elemento importante da IoT é o *middleware*, que é um sistema capaz de abstrair a gestão de dispositivos e prover serviços baseados nestes dispositivos. Os serviços providos são usados por aplicações para obter informações do ambiente. Desta forma, existem muitas pesquisas relacionadas com o desenvolvimento de *middleware* que abordam não só interoperabilidade dos dispositivos, mas também a característica de ciência de contexto. Ciência de contexto é uma característica importante dos sistemas da IoT. Esta característica facilita o descobrimento, compreensão e armazenamento de informações relevantes relacionadas aos dispositivos. Estas informações podem ser usadas para prover serviços e tomada de decisão com base no contexto do ambiente.

Neste sentido, este trabalho apresenta o Context-Aware System (CONASYS), que é um sistema para provimento de serviços de informação contextualizada sobre dispositivos da IoT em ambientes heterogêneos. O sistema é acoplado ao middleware COMPaaS e é capaz de agir conforme o ambiente que está inserido. A arquitetura do CONASYS é apresentada em detalhes, assim como os testes realizados. Nosso objetivo é prover serviços contextualizados que atendam às necessidades dos usuários que não possuem conhecimento específico do ambiente, melhorando assim a Qualidade da Experiência (QoE).

**Palavras Chave:** Internet of Things, Context-Awareness, Middleware, Context life-cycle, Acquisition, Modelling, Reasoning, Distribution, COMPaaS.



# CONTEXT-AWARE INFORMATION SERVICES PROVISION FOR IOT ENVIRONMENTS

## ABSTRACT

The computing paradigm called Internet of Things (IoT) will connect billions of devices deployed around the world together in a near future. In the last years, IoT is gaining more attention. This paradigm has become popular by embedding mobile network and processing power into a wide range of physical computing devices used in everyday life of many people.

An important element of the IoT is a middleware, which is a system able to abstract the management of physical devices and to provide services based on the information from these devices. The services provided are used by application clients to perform queries and obtain environmental information. In this way, it is already a subject in literature studies that address middleware systems not only interoperability of devices, but also context awareness feature. Context-aware is an important feature of IoT systems. This feature makes easy to discover, understand, and store relevant information related to devices. This information can be used for a refined provision of services based on the environment context and also for decision making.

This work aims to present the Context-Aware System (CONASYS), that is a system to provide services of contextualized information about IoT devices in heterogeneous environments. The system is attached to COMPaaS IoT middleware and is able to act accordingly to the environment that it is inserted. We present in details the architecture of CONASYS, the technical issues related to the implementation of the system and perform some tests based in a real-world scenario. We also present some related work. Our objective is to provide a well-defined range of contextualized services that meet the users needs without specific knowledge of the environment, improving users Quality of Experience (QoE).

**Keywords:** Internet of Things, Context-Awareness, Middleware, Context life-cycle, Acquisition, Modelling, Reasoning, Distribution, COMPaaS.



## LIST OF ACRONYMS

API – Application Programming Interface  
CEP – Complex Event Processing  
COMPAAS – Cooperative Middleware Platform as a Service  
CONASYS – Context-Aware System  
ECA – Event Condition-Action  
GPS – Global Positioning System  
HTTP – Hypertext Transfer Protocol  
IOT – Internet of Things  
LHS – Left Hand Side  
M2M – Machine-to-Machine  
ODMRP – On-Demand Multicast Routing Protocol  
ORM – Object Role Modelling  
QOE – Quality of Experience  
REST – Representational State Transfer  
RFID – Radio-Frequency IDentification  
RHS – Right Hand Side  
SOA – Service-Oriented Architecture  
SOAP – Simple Object Access Protocol  
SQL – Structured Query Language  
TCP – Transmission Control Protocol  
UDP – User Datagram Protocol  
UML – Unified Modelling Language  
URI – Uniform Resource Identifier  
WOT – Web of Things  
WSN – Wireless sensor networks  
XML – Extensible Markup Language



## LIST OF FIGURES

Figure 1.1 – City pollution scenario. . . . .	27
Figure 2.1 – Generic layered architecture of the Internet of Things. . . . .	29
Figure 2.2 – Structure of a Rule. . . . .	37
Figure 4.1 – COMPaaS architecture overview. . . . .	48
Figure 4.2 – CONASYS layers and modules overview. . . . .	50
Figure 4.3 – Case study example. . . . .	52
Figure 4.4 – Interface that shows CONASYS available services. . . . .	53
Figure 4.5 – CONASYS activity flow. . . . .	54
Figure 4.6 – The data life-cycle to provide an information service. . . . .	55
Figure 4.7 – Example of an XML file with information about a device. . . . .	56
Figure 4.8 – Example of a Drools rule. . . . .	57
Figure 4.9 – Overview of CONASYS database modelling. . . . .	58
Figure 4.10 – CONASYS access through subscription. . . . .	59
Figure 4.11 – CONASYS access through query. . . . .	59
Figure 4.12 – Example of an XML that CONASYS must interpret. . . . .	60
Figure 4.13 – Phases of the Processing Cycle. . . . .	60
Figure 4.14 – Example of a specification report. . . . .	61
Figure 5.1 – Time taken by the CONASYS function to process rules. . . . .	64
Figure 5.2 – CONASYS background functions execution time variation. . . . .	66
Figure 5.3 – Measuring time of real-time processing function divided in three phases. . . . .	68
Figure 5.4 – Processing Cycle comparison between real-time processing and no real-time flow. . . . .	70



## LIST OF TABLES

Table 3.1 – IoT-middleware comparison. . . . .	39
Table 3.2 – Context-aware systems comparison. . . . .	40
Table 4.1 – Comparison between real-time processing and no real-time Processing Cycle flow. . . . .	61
Table 5.1 – Execution time (ms) for modelling the knowledge base. . . . .	64
Table 5.2 – Execution time (ms) of the background functions. . . . .	66
Table 5.3 – Execution time (ms) of the real-time processing function. . . . .	67
Table 5.4 – Execution time (ms) of the Processing Cycle with real-time and no real-time flow. . . . .	69



# CONTENTS

<b>1</b>	<b>INTRODUCTION</b> .....	<b>25</b>
1.1	MOTIVATION .....	26
1.2	CONTRIBUTIONS .....	27
1.3	WORK SCOPE .....	28
1.4	OUTLINE .....	28
<b>2</b>	<b>THEORETICAL BACKGROUND</b> .....	<b>29</b>
2.1	INTERNET OF THINGS .....	29
2.2	SERVICE-ORIENTED IOT MIDDLEWARE .....	30
2.3	CONTEXT AWARENESS .....	32
2.4	CONTEXT LIFE-CYCLE .....	32
2.4.1	CONTEXT ACQUISITION .....	33
2.4.2	CONTEXT MODELLING .....	34
2.4.3	CONTEXT REASONING .....	36
2.4.4	CONTEXT DISTRIBUTION .....	38
<b>3</b>	<b>RELATED WORK</b> .....	<b>39</b>
3.1	IOT MIDDLEWARE SYSTEMS .....	39
3.2	CONTEXT-AWARE SYSTEMS .....	40
3.2.1	SYSTEMS APPROACHES .....	42
3.2.2	OUR SYSTEM APPROACH .....	44
<b>4</b>	<b>PROPOSED APPROACH</b> .....	<b>47</b>
4.1	REFERENCE PLATFORM .....	47
4.2	CONASYS: CONTEXT-AWARE SYSTEM .....	49
4.2.1	ARCHITECTURAL OVERVIEW .....	50
4.2.2	SYSTEM USABILITY .....	52
4.2.3	SERVICES PROVISION .....	54
4.2.4	TECHNICAL OVERVIEW .....	56
<b>5</b>	<b>EVALUATION</b> .....	<b>63</b>
5.1	SCENARIO 1 - MODELLING THE KNOWLEDGE BASE .....	63
5.2	SCENARIO 2 - EVENT AND DEVICE MODELLING .....	65
5.3	SCENARIO 3 - REAL-TIME PROCESSING .....	67

5.4	SCENARIO 4 - PROCESSING CYCLE .....	68
<b>6</b>	<b>CONCLUSION AND FUTURE WORK .....</b>	<b>71</b>
6.1	PUBLICATIONS .....	71
6.2	CONCLUSIONS .....	71
6.3	FUTURE WORK .....	73
	<b>REFERENCES .....</b>	<b>75</b>



# 1. INTRODUCTION

During the past few years in the area of wireless communications and networking, a novel computing paradigm called Internet of Things (IoT) has gained increasingly attention in academia and industry [39]. By embedding mobile networking and information processing capability into a wide array of gadgets and everyday computing items, and enabling new forms of communication among people and things, and between things themselves, IoT has been adding new dimensions to the world of information and communication technology [6].

Unquestionably, the main strength of IoT is the high impact it has had on several aspects of everyday-life and behavior of potential users. From the point of view of a private user, the most obvious effects of IoT have been in both working and domestic fields. In this context, assisted living, e-health, enhanced learning, and smart cities are only few examples of possible application scenarios in which IoT has been playing a leading role. Similarly, from the perspective of business users, the most apparent consequences have been equally visible in fields such as, automation and industrial manufacturing, logistics, business/process management, and intelligent transportation of people and goods [4].

In IoT, when large numbers of sensor and actuator devices are deployed and start generating data, the traditional device-oriented application approach (i.e., connect sensors directly to applications individually and manually) becomes infeasible, since all the complex responsibility regarding the management of IoT devices and their data becomes an application role, which can bring scalability problems for large scale scenarios. In order to mitigate this inefficiency, significant amounts of middleware solutions have been introduced by researchers [35].

Developing middleware solutions in the domain of IoT is an active area of research. In this way, it is available in literature a large number of studies on building up middleware systems addressing interoperability of devices, adaptation, device discovery and management, scalability, management of large data volumes, privacy, and security aspects of IoT environments [8]. Another feature of middleware systems is the context-awareness, that is a challenge of this area according to [35].

As we are moving towards the popularization of IoT, the number of IoT devices deployed around the world is growing at a rapid pace. It is a common sense that these sensors will generate a large amount of data [45], and, unless we can analyse, interpret, and understand these data, they will keep useless and with no meaning to improve the society. Context-aware computing has played an important role in tackling this challenge in previous paradigms, such as mobile and pervasive computing [35], which lead us to believe that it would continue to be successful in the IoT paradigm as well. Context-aware computing allows us to discover and store context information linked to devices data so the interpretation can be done easily and more meaningfully. In addition, understanding context makes it easier to perform machine to machine (M2M) communication as it is also a core element in the IoT vision [35].

In context-aware environments, there is a cycle for the creation of the information that will be delivered to the users, which consists of data acquisition, information modeling, reasoning, and sharing or distribution of contextualized information. Thus, the architecture of a context-aware system must be able to provide services for the integration and automation of all stages of the information cycle. This work aimed to develop a system mechanism to provide context-aware information services to IoT environments. This mechanism is attached to an IoT middleware system which is the reference platform used in this work [3].

Other goal of this work is to provide a system able to produce context-aware information in order to give semantic meaning or context to entities and their data in IoT environments. Entities are persons, places, or objects that are considered relevant to the interaction between users and applications, including the users and applications themselves. The main intention is to avoid the manual user intervention in the interpretation of the data and also facilitates the systems/entities interactions. We also present in this work a study of the definitions, characteristics, and some components of IoT environments. Besides, we make a research on context-aware IoT systems, which is an important area of the IoT. The analysis of works regarding IoT middleware and context-aware systems is important to identify the next trends in IoT, as well as discover gaps that can be mitigated.

## 1.1 MOTIVATION

IoT tends to grow and gain space on a global level [4]. As a consequence, the data generated by any type of IoT devices will increase. Besides, even though there have been some IoT systems that provide an amount of context-aware functionality, they do not satisfy all the context requirements demanded by the IoT, since IoT is an evolving computing paradigm that blends couples of heterogeneous system technologies, which difficult the unification of the context management [35].

In context-awareness, there are some steps in order to produce context information. In each one of these steps there are different consolidated techniques. No standard architecture for contextualization was seen. Therefore, the definition of a standard architecture to provide context-aware information services is an open issue and remains a topic of extreme importance.

A real example of a context-aware system usability can be seen in the Figure 1.1. This scenario shows an IoT environment that monitors the pollution of a fictional city. In this example, for the information service that monitors the pollution of the city is necessary to know data of ground, air and water elements. The user makes a high level request informing the desire to know if the city is polluted. The context-aware system interprets this request and gather data related to the city pollution. Without the context-aware system, the user must know what devices are real parameters to indicate the city pollution, and needs to make a direct request to each device, as well as a fusion of the meaningful data. With the context-aware system, this process is summarized with a direct request because the system knows the environment that it is inserted in.

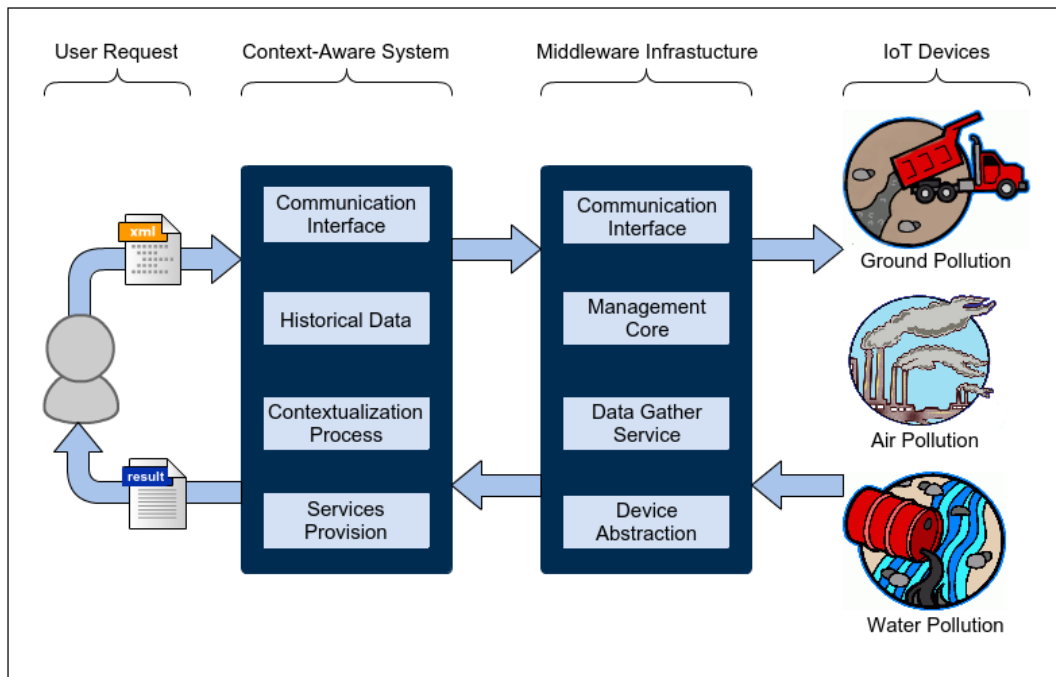


Figure 1.1 – City pollution scenario.

In this work, we aim to identify the context-aware computing requirements and characteristics that are required by an ideal IoT system. Furthermore, we propose a solution able to provide context-aware information services to IoT users and applications in order to satisfy the identified requirements.

Another motivation is the fact that the COMPaaS middleware to be used as reference platform for this work does not provide context-based services [3], which hinders its diffusion for many IoT application areas.

## 1.2 CONTRIBUTIONS

The main contributions of this work are:

- A detailed state of the art of the Internet of Things and contextualization in IoT environments, that presents the definitions and techniques of each topic. Moreover, a detailed state of the art of context-aware systems, the features and characteristics of the systems, in addition to a discussion comparing them.
- The definition of a context-aware architecture composed of context acquisition, context modelling, context reasoning and context distribution (i.e., the context life-cycle). This architecture must be possible to be used in an IoT environment.
- The implementation of a context-aware system to proof the architecture, which is able to use the techniques of context life-cycle in order to produce contextualized information and provide this information to the users in form of services.

- The implementation of the context-aware system features in the COMPaaS middleware (i.e., in our reference platform). In this sense, COMPaaS can provide contextualized information services with the context-aware feature.

### **1.3 WORK SCOPE**

This is a research work focused on context-aware in IoT environments in order to provide contextualized information services. The goal is to present the four steps of contextualization that must be used in the IoT context-aware systems to provide information services. In addition, this work presents the implementation of the context-aware approach as a system, called CONASYS, to be attached to the COMPaaS IoT middleware and shows the advantages of data contextualization in IoT environments. This work can be applied to different application scenarios since the context-aware services are requested in every applications. In our approach, the application scenarios can be identified as domains, as healthcare, industry, smart city, just to name a few. The system can be deployed in different domains, since the domain depends on the rules that are inserted in the system. In this sense, the proposed system is domain-specific and for our experimentation and tests we use different domains examples.

### **1.4 OUTLINE**

The remainder of this Dissertation is organized as follows. Chapter 2 presents theoretical references that are used in this work, such as definitions of IoT, middleware, and context-aware. Chapter 3 presents two types of related work: IoT middleware and context-aware systems. Chapter 4 presents the characteristics of our system, an architectural overview, an example of use and the technical definitions of the system. Chapter 5 presents the evaluation and tests of the systems. Finally, Chapter 6 presents the conclusions and future directions.

## 2. THEORETICAL BACKGROUND

This Chapter introduces concepts and definitions that are mentioned in this work. Section 2.1 presents the concept of the Internet of Things. Section 2.2 provides an overview of middleware technology. Section 2.3 provides definitions of the context-aware area and the related challenges. Finally, Section 2.4 shows the context life-cycle process needed in order to obtain context information.

### 2.1 INTERNET OF THINGS

Internet of Things (IoT) is a novel computing paradigm that is rapidly gaining space in scenarios of modern communication technologies. The idea of IoT is the pervasive presence of a variety of things or objects (e.g., RFID tags, sensors, actuators, smart phones, smart devices, etc), that are able to interact with each other and cooperate with their neighbors to reach common goals through unique addressing schemes and reliable communication media over the Internet [4] [21].

During the past decades, IoT has gained significant attention in academia as well as in industry. The main reasons behind this interest are the capabilities that IoT is able to offer. It promises to create a world where all the objects around us (also called smart objects) are connected to the Internet and communicate with each other with minimum human intervention [27]. The ultimate goal is to create a better world for human beings, where objects around us know what we like, what we want, and what we need and act accordingly without explicit instructions [19].

Implementation of IoT environments is usually based on a standard system architecture consisting of several layers [6]: from the data acquisition layer at the bottom to the application layer at the top. Figure 2.1 presents a generic architecture for IoT, adapted from [4].

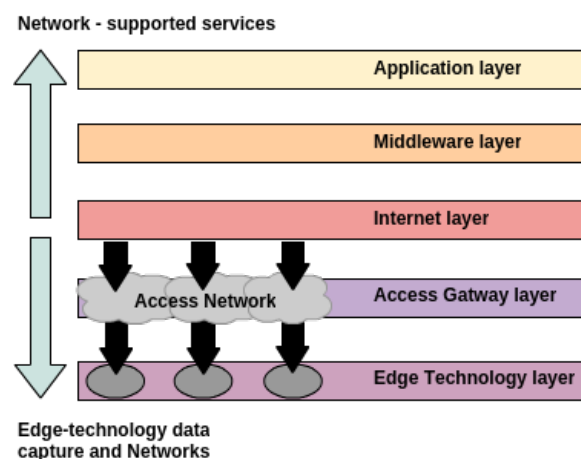


Figure 2.1 – Generic layered architecture of the Internet of Things.

The two layers at the bottom contribute to data capturing while the two layers at the top are responsible for data utilization in applications. Next, we present the functionality of these layers [6]:

- **Edge layer:** This hardware layer consists of sensor networks, embedded systems, RFID tags and readers or other IoT devices in different forms. These entities are the primary data sources deployed in the field. Many of these hardware elements provide identification and information storage (e.g. RFID tags), information collection (e.g. sensor networks), information processing (e.g. embedded edge processors), communication, control and actuation. However, identification and information collection are the primary goals of these devices, leaving the processing activities for the upper layers.
- **Access gateway layer:** The first stage of data handling happens at this layer. It takes care of message routing, publishing and subscribing, and also performs cross platform communication, if required.
- **Middleware layer:** This layer acts as an interface between the hardware layer at the bottom and the application layer at the top. It is responsible for critical functions such as device management and information management, and also takes care of issues like data filtering, data aggregation, semantic analysis, access control and information discovery.
- **Application layer:** This layer at the top of the stack is responsible for the delivery of various services to different users/applications in IoT environments. The applications can be from different industry verticals such as: manufacturing, logistics, retail, environment, public safety, healthcare, food and drug etc.

In the last years, researchers have proposed and analyzed the advantages of using middleware systems in the existing solutions for IoT (e.g., management of devices, interoperability). One of the main roles of an IoT middleware is to provide continuous communications among different devices and use communication channels characterized by heterogeneous technologies. Furthermore, in some cases the connectivity could be intermittent due to mobility or interferences. In this sense, middleware systems are used to ensure interoperability among several different devices and applications, for example: medical instruments, body and environmental sensors, logic applications, graphic interfaces, etc [9].

## 2.2 SERVICE-ORIENTED IOT MIDDLEWARE

IoT middleware is a software layer or a set of sub-layers interposed between access gateway 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 her focus,

which is the development of specific applications enabled by IoT infrastructures [4]. 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.

IoT middleware systems are required for various reasons. Some reasons are described next [7]: (i) Difficult to define and enforce a common standard among all the diverse devices belonging to diverse domain in IoT; (ii) Middleware acts as a bond joining the heterogeneous components together; (iii) Applications of diverse domains demand abstraction/adaptation layer; and (iv) Middleware provides API (application programming interface) for physical layer communications, and required services to the applications, hiding all the details of diversity.

Many of the middleware system architectures proposed comply with the Service-Oriented Architecture (SOA) approach. The SOA standard has been used as a viable choice to cope with middleware since it helps to ensure high levels of systems interoperability, providing system services that are based on devices and used by applications. The adoption of SOA principles allows the decomposition of complex systems into applications consisting of a system of simpler and well-defined components. In a SOA architecture, each system offers its functionality as standard services. Moreover, a SOA architecture supports open and standardized communication through all layers of web services that can be used in the IoT [4]. The architecture of a SOA-based IoT middleware is composed of following items:

- **Services Provision:** This is the highest level middleware layer in which services are available to be used by applications. In this layer there is no notion of devices and the only visible assets are services. Each available service has a respective infrastructure of devices connected to the middleware.
- **Management Core:** This is the main layer of the middleware. It is composed of functions that allow the management of each device of the environment. The basic set of functions encompasses: dynamic discovery of device, status monitoring, service configuration, data management and context management. This layer can provide a catalogue of services to the upper layer. The upper layer can then compose complex services by joining services provided at this layer.
- **Devices Abstraction:** This is the lowest layer of the middleware and facilitates the management of devices through a common language and procedures. This layer is useful because IoT is composed of heterogeneous set of devices, each one providing specific functions accessible through its own addresses.

The services provision characteristic of a SOA-based IoT middleware makes easy the implementation of context-aware feature. In this way, the application/user can request a service without knowledge of the devices infrastructure.

## 2.3 CONTEXT AWARENESS

Context is considered any information that can be used to characterize the situation of an entity [1]. Entity is a person, place, or computing device (also called thing) that is relevant to the interaction between a user and an application, including the user and the application themselves. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task [1]. In this way, an IoT ecosystem requires a context-aware mechanism to be aware of the environment situation in order to help the user in the most useful way. In various cases, context-awareness becomes a feature of an IoT middleware. For example, when a middleware works based on the environment that it was inserted in.

Several researchers have identified different context types based on different perspectives. Abowd et al. [1] introduced one of the leading mechanisms of defining context types. They identified location, identity, time, and activity as the primary context types. Further, they defined secondary context as the context that can be found using primary context [35]. For example, given primary context such as a person's identity, we can acquire many pieces of related information such as phone numbers, addresses, email addresses, etc. Some examples defined by Perera et al. are:

- **Primary context:** Any information retrieved without using existing context and without performing any kind of sensor data fusion operations (e.g., GPS sensor readings as location information, the temperature of a sensor, the amount of oxygen in the air).
- **Secondary context:** Any information that can be computed using primary context. The secondary context can be computed by using sensor data fusion operations or data retrieval operations such as web service calls (e.g., identify the distance between two sensors by applying sensor data fusion operations on two raw GPS sensor values). Further, retrieved context such as phone numbers, addresses, email addresses, birthdays, list of friends from a contact information provider based on a personal identity as the primary context can also be identified as secondary context. For example, we can use the data of primary context as the information of a user and the location that he was to inferring the preferences of the user.

## 2.4 CONTEXT LIFE-CYCLE

A set of methods is mandatory in order to obtain the context of an entity. Furthermore, there is a set of actions, organized in phases, that characterizes the context life-cycle of an information. Perera et al. proposed a life-cycle and explained how acquisition, modelling, reasoning, and distribution of context should occur.



## 2.4.1 CONTEXT ACQUISITION

In the acquisition process, context needs to be acquired from various information sources (e.g., physical or virtual devices). The techniques used to acquire context can be varied based on responsibility, frequency, context source, sensor type, and acquisition process [35].

1) *Based on Sensor Types*: In general usage, the term 'sensor' is used to refer the tangible hardware sensor devices. However, among the technical community, sensors refer to as any data source that provides relevant context. Therefore, sensors can be divided into three categories: physical, virtual, and logical.

- Physical sensors: These are the most commonly used type of sensors. These sensors generate sensor data by themselves. Most of the devices we use today are equipped with a variety of sensor (e.g. temperature, humidity, microphone, touch).
- Virtual sensors: These sensors do not necessarily generate sensor data by themselves. Virtual sensors retrieve data from many sources and publish it as sensor data (e.g. calendar, contact number directory, twitter statuses, email and chat applications). These sensors do not have a physical presence.
- Logical sensors (also called software sensors): They combine physical sensors and virtual sensors in order to produce more meaningful information. A web service dedicated to providing weather information can be called a logical sensor.

2) *Based on Responsibility*: Context acquisition can be primarily accomplished using two methods [36]: push and pull.

- Push: The physical or virtual sensor pushes data to the data consumer which is responsible to acquiring sensor data periodically or instantly. Periodical or instant pushing can be employed to facilitate a publish and subscribe communication model.
- Pull: The data consumers make a request from the hardware sensor periodically or instantly to acquire data.

3) *Based on Frequency*: There are two different types: Instant and Interval.

- Instant: These events occur instantly. The events do not span across certain amounts of time. In order to detect this type of event, sensor data needs to be acquired when the event occurs. Both push and pull methods can be employed.
- Interval: These events span in a certain period of time. In order to detect this type of event, sensor data needs to be acquired periodically. Both push and pull methods can be employed.

4) *Based on Source*: Context acquisition methods can be organized into three categories [14] .

- Acquire directly from hardware sensor: In this method, context is directly acquired from the sensor by communicating with the hardware sensor and related APIs. Software drivers and libraries need to be installed locally.
- Acquire through a middleware infrastructure: In this method, sensor data (context) is acquired by middleware solutions. The applications can retrieve sensor data from the middleware and not from the hardware sensor directly.
- Acquire from context servers: In this method, context is acquired from several other context storages (e.g. databases, web services) via different mechanisms such as web service calls.

5) *Based on Acquisition Process*: Here are three ways to acquire context: sense, derive, and manually provided.

- Sense: The data is sensed through sensors, including the sensed data stored in databases (e.g. retrieve temperature from a sensor, retrieve appointments details from a calendar).
- Derive: The information is generated by performing computational operations on sensor data. These operations could be as simple as web service calls or as complex as mathematical functions running over sensed data (e.g. calculate distance between two sensors using GPS coordinates).
- Manually provided: Users provide context information manually via predefined settings options such as preferences (e.g. understanding that user does not like to receive event notifications between 10pm to 6am).

## 2.4.2 CONTEXT MODELLING

Context modelling is organized in two steps [10]. First, new context information needs to be defined in terms of attributes, characteristics, and relationships with previously specified context. In the second step, the outcome of the first step needs to be validated and the new context information needs to be merged and added to the existing context information repository. Finally, the new context information is made available to be used when needed.

The most popular context modelling techniques are surveyed in [13] and [41]. These surveys discuss a number of systems that have been developed based on the following techniques. Each technique has its own strengths and weaknesses.

1) *Key-Value Modelling*: In the key-value each data has a key. The key-value technique is an application oriented and application bounded technique that suits the purpose of temporary

storage such as less complex application configurations and user preferences. It models context information as key-value pairs in different formats such as text files and binary files. This is the simplest form of context representation among other techniques. They are easy to manage when they have smaller amounts of data. However, key-value modelling is not scalable and not suitable to store complex data structures.

2) *Markup Scheme Modelling (Tagged Encoding)*: It models data using tags. An example of tags can be seen as the fields of an eXtensible Markup Language (XML) file (e.g., `<field>`). Therefore, context is stored within tags. This technique is an improvement over the key-value modelling technique. The advantage of using markup tags is that it allows efficient data retrieval [13]. Markup schemas such as XML are widely used in almost all application domains to store data temporarily, transfer data among applications, and transfer data among application components. In contrast, markup languages do not provide advanced expressive capabilities to allow reasoning.

3) *Graphical Modelling*: It models context with relationships. Some examples of this modelling technique are Unified Modelling Language (UML) [43] and Object Role Modelling (ORM) [33]. Actual low-level representation of the graphical modelling technique could be varied. For example, it could be a SQL database, noSQL database, etc. Further, as we are familiar with databases, graphical modelling is a well known, easy to learn, and easy to use technique. Databases can hold massive amounts of data and provide simple data retrieval operations, which can be performed relatively quickly. In contrast, the number of different implementations makes it difficult with regards to interoperability.

4) *Object Based Modelling*: Object based (or object oriented) concepts are used to model data using class hierarchies and relationships. The Object Oriented paradigm promotes encapsulation and re-usability. As most of the high-level programming languages support object oriented concepts, modelling can be easily integrated into context-aware systems. Object based modelling is suitable to be used as a non-shared, code based, run-time context modelling, manipulation, and storage mechanism. Validation of object oriented designs is difficult due to the lack of standards and specifications.

5) *Logic Based Modelling*: Facts, expressions, and rules are used to represent information about the context. Rules are primarily used to express policies, constraints, and preferences. It provides much more expressive richness compared to the other previously models discussed. Therefore, reasoning is possible up to a certain level. Logic based modelling allows new high-level context information to be extracted using low-level context.

6) *Ontology Based Modelling*: The context is organised into ontologies using semantic technologies. A number of different standards and reasoning capabilities are available to be used depending on the requirement. A wide range of development tools and reasoning engines are also available. However, context retrieval can be computationally intensive and time consuming when the amount of data is increased.

### 2.4.3 CONTEXT REASONING

Context reasoning can be defined as a method of deducing new knowledge based on the available context [11]. It can also be explained as a process of giving high-level context deductions from a set of contexts. Reasoning is also called inferencing. Broadly, reasoning can be divided into three phases [32]:

- **Context pre-processing:** This phase cleans the collected sensor data. Due to inefficiencies in hardware sensor and network communication, collected data may be not accurate or can be missing. Therefore, data need to be cleaned by filling missing values, removing outliers, validating context via multiple sources, and many more.
- **Sensor data fusion:** It is a method of combining sensor data from multiple sensors to produce more accurate, more complete, and more dependable information that could not be achieved through a single sensor [23].
- **Context inference:** It is a method of generation of high-level (secondary) context information using lower-level (primary) context. The inferencing can be done in a single interaction or in multiple interactions. For example in a situation where the context is represented as tuples (e.g. Who: Leonardo, What: walking:1km/h, Where: Porto Alegre, When: 2016-01-05:11.30am). This low-level context can be inferred through a number of reasoning mechanisms to generate the final results. For example, in the first iteration, longitude and latitude values of a GPS sensor may be inferred as Rei do Cordeiro restaurant in Porto Alegre. In the next iteration Rei do Cordeiro restaurant in Porto Alegre may be inferred as Leonardo's favourite restaurant. Each iteration gives more accurate and meaningful information.

Context reasoning techniques are classified into six categories [35]: supervised learning, unsupervised learning, rules, fuzzy logic, ontological reasoning, and probabilistic reasoning.

1) *Supervised learning:* In this category of techniques, we first collect training examples. We label them according to the results we expect. Then we derive a function that can generate the expected results using the training data. Decision tree is a supervised learning technique where it builds a tree from a dataset that can be used to classify data.

2) *Unsupervised learning:* This category of techniques can find hidden structures in unlabelled data. Due to the use of no training data, there is no error or reward signal to evaluate a potential solution.

3) *Rules:* This is the simplest and most straightforward method of reasoning. Rules are usually structured in an IF-THEN-ELSE format. Rules are expected to play a significant role in IoT, where they are the easiest and simplest way to model human thinking and reasoning in machines [35]. A rule specifies that *when* a particular set of conditions occurs, specified in the Left Hand Side (LHS), *then* do what is specified as a list of actions in the Right Hand Side (RHS) [25]

(see Figure 2.2 ). The rules also supports Complex Event Processing (CEP), which can be broadly defined as being an event processing concept that deals with the task of processing multiple events (e.g., data generation, device update) with the goal of identifying the meaningful events within the event cloud [5].

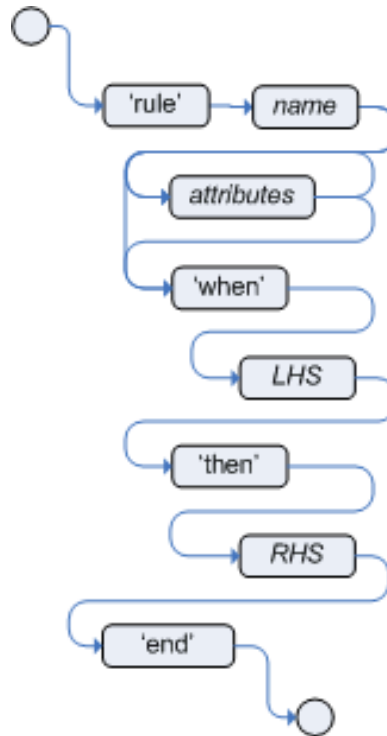


Figure 2.2 – Structure of a Rule.

4) *Fuzzy logic*: This allows approximate reasoning instead of fixed and crisp reasoning. Fuzzy logic is similar to probabilistic reasoning but confidence values represent degrees of membership rather than probability [38]. In traditional logic theory, acceptable truth values are 0 or 1. In fuzzy logic partial truth values are acceptable. It allows real world scenarios to be represented more naturally; as most real world facts are not crisp.

5) *Ontology based*: It is based on description logic, which is a family of logic-based knowledge representations of formalisms. The advantage of ontological reasoning is that it integrates well with ontology modelling. In contrast, a disadvantage is that ontological reasoning is not capable of finding missing values or ambiguous information where statistical reasoning techniques are good at that. Rules can be used to minimize this weakness by generating new context information based on low-level context.

6) *Probabilistic logic*: This category allows decisions to be made based on probabilities attached to the facts related to the problem. This technique is used to understand occurrence of events. For example, it provides a method to bridge the gap between raw GPS sensor measurements and high level information such as a user destination, mode of transportation, calendar-based observable evidence such as user calendar, weather, etc.

#### 2.4.4 CONTEXT DISTRIBUTION

Finally, context distribution is a fairly straightforward task. It provides methods to deliver context to the consumers. From the consumer perspective this task can be called context acquisition. There are two methods that are commonly used in context distribution [35]:

- Query: A context consumer makes a request in terms of a query, so the context management system can use that query to produce results.
- Subscription (also called publish/subscribe): A context consumer subscribes to a context management system by describing the requirements (i.e., what the consumer wants). The system will then return the results periodically or when an event occurs. In other terms, consumers can subscribe for a specific sensor or to an event.

### 3. RELATED WORK

In this chapter we make an analysis of two types of related work. First, section 3.1 presents a comparison among IoT middleware systems. Second, section 3.2 presents a comparison among systems that provide context-aware features. The systems of section 3.1 can appear at section 3.2 if it had the context-aware feature. Finally, Section 3.2.2 argue about CONASYS features comparing with others systems.

#### 3.1 IOT MIDDLEWARE SYSTEMS

This section presents and classifies some prominent IoT middleware systems. The idea is to discover the existing gaps in this area of research in order to verify if context-aware is a challenge in this role. Table 3.1 depicts the classifications of these systems according to five key requirements for IoT environments: device management, interoperation, platform portability, context-awareness, security and privacy [8].

Table 3.1 – IoT-middleware comparison.

IoT Middleware	Device Management	Interoperation	Platform Portability	Context Awareness	Security and Privacy
Hydra	✓	✓	✓	✓	✓
ISMB	✓	-	✓	-	-
ASPIRE	✓	-	✓	-	-
UbiWARE	✓	-	✓	✓	-
ubiSOAP	✓	✓	✓	-	-
UbiRoad	✓	✓	✓	✓	✓
GSN	✓	-	✓	-	✓
SMEPP	✓	-	✓	✓	✓
SOCRADES	✓	✓	✓	-	✓
SIRENA	✓	✓	✓	-	✓
WhereX	✓	✓	✓	-	-
COMPaaS	✓	✓	✓	-	✓

According to Table 3.1, we can notice that the majority of the analyzed IoT middleware solutions do not provide context-awareness functionality. In contrast, all the solutions are highly focused on device management, which basically involves connecting sensors to the IoT middleware. In the early days, context-awareness was a requirement strongly bounded to pervasive and ubiquitous computing. However, even though there were middleware solutions that provided context-aware functionality, today this scenario is different, and most of the existing middleware systems do not satisfy the context requirements demanded by IoT [35].

This lack of context-aware functionality in modern IoT systems opens a way for research and development in context-aware middleware for IoT. The context-aware feature for IoT systems

has a lot of items that can be analyzed. In the next section we evaluate some systems that help finding the existing gaps in the context-aware area.

### 3.2 CONTEXT-AWARE SYSTEMS

Table 3.2 presents a comparison between systems with context-aware features. The comparison covers the main features of context-aware systems according to [35] and [1]. The items used for the comparison are: (1) Modelling, (2) Reasoning, (3) Distribution, (4) History and Storage, (5) Knowledge Management, (6) Event Detection, (7) Level of Context Awareness, (8) Data Source Support, (9) Quality of Context, (10) Data Processing, (11) Dynamic Composition, (12) Real Time Processing and (13) Registry Maintenance and Lookup Services. We only analyzed systems that provide details of these items in the literature. The definition of each item is given after the table.

Table 3.2 – Context-aware systems comparison.

IoT Systems	Context-Aware Features												
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
Hydra	K,On,Ob	R,O	Q	✓	✓	✓	H	P	V	-	-	-	-
COSMOS	Ob	R	Q	-	-	✓	H	P	-	A	✓	-	✓
SALES	M	R	Q	-	-	✓	L	P	-	F	-	-	✓
C-Cast	M	R	P,Q	✓	-	✓	H	A	-	-	-	-	✓
CoMiHoc	Ob	R,P	Q	-	✓	✓	H	A	V	-	-	-	-
MidSen	K	R	P,Q	-	✓	✓	H	P	-	-	-	-	✓
CARISMA	M	R	Q	-	-	-	H	M	C	-	-	-	-
ezContext	K,Ob	R	Q	✓	✓	✓	H	A	-	A	-	-	✓
Feel@Home	G,On	O	P,Q	-	✓	✓	H	A	-	-	-	-	✓
UbiQuSE	M	R	Q	✓	-	✓	H	A	-	-	-	✓	-
CONASYS	K,M	R	P,Q	✓	✓	✓	H	A	V	F	-	✓	✓

#### 1) Modelling

It has been discussed in detail in Chapter 2, Section 2.4.2. The following abbreviations are used to denote the context modelling techniques employed by the system: key-value modelling (K), markup Schemes (M), graphical modelling (G), object oriented modelling (Ob), logic-based modelling (L), and ontology-based modelling (On).

#### 2) Reasoning

It has been discussed in detail in Chapter 2, Section 2.4.3. The following abbreviations are used to denote the reasoning techniques employed by the system: supervised learning (S), un-supervised learning (U), rules (R), fuzzy logic (F), ontology-based (O), and probabilistic reasoning (P). The symbol (✓) is used where reasoning functionality is provided but the specific technique is not mentioned.

#### 3) Distribution

It has been discussed in detail in Chapter 2, Section 2.4.4. The following abbreviations are



used to denote the distribution techniques employed by the system: publish/subscribe (P) and query (Q).

#### 4) History and Storage

Storing context history is critical in both traditional context-aware computing and IoT [18]. Historic data allows historic sensor data to be better understood. Specifically, it allows user behaviours, preferences, patterns, trends, needs, and many more to be understood by using the historic information in order to provide new context. The symbol (✓) is used to denote that context history functionality is facilitated and employed by the system.

#### 5) Knowledge Management

Most of the tasks that are performed by IoT middleware solutions require knowledge (context) in different perspectives, such as knowledge on sensors, domains, users, activities, and many more. Knowledge can be used for tasks such as automated configuration of sensors to IoT middleware, automatic sensor data annotation, reasoning, and event detection. The symbol (✓) is used to denote that knowledge management functionality is facilitated and employed by the system in some perspective.

#### 6) Event Detection

IoT envisions machine-to-machine (M2M) and machine-to-person communication. Most of these interactions are likely to occur based on an event. An occurrence of event is also called an event trigger. Once an event has been triggered, a notification or action may be executed. For example, detecting current activity of a person or detecting a meeting status in a room, can be considered as events. Mostly, event detection needs to be done in real-time. However, some events such as trends may be detected using historical data, detecting patterns. The symbol (✓) is used to denote that event detection functionality is facilitated and employed by the system in some perspective.

#### 7) Level of Context Awareness

Context-awareness can be employed at two levels: low (hardware) level and high (software) level. At the hardware level, context-awareness is used to facilitate tasks such as efficient routing, modelling, reasoning, storage and event detection [24]. The software level has access to a broader range of data and knowledge as well as more resources, which enables more complex reasoning to be performed. The following abbreviations are used to denote the level of context awareness facilitated and employed by the system: high level (H) and low level (L).

#### 8) Data Source Support

There are different sources that are capable of providing context. (P) denotes that the solution supports only physical sensors. Software sensors (S) denotes that the solution supports either virtual sensors, logical sensors or both. (A) denotes that the solution supports all kinds of data sources (i.e. physical, virtual, and logical). (M) denotes that the solution supports mobile sensors.

### 9) Quality of Context

It denotes the presence of conflict resolution functionality (C) and context validation functionality (V). Conflict resolution is critical in the context management domain [20]. Context validation ensures that collected data is correct and meaningful. Possible validations are checks for range, limit, logic, data type, cross-system consistency, uniqueness, cardinality, consistency, data source quality, security, and privacy.

### 10) Data Processing

Denotes the presence of context aggregation functionality (A) and context filter functionality (F). Context filter functionality makes sure the reasoning engine processes only important data. Filtering functionality can be presented in different solutions in different forms: filter data, filter context sources, or filter events. One of the simplest forms of aggregation of context is just collect similar information together.

### 11) Dynamic Composition

IoT solutions must have a programming model that allows dynamic composition without requiring the developer or user to identify specific sensors and devices. Software solutions should be able to understand the requirements and demands on each situation, then organize and structure its internal components according to them. The symbol (✓) denotes the presence of dynamic composition functionality in the system in some form.

### 12) Real Time Processing

Most of the interactions are expected to be processed in real time in IoT. This functionality has been rarely addressed by the research community in the context-aware computing domain, even it being an of the most important context-aware features for IoT. The symbol (✓) denotes the presence of real time processing functionality in some form.

### 13) Registry Maintenance and Lookup Services

The (✓) symbol is used to denote the presence of registry maintenance and lookup services functionality in the systems. This functionality allows different components such as context sources, data fusion operators, knowledge bases, and context consumers to be registered.

## 3.2.1 SYSTEMS APPROACHES

Some systems provide context-aware functions to IoT environments. This subsection presents a review about context-aware features of the systems presented in Table 3.2. We also show how these systems generate the context information. The last row of the Table 3.2 presents CONASYS, the system developed by this work.

Hydra [5] is a system that comprises a Context Aware Framework that is responsible for connecting and retrieving data from sensors, context management and context interpretation. A rule engine called Drools [25] has been employed as the core context reasoning mechanism. COSMOS [15]

is a system that enables the processing of context information in ubiquitous environments. COSMOS consists of three layers: context collector (collects information), context processing (derives high level information), and context adaptation (provides context access to applications). COSMOS follows distributed architecture which increases the scalability of the system.

SALES [16] is a context-aware system that achieves scalability in context dissemination. XML schemes are used to store and transfer context. C-Cast [37] is a system that integrates WSN (Wireless sensor networks) into context-aware systems by addressing context acquisition, dissemination, representation, recognizing, and reasoning about context and situations. The data history can be used for context prediction based on expired context information.

CoMiHoc [44] is a framework that supports context management and situation reasoning. CoMiHoc architecture comprises six components: context provisioner, request manager, situation reasoner, location reasoner, communication manager, and On-Demand Multicast Routing Protocol (ODMRP). MidSen [34], as C-Cast, is a context-aware system for WSN. MidSen is based on Event-Condition-Action (ECA) rules. The system has proposed a complete architecture to enable context awareness in WSN.

CARISMA [12] is focused on mobile systems where they are extremely dynamic. Adaptation is the main focus of CARISMA. Context is stored as application profiles (XML based), which allows each application to maintain meta-data. The framework ezContext [29] provides automatic context life cycle management. The ezContext comprises several components that provides context, retrieves context, does modelling and storage context.

Feel@Home [22] is a context management framework that supports interaction between different domains. Feel@Home decides what the relevant domain needs to be contacted to answer the user query. Then, the framework redirects the user query to the relevant domain context managers. Feel@Home consists of context management components responsible for context reasoning and store context. UbiQuSE [40] (Ubiquitous Queries for Sensing Environments) is a framework to manage streaming, contextual and data mining queries, providing a query interface. UbiQuSE shows how real time query processing can be done incorporating live streaming data and historic context in repositories.

The first feature to be analyzed in Table 3.2 is related to systems context modeling. The modeling approaches that more appeared in the comparison were markup schemes, key-value and object-oriented modeling. Modeling through key-value is made by Hydra for simplicity of use [5]. CARISMA uses markup schemes, because the way it models the context can be easily understood, both by machines and by human [12].

In reasoning and distribution (2 and 3 respectively) almost all analyzed systems seem to have a consensus regarding which technologies to use. With respect to reasoning, the most of analyzed systems use rules as a tool. A study by [35] showed that rules is the most popular method of reasoning used by systems. Hydra besides rules also uses ontologies as a promising technology [46]. On the other hand, Feel@Home makes use only of ontologies. To supply context distribution all

analyzed systems use query. However, some systems as C-Cast, MidSen and Feel@Home also offer the possibility of using publish/subscribe as an additional feature.

As shown in Table 3.2, the function of history and storage (4) is a differential of the analyzed systems, only a few systems have this feature. For the C-Cast, the history can be used for context prediction based on expired context information [37]. Another differential feature is knowledge management (5). One of the few that provide this functionality is CoMiHoc. In this system, knowledge is required to overcome the limitations of the environment and to provide reliable support for the applications [44]. Detection of events (6) is a feature provided by almost all systems. When specific context events occurs, event detection takes action such as shutting down if the battery is low [5].

In terms of level of context awareness (7), only one system has a low level, which works with the context in hardware. All other analyzed systems work with context in terms of software, which allows a greater capacity for reasoning [35]. Regarding data source support (8), most analyzed systems support physical sensors. CARISMA supports mobile sensors because it is a specific solution for this area [12]. A better alternative is to support the largest possible range of different sensors, since IoT provides heterogeneous environment [4].

A comparison was made between systems on quality of context (9). A low number of analyzed systems control quality of context. In CoMiHoC the quality of context is addressed by validation that is integrated into the communication protocol [44]. Data processing (10) is another analyzed functionality. Only a few systems perform some kind of processing. SALES uses filtering techniques to reduce traffic [16].

Another feature compared between systems was dynamic composition (11). This is only attended by COSMOS [15]. The real time processing (12) becomes a challenge of future context-aware systems, as only one of the analyzed systems had this feature. Finally, the last item used for systems analysis was registry maintenance and lookup services (13). Many of compared systems have this feature. Through it, the systems can have a history of performed processes, thus facilitating future operations [35].

### 3.2.2 OUR SYSTEM APPROACH

Context-awareness is an important feature of IoT middleware and the process to provide context-aware services encompasses some procedures including: context acquisition, context modeling, context reasoning, and context distribution. For all these phases, the selection of techniques that fulfill the system's necessities is essential.

The technologies for contextualized information services provision were presented in Chapter 2. These technologies are essential for improve the Quality of Experience (QoE) in the services provision. QoE focuses on how good is the interaction of the user with the system. We demonstrate that in CONASYS, for context acquisition, PUSH and PULL methods were used in order

to meet the heterogeneous devices types. In the context modeling, both Key-Value and Markup Scheme techniques were used. These techniques are simple to use and facilitate the organization and storage of the context.

For context reasoning, rules supply the necessities of the system, as it has a good cost-benefit compared to other technologies. Rules are simple to define, have a well-defined structure, and consume few resources (e.g., processing, storage). Finally, to give different options to the user, we intended to provide context distribution by two ways: query and subscription. In this sense, a user can make a simple request or subscribe one or more services by the time that he wants.

Custom context-aware approaches offered by the IoT research community mostly not offer details of its architecture. In Chapter 3, we shown that the context-aware systems have well-defined functions to obtain and provide contextualized information. However, they may differ in terms of architecture and technologies according to the need of the systems. Moreover, just one of the studied solutions address real-time processing of the context, which is considered a gap in the area [35].

The real-time feature of CONASYS appears as a good characteristic. This feature makes the user request be answered in less steps compared with the normal flow (see Figure 4.13). This shortcut reduces the processing effort of the cycle and eliminates the need of create a middleware connection, what could add a communication delay caused by the network.

Comparing with other works, through Table 3.2, we can see that CONASYS is a good alternative to be implemented in IoT environments. CONASYS addresses almost all the context-aware requirements. The real-time requirement, that is a key feature of context-aware systems [35], is only addressed by one of the analyzed systems. Although this system addresses real-time, it fails in address other basic context-aware IoT requirements, like the registry maintenance that allows components such as context sources and knowledge bases to be registered. CONASYS can stand out in IoT context-aware scenario as being a complete system.

A context-aware system that provides real-time information services and having a well-defined structure that is able to handle with different context situations is not yet defined and is necessary in IoT environments. IoT mobile ecosystems are in constant change, so the feature of providing contextualized information services in a real-time way becomes an important characteristic of our system. Besides, as the IoT can be applied to different situations, the possibility of working with multiples domains is also a strong characteristic of CONASYS. The CONASYS has these features in order to cover the IoT context-awareness gaps, that are the less addressed context-aware features of all the systems. By having this features, CONASYS stands out comparing with other systems. The system structure will be presented in more detail in Chapter 4.



## 4. PROPOSED APPROACH

In this chapter we present the definitions and characteristics of our context-aware system for IoT environments, CONASYS. Section 4.1 presents our prior work COMPaaS, which was the reference platform used as basis for our context-aware system. Section 4.2 presents all the characteristics of CONASYS. Moreover, we present an example of use in addition to some definitions of the system services provision and a technical overview of the system.

### 4.1 REFERENCE PLATFORM

COMPaaS (Cooperative Middleware Platform as a Service) is an IoT middleware developed by the GSE/PUCRS [3]. COMPaaS is a software system that provides to users a simple and well-defined infrastructure of middleware services to be used in IoT environments. Behind the services provided by the middleware, there is a set of system layers that deal with the users and applications requirements, for example, request and notification of data, discovery and management of physical devices, communication issues, and data management. COMPaaS is the reference platform for this work and will be extended to support the proposed context-aware services provision system. We chose COMPaaS for the easily access to the source code, architecture that supports the services provision, and because COMPaaS did not has context-aware features.

The goals of COMPaaS can be summarized as follow:

- Abstract the integration and interoperability with physical devices (physical resources) through the provision of hierarchical services according to the profile of the device.
- Abstract the collection and management of the data provided by physical devices through the provision of application level services.
- Provide high-level services to facilitate the development and integration of IoT applications.
- Provide well-defined software architecture based on IoT/M2M and WoT (Web of Things) standards.

COMPaaS is based on a Service-Oriented Architecture (SOA). It is composed of three main systems: Middleware API, Middleware Core, and Logical Device. Middleware API is the system that has the methods to be used by applications that want to use COMPaaS services. Middleware Core is the system responsible for abstracting the interactions between applications and devices and also for hide all the complexity involved in these activities. Logical Device is the system responsible for hiding all the complexity of physical devices and abstracts the functionalities of these devices to the upper layer. Figure 4.1 presents an overview of the middleware platform and highlights some technical details around the integration between application, middleware and logical devices, as well as the

main flow of information. The idea is to present the communication interfaces, communication patterns, and information that are exchanged between the systems.

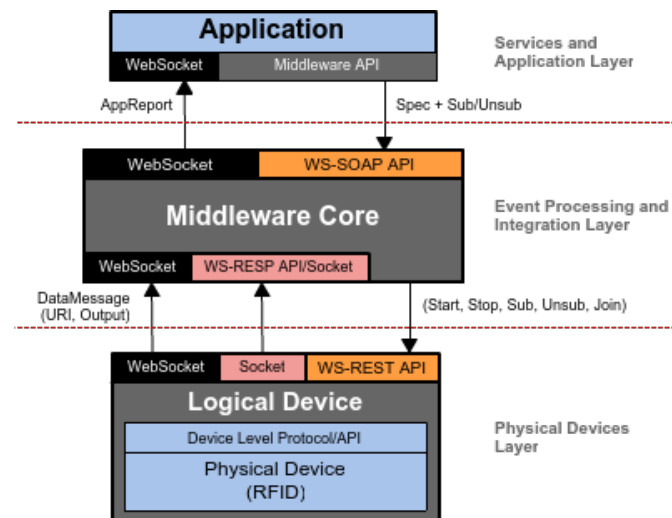


Figure 4.1 – COMPaaS architecture overview.

To help the description of the systems we use a simple use case regarding “a generic application requesting data from an RFID reader”. First we describe the main characteristics of the environment and some technical details of the systems. Further, we present the basic interactions and the data exchanged by items.

- The operational environment is composed of the Application, Middleware Core, and Logical Device, that encapsulates and controls an RFID reader.
- Each Logical Device provides its own services to the Middleware Core through a standard RESTful API (HTTP protocol for subscription). These services abstract the main methods of each device and allow Middleware Core to send commands like start, stop, subscribe, unsubscribe and join. Each Logical Device must be implemented in order to define its “profile” and to “encapsulate the low-level API of the device (device driver)”. Thus, the native API of the device can be mapped into the RESTful API.
- Logical Device use a WebSocket channel (a data notification service provide by the Middleware Core) for asynchronous responses (TCP-based protocol for notification of data). The WebSocket is used not only to avoid the lack of performance of the HTTP (request-response style), but also to allow that physical devices can notify the Middleware Core without a synchronous request.
- The “DataMessage object” is used as the default data packet between Logical Device and Middleware Core. DataMessage is an object that represents the data generated by the Logical Device. It is a generic object and must be created in the design time according to the profile of the device. It demands a proper and pre-defined CEP-based rule in the Middleware Core to be processed.



- Middleware Core provides its on services to the application through Web Service SOAP. The SOAP is used as the communication pattern between applications and Middleware Core because SOAP is a standard and fits well in scenarios with “medium-performance” in terms of real-time communication requirements. Furthermore, SOAP is “neutral” in terms of transport protocol. It runs over any transport protocol (HTTP, TCP, UDP) and also allows independence of any client-programming model (loosely-coupled distributed communication pattern) what is an important requirement for the interoperability required by the middleware project.
- WebSocket is used for asynchronous responses from Middleware Core to applications (TCP-based protocol for notification of data). The WebSocket is used to allow that the Middleware Core can notify the application without a synchronous request.
- Both systems architectures (Middleware Core and Logical Device) are based on “Subscribe/Notify” and “Observer” communication pattern. “RESTful + WebSocket” for the Logical Device, and “SOAP + WebSocket” for the Middleware Core.

Although COMPaaS has many features in its architecture, it is not able to work according to the context in which it is inserted. An middleware must be context-aware in order to work with IoT environments, which is considered a challenge for these systems [6]. In this way, we intend to address this important challenge providing context-aware features in this system.

## 4.2 CONASYS: CONTEXT-AWARE SYSTEM

The Context-Aware System (CONASYS) aims to provide to user/application a set of services of contextualized information by a well-defined structure of features that understands the environment in which the system is inserted and provides services based on this environment, both on-line (i.e., through newest contextualized data) and off-line (i.e., through historical contextualized data). These services are called information services because they provide somehow with data/knowledge/information. The information services must be used independent of the knowledge of the environment. In other words, a user can request the information services without knowing exactly which things or devices will be used in the process to collect/provide the information.

CONASYS interacts with the infrastructure provided by COMPaaS middleware, including the devices connected to it, in order to have access to the IoT environment infrastructure (e.g., devices). Moreover, several instances of COMPaaS middleware may be connected to CONASYS and each one is responsible for dealing with a specific domain (e.g., smart home, smart office, healthcare, and mobile). Each domain should have a specific set of business rules that must be registered in the system.

In this section we present the architectural overview of the CONASYS with details of the structure that compose the system. Moreover, we present in more detail the process of services

provision, showing how the system composes the information services that is displayed to the final user. We also present the technical overview of the system, focusing in how the system works. Finally, we present an example of system usability in a real use case scenario.

#### 4.2.1 ARCHITECTURAL OVERVIEW

An API was developed to allow users (developers) to interact with the system. In this sense, the users must send an XML file containing information regarding their requests. In addition to the API, CONASYS provides a system architecture composed of three main layers (see Figure 4.2): Communication Layer, Storage Layer and Processing Layer. Each layer has specific goals that are presented next:

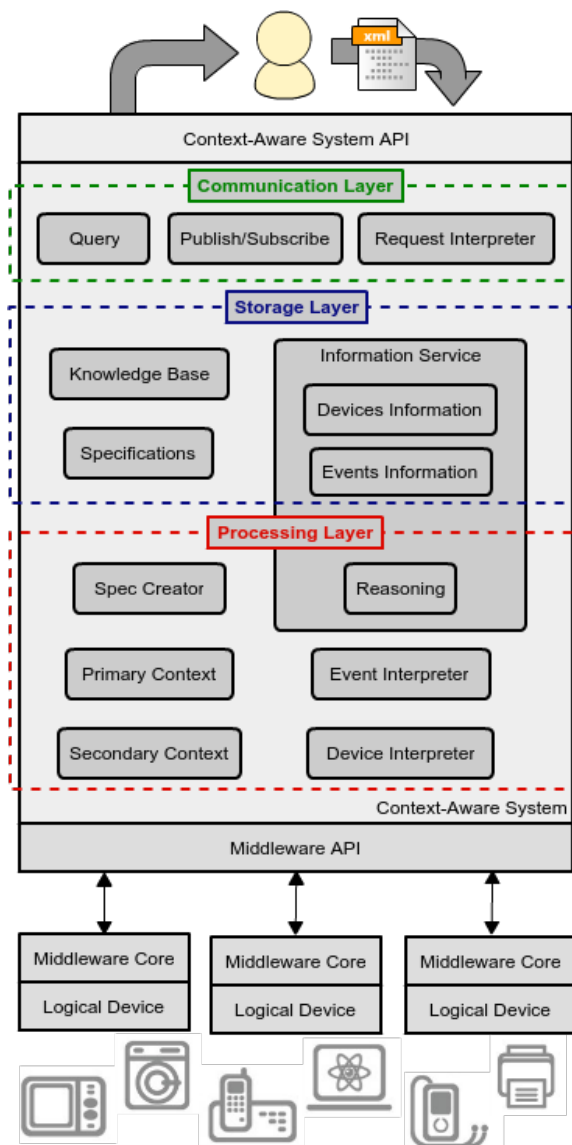


Figure 4.2 – CONASYS layers and modules overview.

- **Communication Layer:** The process of receiving and interpreting the user's request is made by the Communication Layer. This layer is also responsible for the context distribution process related to context life cycle (send results to users). The Communication layer is composed of three modules: Query, Publish/Subscribe, and Request Interpreter.

The Request Interpreter is responsible for understanding the user's request and also for starting the response process. For example, if a user wants to subscribe a service, the Request Interpreter module communicates with the Publish/Subscribe module informing the user contact information. On the other hand, if a user wants to query some information without subscription, the Query module is communicated with the user information. Both Query and Publish/Subscribe modules are responsible for maintaining user contact information and also for sending the request result.

- **Storage Layer:** This layer is responsible for storing the content of all modules presented in CONASYS. The Storage layer is composed of four modules: Knowledge Base, Specifications, Devices Information, and Events Information. In addition, the Information Service module belongs to both Storage and Processing Layers. It is shared with the Processing Layer because it has context reasoning functions.

The context modeling phase of context life-cycle can be identified in the Storage Layer. The Knowledge Base module is responsible for storing the context information. Thus, this module has great importance in the system off-line mode, which uses historical data. In some cases, if the desired information is properly updated, the interaction with the middleware is not necessary. In addition, it is also responsible for interpreting the Drools rules [25] of the system, as well as for storing the key parts of each rule. The Specifications module of Storage layer is responsible for storing the information that allows CONASYS to interact with the middleware. The characteristics of each device connected to the middleware are stored in Devices Information module. Finally, the Events Information module stores contents of each event (e.g., change of state, data generation) that happens related to any device connected to the middleware.

- **Processing Layer:** This layer is very important in the context generation since it is responsible for the context reasoning phase of the context life-cycle. The context life-cycle was explained in more detail at Chapter 2. The processing layer has six specific modules: Spec Creator, Primary Context, Secondary Context, Reasoning, Event Interpreter, and Device Interpreter.

The main component of the Processing layer is the Reasoning module, that contains the Drools rules [25]. These rules are responsible for the context reasoning process and can be from different domains. The Spec Creator module is responsible for creating specifications that are needed to allow the communication with the middleware. The Primary and Secondary Context modules receive the information about the devices that should be used to access data through the middleware from Request Interpreter module. The difference between them is that the Secondary Context module also have fusion methods. These fusion methods can be

related to two or more device data. For example, a fusion method can receive the information that two devices data are needed to answer a user request and transform it in a single data. The Event Interpreter module is responsible for receiving events from the middleware. The Device Interpreter module has functions that collect information from devices connected to the middleware. The Reasoning module is responsible for giving meaning (i.e., context reasoning process) to the information collected by Event and Device Interpreter modules.

#### 4.2.2 SYSTEM USABILITY

With the dissemination of the IoT, an increasing number of devices will be connected to the Internet in the coming years. These devices will be of a variety of types, as sensors, tags, and smartphones. Bearing in mind this scenario with heterogeneous devices, we can create a case study. In this case study, CONASYS has a specific service. This service is responsible for notifying a user of the buses that stop at the user's nearest station in an smart city environment. In addition, the service is responsible for showing the distance between the bus and the station (Figure 4.3). In this case, the user's smartphone acts as a source of location data (GPS) as well a consumer of the service provided by CONASYS. The actions are taking automatically by the CONASYS according to the occurrence of events. Moreover, location sensors can be identified in the buses and in the station for context acquisition.

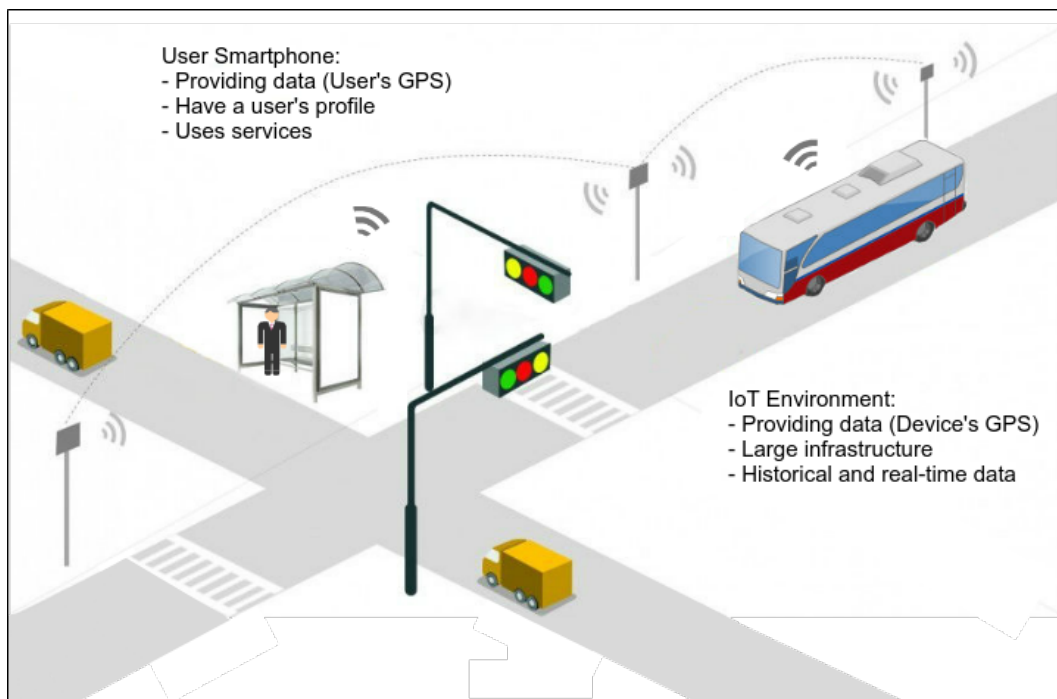


Figure 4.3 – Case study example.

In the first step, the user must consult the list of information services provided by CONASYS (see Figure 4.4). This list shows the services that the system provides and the settings to use them. The list is directly linked to the rules that are registered in the system. The services that appear in

this list are strictly linked to one or more rules. If one of the devices that compose a service had the status offline (OFF), the service can not be used. A service only can be used if all the devices of the rule that compose a service has the status online (ON). Before that, the user selects the service responsible for informing about the buses. If desired, the user can select more than one service. The second option that must be defined is which type of request has to be performed: Query or Subscription. If the Subscription option is chosen, then it must be specified certain parameters related to the subscription duration time, which are: Duration and Repeat Period. The Duration parameter is a numeric type and indicates the amount of time that the service will be subscribed. The Repeat Period parameter is also of a numeric type and indicates how long it takes to receive new data for the chosen service. However, depending on the type of service, the user can be notified when changes related to the service occurs. The Query option does not have parameters.

ID	Name	Purpose	Devices	Status	Frequency	Type of Return
#07	City_Pollution	- Monitors the overall pollution of Porto Alegre, Brazil. - Considers air pollution, ground pollution and water pollution.	- Air_152 - Ground_014 - Water_S01	- ON - ON - ON	- 4500 ms	- Output Object
#15	Monitor_Bus_L01	- Monitors the location of the bus line 01. - Returns information when the touristic bus go through certain points of the city.	- GPS_B01 - Location_77 - Location_21 - Location_30	- ON - ON - ON - ON	- 2000 ms	- Output Object
#23	Fire_Caution_B32	- Monitors the overall temperature of the PUCRS building 32. - Triggers alerts if it detects fire.	- Temp_B32 - Smoke_B32	- OFF - ON	- 7000 ms	- Output Object

Figure 4.4 – Interface that shows CONASYS available services.

From this moment, the user stops the interaction with the system and only waits the request result. CONASYS receives this request and interprets it. After that, the system understands the user needs and communicates with the Information Service in order to find out if the information that user requested is stored and updated. However, if the information is not stored or updated, the information capturing process is started, and the system communicates with the middleware in order to collect the data in real-time. The Knowledge module contains the necessary information to know what devices are needed. After the CONASYS collects data from middleware, the system interprets these data and does the real-time processing in order to give context to the event. Thus, through the Drools rules, it is possible to know if the event interests the user, and the notification to the user is done. Moreover, if the data requested by the user are part of the knowledge of CONASYS but the system has no rules to treat them, the system has the ability to adapt to this request by creating real-time rules through a preset template.

Figure 4.5 shows the flow of these activities by CONASYS view without details of the interaction with the middleware. It is possible also to identify the two different ways of using the

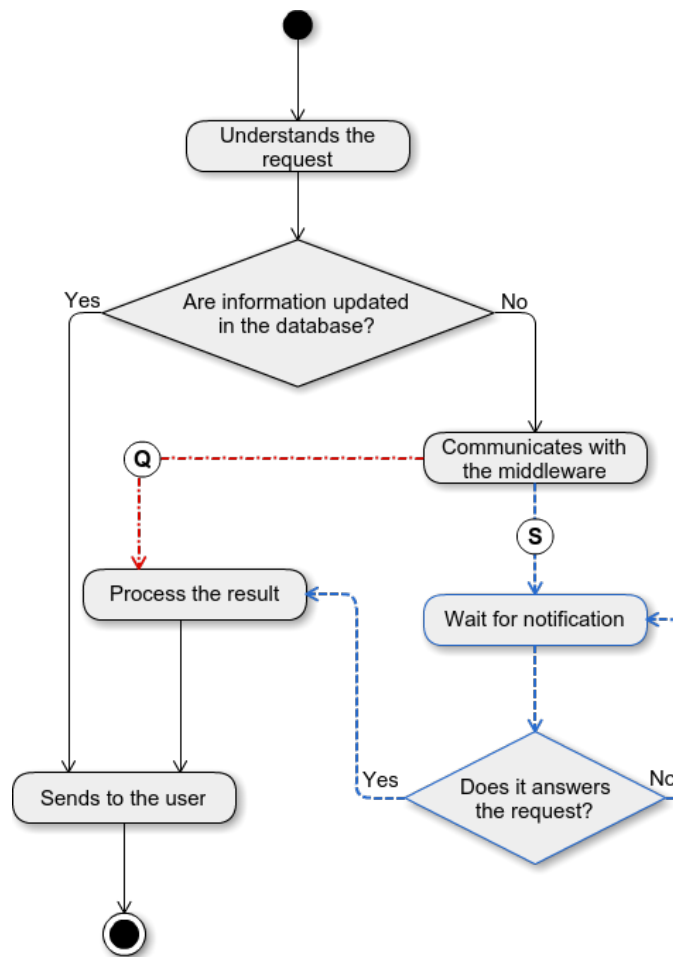


Figure 4.5 – CONASYS activity flow.

system. These two ways are shown with letters “Q” and “S”. The letter “Q” is when the user sends a request and receives the answer immediately (i.e., query). The letter “S” indicates the flow taken when it is desired to subscribe any available service and receive updates (i.e., subscription).

#### 4.2.3 SERVICES PROVISION

There are two different ways for services provision in CONASYS. The first is when a user sends a request to the system and receives an answer immediately. The second one includes a subscription. In this case the user sends requests to the system informing which services he wants to subscribe. The system monitors these services and notifies the subscriber when the services status change or suffer an update. For the provision of both types of services is necessary to take some steps to have contextualized data. These steps are shown in Figure 4.6 and detailed next.

The first step for assigning context is to extract it from some source. As long as the data is generated from any device, a middleware collects the data in different ways. In most of IoT middleware, the acquisition is made through PUSH and PULL methods. PUSH is used when sensors know when to send the data and have enough processing power to do that. And PULL is used

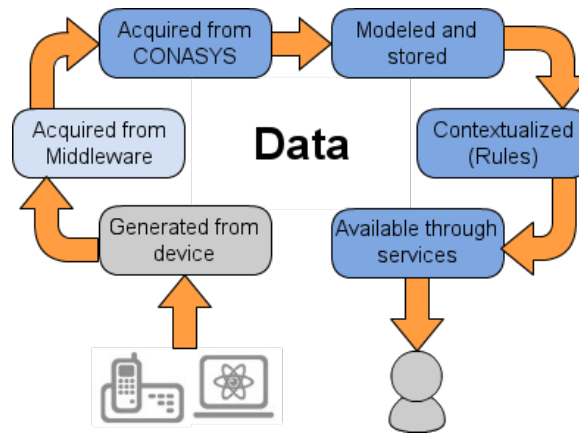


Figure 4.6 – The data life-cycle to provide an information service.

when sensors do not have knowledge about when to send the data to the consumer. In CONASYS the acquisition process is made through a middleware interface. The CONASYS starts a *thread* to capture new information (i.e., data) about devices and events related to the environment that it is inserted. In this way, every new event is captured by the system.

The modeling of the acquired context is an important step in CONASYS. The system uses two different types of modeling: Key-Value and Markup Scheme modeling. With the use of Markup Scheme, context can be stored using tags. These tags are related to the main characteristics of the events. With the Key-Value it is possible to use a key in order to make easy the future search for data. In CONASYS, the key of every event is the URI (Uniform Resource Identifier) of the device, because each device has his own and unique URI. For each query on a specific data, the device URI need to be informed. The implementation of these techniques takes place after the acquisition process. The modeled data are stored in a database of the Storage Layer (see Figure 4.2) and be available for the reasoning process.

It is in the context reasoning step that the modeling context is used to generate knowledge. CONASYS uses rules to make this process. After the data was modeled and stored it is possible to reason over these data. The reasoning process is responsible for defining the available services to the users based on the set of rules. The data is contextualized through Drools [25] rules which give a meaning to the event. These rules are in a Working Memory and each event that happens are added to these memory. In this way, if the event meets determinate condition of one or more rules, a sequence of actions is triggered to give meaning to the event. These sequence of actions can be a simple label to the event or a more complex action like run a method.

Context Distribution is responsible for the provision of the information services based on context [35]. It is a simple process. We use two main ways to have context distribution: query and subscription. The provision of information services can be seen as the context distribution process. In this process the device data is made available to the user in a contextualized way. The services provision is made through an XML file with tags that contains the result of the user's request.

#### 4.2.4 TECHNICAL OVERVIEW

There are some background functions in the CONASYS that occur before users send their requests. The first step is recognize the environment that CONASYS is inserted. CONASYS is strictly connected to an IoT middleware (e.g., the COMPaaS middleware). With a *thread* that receives all the updates that occur in the middleware, there will be possible to understand the environment (i.e., devices). These *threads* monitor the communication channel of the middleware and every time that a device connects/disconnects or update the information with the middleware, it triggers a function to acquire the content and type (e.g., connection, update) of these communications. Figure 4.7 shows an example of a device connection XML that the device sends to the middleware. CONASYS gets this XML file, understands it, and stores the information in a database (i.e., Storage Layer - Devices Information). The information is stored respecting all the individual characteristics of the device (XML tags). The most important information acquired from the XML is the individual URI, that is the device identity.

```

<profileInformation xsi:type="deviceProfile"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <uri>
    http://192.168.0.10:8080/Air_152/resources
  </uri>
  <name>Air_152</name>
  <location>
    <name>Stem 4 - level 1 - pendulum 2</name>
    <latitude>-27.817244 S</latitude>
    <longitude>-51.697457 W</longitude>
  </location>
  <purpose>Air pollution monitor</purpose>
  <owner>GSE</owner>
  <keywords>
    <keyword>CO2</keyword>
    <keyword>Carbon dioxide</keyword>
  </keywords>
  <services>
    <service>
      <responseType>Double</responseType>
      <generationTime>1000</generationTime>
      <availableService>
        CO2 meter
      </availableService>
    </service>
  </services>
  <model>
    <deviceClass>Sensor</deviceClass>
    <number>1903</number>
    <name>Digital Sensor</name>
  </model>
  <information>
    <status>ONLINE</status>
  </information>
</profileInformation>

```

Figure 4.7 – Example of an XML file with information about a device.

Another important background step from CONASYS is the rules interpretation. The rules are pre-inserted in the system through a *.drl* file. The developer that will use the CONASYS defines the set of rules and inserts it at the system in project time. These files can vary by domain. The interpretation is important in order to keep the description of all the rules of the system, thus providing for the user the services available in the system. CONASYS interprets the rules through the Knowledge Base module (see Figure 4.2). This module has directives to decompose the rule in parts and to store it in a database. In this sense, the system can consult all the rule information



in the future. An example of a Drools rules interpretable by CONASYS can be seen in Figure 4.8. The structure of a rule can be seen in the Chapter 2, Section 2.4.3.

```

1 rule "City_Pollution"
2   when
3     obj1 : Output( service_device == "Air_152")
4     obj2 : Output( service_device == "Ground_014")
5     obj3 : Output( service_device == "Water_S01")
6   then
7     String val_str1 = (String) obj1.getValue();
8     boolean val_bool2 = (boolean) obj2.getValue();
9     String val_str3 = (String) obj3.getValue();
10
11    double value = Double.parseDouble(val_str1);
12    if(value <=100 && value>=0) {
13      if(value>55){
14        obj1.setPriority(Priority.HIGHEST);
15      } else {
16        obj1.setPriority(Priority.NORMAL);
17      }
18    } else {
19      obj1.setPriority(Priority.HIGHEST);
20    }
21    double value = Double.parseDouble(val_str3);
22    if(value >=5.75) {
23      obj3.setPriority(Priority.HIGHEST);
24    } else {
25      obj3.setPriority(Priority.NORMAL);
26    }
27    if (val_bool2 == true && obj1.getPriority() == Priority.HIGHEST) {
28      if (obj3.getPriority() == Priority.HIGHEST) {
29        obj1.setPriority(Priority.ALERT);
30        obj2.setPriority(Priority.ALERT);
31        obj3.setPriority(Priority.ALERT);
32      }
33    }
34  end

```

Figure 4.8 – Example of a Drools rule.

All the background information needs to be stored in a database. Figure 4.9 shows an overview of CONASYS database modelling. In CONASYS we use two different types of modeling: Key-Value and Markup Scheme modeling. Key-Value pairs are easy to manage, but lacks modeling resources [41]. With the use of Markup Scheme, context can be stored using tags. The use of XML makes Markup Scheme easily used by applications and middleware since the middleware and CONASYS communications are related to SOAP web services. To every device be connected to the middleware it must send a XML file, and every time that a event occurs it generate a XML file. In this way, the information is extracted from XML tags of devices and events. The information from the rules are extracted with key-value pairs since the rules are easy to manage and every rule has the same basic structure.

As can be seen in Figure 4.9, the rule general information (e.g., name, functions) was summarized in the Characteristics field. All the rules have Conditions and Actions. The Conditions are related to what devices the rule is linked, and the Actions is what the rule do if its Conditions are matched. A Device has an URI that identifies it. As well with the rules, the information about devices (e.g., name, owner, model, purpose, keywords, class) were summarized in the Characteristics field. Moreover, the Device also contains a Device Location, that has the information regarding device localization, as well as GPS coordinates or the name of a specific location. In addition, the Device

Location also has an URI. Finally, the Event is responsible for storing the information related to when a data is generated from the middleware. An Event has a value, that can be any type of data (e.g., String, double, integer, boolean, etc.). The Timestamp of an Event is related to when the Event occurs in order to know in future queries if the Event is updated or not. Each Event contains an Device that is identified by the URI.

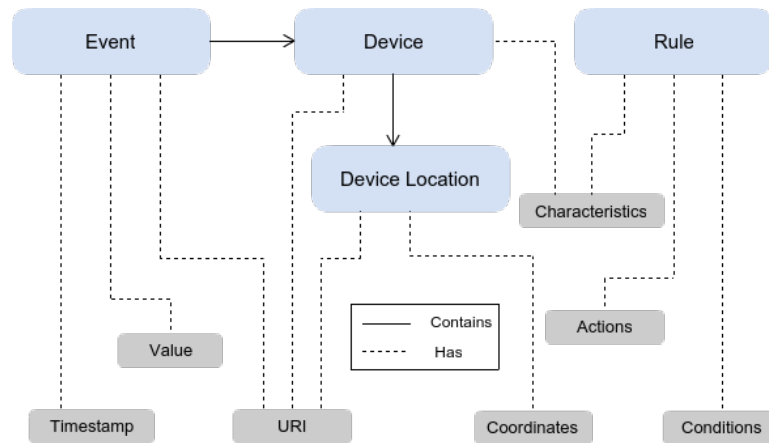


Figure 4.9 – Overview of CONASYS database modelling.

For the users (i.e., developers) to communicate with CONASYS they need to use the CONASYS API. The communication between “user - API” and “API - CONASYS” is made through SOAP web service. In this sense, users must send an XML file containing information regarding their requests. As seen in Section 4.2.2, CONASYS implements the “Observer” pattern, so besides the *query*, the user can also use *subscription*.

Users need only to import the CONASYS API in their project and use the API functions. The CONASYS API usage by *subscription* can be seen in the Figure 4.10. The first thing in order to use CONASYS is create a *ConasysConnection* and use the method *createConnection* with the communications parameters regarding user information (i.e., ip address and port). After that, the user must insert himself as an observer. An *ArrayList* with the name of the services that user wants to use must be created. The available services are provided by CONASYS as a menu (see Figure 4.4).

After the connection, the next step is to create the request. For this task the user must create a *ConasysRequest* and use it with the *createRequest* method. This process can be seen in the line 9 of the Figure 4.10. The first parameter is the name of the request, user can create any name. The second parameter is the *ArrayList* with the services that user wants to use. The third parameter is the type of consult, the options are *subscription* and *query*. The fourth parameter is the *duration*, that is of the numeric type and indicates the amount of time that the service will be subscribed. Finally, the fifth parameter is the *repeatPeriod* that is also of numeric type and indicates how long it takes to receive new data for the chosen service.

The next step after creating a request is send it to CONASYS. This is made by *ConasysRequest* method named *sendRequest*. In this method, the user must enter a request name of an existing request as first parameter. In Figure 4.10 the user creates a request in the line 9 and sends

```

1 ConasysConnection con = new ConasysConnection();
2 ConasysRequest conreq = new ConasysRequest();
3 ArrayList<String> services = new ArrayList<String>();
4
5 con.createConnection("192.168.0.1", 8380);
6 con.addObserver(this);
7 services.add("City_Pollution");
8
9 conreq.createRequest("Request01", services, TypeOfConsult.SUBSCRIPTION, 30000, 5000);
10 conreq.sendRequest("Request01", con, 35000);

```

Figure 4.10 – CONASYS access through subscription.

it in the line 10. The second parameter is the instance of *ConasysConnection*. Finally, the third parameter is the time for total subscription, it must be higher than *duration* present in the creation of the request. The total subscription time will be used to CONASYS API control the request internally. After this step the user will wait until the request results that will be received in the *update* method implemented by “Observer” pattern.

Figure 4.11 shows an example of a CONASYS usage by *query*. The main difference between *subscription* and *query* is that in the *query* method the user only receives one result and in the *subscription* user still receives results until the *duration* time. Some functions may vary between *query* and *subscription*. The big difference is that in the *query* example there was no needed of setting any time parameter (see Figure 4.11).

```

1 ConasysConnection con = new ConasysConnection();
2 ConasysRequest conreq = new ConasysRequest();
3 ArrayList<String> services = new ArrayList<String>();
4
5 con.createConnection("192.168.0.1", 8380);
6 con.addObserver(this);
7 services.add("City_Pollution");
8
9 conreq.createRequest("Request01", services, TypeOfConsult.QUERY);
10 conreq.sendRequest("Request01", con);

```

Figure 4.11 – CONASYS access through query.

The CONASYS API understands the user request and creates an XML file with the request regarding CONASYS patterns. CONASYS receives this XML file through SOAP web service. This file contains the request details specified by the user. Figure 4.12 shows an example of XML file. In this file is specified the service that the user wants to consult. In Figure 4.12 the service is exemplified as “City\_Pollution”, but more than one service may be consulted at one request. The second option that appears in the XML file is the *typeOfConsult*. If the *typeOfConsult* is a *Subscription*, the time parameters will appear as shown in the Figure 4.12.

After CONASYS receives the XML file and understands it the Processing Cycle (PC) is started. The PC is responsible for the most important functions of CONASYS in order to delivery contextualized information to the user. Figure 4.13 shows all the PC steps.

The first thing that PC does is to verify if the XML with the request is of a *query* or a *subscription* type and store the type in a session variable. After that, the PC changes its state to *requested*. This is done to internal PC control. The next step is extract the information of the received XML file (see Figure 4.12). The extracted information is also added in session variables.

```

<ns3:ServiceConsult>
  xmlns:ns2="http://service.middleware.compaas.conasys/"
  xmlns:ns3="http://service.gather.middleware.compaas.conasys/"
  <services>
    <service>
      <name>
        City_Pollution
      </name>
    </service>
  </services>
  <tipoOfConsult>
    Subscription
  </tipoOfConsult>
  <constraintConsult>
    <repeatPeriod unit="MS">500</repeatPeriod>
    <duration unit="MS">30000</duration>
  </constraintConsult>
</ns3:ServiceConsult>

```

Figure 4.12 – Example of an XML that CONASYS must interpret.

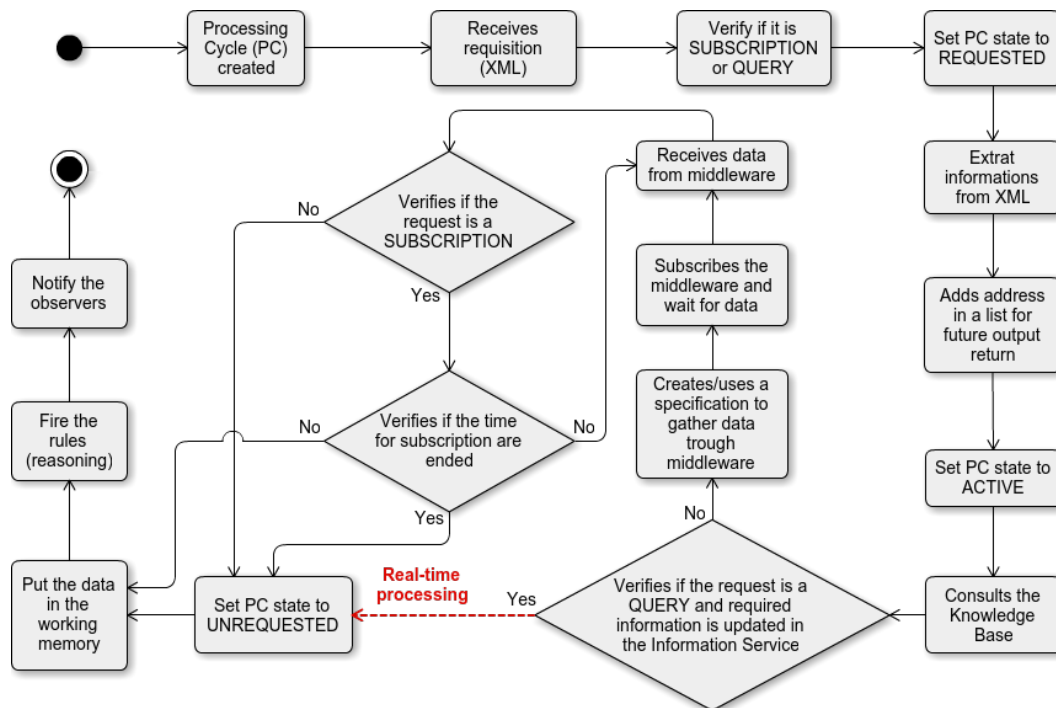


Figure 4.13 – Phases of the Processing Cycle.

The PC implements the “Observer” pattern which adds the address of the user in a list for future notifications. After these steps, the PC changes its state again, this time to *active*. This state change allows PC to move on and to do the future tasks. With the knowledge of the services that user wants, PC consults the CONASYS Knowledge Base which contains all the information of the rules. By the name of the service the PC knows which rules to apply to this service. This is possible because each rule corresponds to a service.

At this point, with the knowledge of the rule that corresponds to the user request, the PC is able to query CONASYS database looking for events generated by the middleware. Indeed, each event data generated by the middleware is stored in the CONASYS database. PC verifies if the database has events related to the devices of the rule. This is a very important point of the PC: if there is a corresponding event in the database and the information is updated (i.e., depends on the device data generation time) and the user request is a *query*, then the real-time processing

occurs. With the real-time processing, PC changes its state to *unrequested* and goes to the context reasoning and context distribution process. If there is no possibility of real-time processing because the request is a *subscription* or the event information does not exist in the database or even it is not updated, the normal flow continues. The real-time processing is like a shortcut of the PC. Table 4.1 shows the main differences between real-time processing and the no real-time flow.

Table 4.1 – Comparison between real-time processing and no real-time Processing Cycle flow.

Real-time processing	No real-time flow
<ul style="list-style-type: none"> <li>- Works only with QUERY</li> <li>- Less steps in the whole process</li> <li>- It takes less time</li> <li>- No middleware access</li> <li>- Do not update the database events</li> </ul>	<ul style="list-style-type: none"> <li>- Works with QUERY and SUBSCRIPTION</li> <li>- More steps in the whole process</li> <li>- It takes longer</li> <li>- Makes a request to the middleware</li> <li>- Update database with new events</li> </ul>

When real-time processing does not occur, the PC needs to get data from the middleware and a specification is needed. In this sense, PC verifies in its database if the specification regarding the rule corresponds to the user request. If there is no specification for the request, then PC creates one. The process of creation an specification is similar to the creation of a CONASYS request and depends if it is a subscription or a query (see Figure 4.10 and Figure 4.11). The creation of a specification is made using the middleware API, and the communication is made through SOAP web service. An example of a specification to get data from middleware can be seen in the Figure 4.14.

The middleware also works with the “Observer” pattern. In this way, the output of the specification is received in the PC *update* method. The next step is wait until the middleware processes the specification and starts notifying data. When PC receives the data it verifies if the request type is *query*. If it is, PC changes state to *unrequested* and goes to the context reasoning and context distribution process. If it is a *subscription*, then the next step is to verify if the *duration* time has ended, and if it has, PC changes state to *unrequested*, and if not, the PC waits the *repeatPeriod* to receive new data from the middleware and puts the data in the CONASYS working memory in order to to be contextualized.

```

<ns3:ReportSpec
  xmlns:ns2="http://service.middleware.compaas/"
  xmlns:ns3="http://service.gather.middleware.compaas/">
  <resources>
    <resource>
      <uri>
        http://192.168.0.10:8080/Air_152/resource
      </uri>
    </resource>
  </resources>
  <constraintSpec>
    <repeatPeriod unit="MS">5000</repeatPeriod>
    <duration unit="MS">30000</duration>
  </constraintSpec>
  <outputSpecs>
    <outputSpec outputName="Spec03">
      <setSpec set="CURRENT"/>
      <groupSpec groupBy="RESOURCE"/>
      <dataOutputSpec operation="AVERAGE"/>
    </outputSpec>
  </outputSpecs>
</ns3:ReportSpec>

```

Figure 4.14 – Example of a specification report.

After both real-time processing and no real-time flow, the PC changes its state to *unrequested* and starts the context reasoning and context distribution process. The first thing of these processes is to put the data in the working memory. The working memory is linked to the process of context reasoning. After that, the rules are fired (i.e., executed). This is the moment in which the data is matched against the set of rules from CONASYS. The rules are in an IF-THEN-ELSE structure. In this sense, if any data reaches the conditions of the rule, the actions of the rule are fired. This statement can produce the contextualization of the data in many levels. An example of a rule can be seen in Figure 4.8. In this example the rule uses data from three different devices and, if the data fits in the conditions of the rule, then a label is inserted in the data indicating a new condition that did not appear before.

The context distribution process is responsible for the provision of the information based on context [17]. It is reached when the data already passed by the set of rules and has context on it, so the distribution occurs when PC notifies its subscribers with these contextualized information. Moreover, there are different ways to send data to the user: Web Service, WebSocket, and HTTP. In this way, the user can change the method of communication by CONASYS API. The user receives data through the *update* method and can use these data in countless ways.

## 5. EVALUATION

This chapter presents the evaluation tests performed in CONASYS. The tests were divided in sections. Each section represents a test scenario with different objectives and methodology.

The main goal of the tests is to evaluate CONASYS in terms of performance and usability. With the tests, we have evaluated the context-aware characteristics of CONASYS. All the tests were performed in a notebook with Ubuntu 14.04 LTS (64-bit), Intel Core i5-3230M 2.60GHz and 8GB RAM. The CONASYS was developed with the programming language Java and we used PostgreSQL as the database.

### 5.1 SCENARIO 1 - MODELLING THE KNOWLEDGE BASE

The CONASYS Knowledge Base has the information of the rules that are inserted in the system. These information will be used in the process of context reasoning. This test scenario is related to the process of modelling the Knowledge Base. The modelling of the Knowledge Base happens in two steps: (i) interpretation of the rules that will be inserted in the system and (ii) store it in an organized way, that facilitates the future search and context reasoning process. The interpretation of the rules consists in cutting each rule in parts based in the rule structure and store the parts in CONASYS database (i.e., conditions, actions as seen in Chapter 2 at Section 2.4.3). This process occurs only when the CONASYS is deployed. This process is linked with the Drools rules that compose the system services. Each available service of the system corresponds to one or more Drools rules.

In CONASYS the rules are separate by domain. Each domain can be linked to a middleware system. In other words, each domain will have an specific set of rules that must be registered in CONASYS. In this test scenario we test the capacity of rule interpretation by CONASYS. The rule interpretation function is part of Knowledge Base (see Figure 4.2). This function has database access in order to store the rule information.

The main goal of this test is to measure the time taken by the system to interpret a Drools file, model the rules of the file, and to store them in the database. With this test, we can analyze the performance of the CONASYS and verify if it is able to deal with an specific IoT environment. Other test objective is to analyze the correct functioning of the system while it manage the rules. In this way, all the rules must be properly interpreted in order to have a fully function rule background for the context reasoning in the next steps.

In order to reproduce a real-world scenario, we create a synthetic Drools rule file with different rules. This file has domain-specific rules with the traditional structure IF-THEN-ELSE. For the tests, we modify this file in order to increment the number of rules. Moreover, the conditions of each rule also increase. For example, we collect the time taken for the system to interpret a Drools file that has 10 rules and each rule has 5 conditions. These conditions are the field *when*

of a rule. A rule with 5 conditions can act up to 5 devices. An example of one Drools rule can be seen in Figure 4.8. The rule of Figure 4.8 only acts on 3 devices as we can see in the *when* field. In the most extreme scenario of this test we have a Drools file with 500 rules and each rule with 100 conditions (i.e., *when* field). With this scenario we can reach, in an ideal setting, a total of 50000 (fifty thousand) devices (i.e., 500 rules · 100 conditions).

We made tests in 15 different scenarios. All the times collected were a average of 10 test executions. The scenarios vary in two terms: number of rules in the file and number of conditions in each rule of the file. The number of rules in the file vary from 10, 50, 100, 200 and 500. The number of conditions on each rule was 5, 50 and 100.

Table 5.1 – Execution time (ms) for modelling the knowledge base.

Rules/Conditions	5	50	100
10	133,3	135,2	138,6
50	583,6	598,9	616,7
100	1164,9	1185,9	1197,1
200	2419,2	2426,2	2624,8
500	6269,1	6348,2	6520,5

The results of the tests can be seen in the Table 5.1. The first column of the table shows the number of rules in the file for each test. The first row of the table shows the number of conditions on each rule of the file. Figure 5.1 shows the data of Table 5.1 in a graph way.

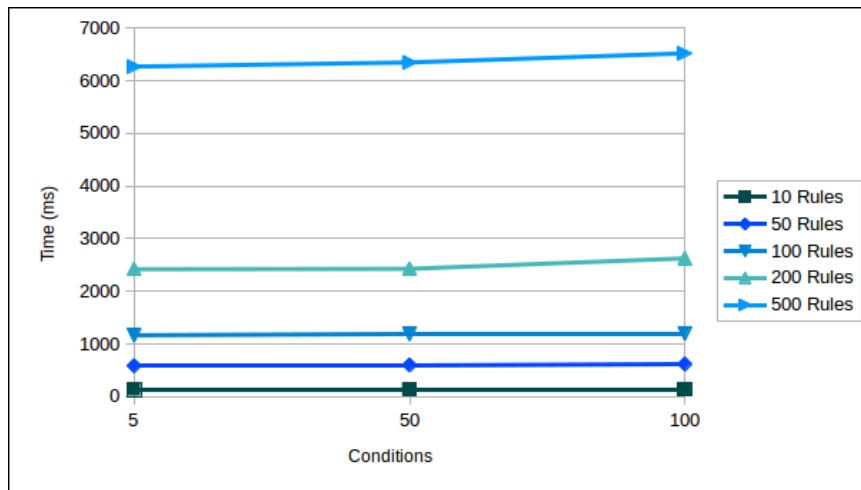


Figure 5.1 – Time taken by the CONASYS function to process rules.

Figure 5.1 shows that independently of the number of conditions, the time taken by the system to interpret and model the rules almost did not change. Considering the test scenario with 100 rules in the file, the time taken by the system to interpret, model and store the rules was 1164,9 ms for 5 conditions in each rule. This scenario covers a total of 500 (five hundred) devices, once each of the 100 rules cover 5 different devices. If we increase the conditions of this scenario to 100, then the scenario will cover a total of 10000 (ten thousand) devices. This conditions raising represents a 20 times (1900%) increase in the number of devices (five hundred to ten thousand),



and the time taken for the function process it suffers a increase of less than 3% in the time (1164,9 ms to 1197,1 ms).

The execution time becomes bigger when the Drools rule file has more rules. Taking account our tested scenario, we have 500 rules in the file and each rule with 100 conditions. For this case, the time taken by the function was 6520,5 ms (6,5 s). This scenario can cover a total of 50000 (fifty thousand) devices, that is a reasonable number of devices for a specific domain. Bearing in mind that this function is only triggered in the deployment state of the CONASYS, it is an acceptable time.

## 5.2 SCENARIO 2 - EVENT AND DEVICE MODELLING

This test scenario is related to the background functions that occur in CONASYS before the contextualization. These functions are essential for the proper functioning of the system. We analyze two specific background functions that are: device and event registration. The devices registration occurs simultaneously when a device registers itself in the middleware. This function interprets the registration, that can be the first registration of the device or an information update (e.g., status, characteristics), model it according to context modelling techniques and store it in a database in a way that context reasoning could occur in some steps forward. The modelling is related to the process of extracting information of a source and work it in a meaningful way. The modelling techniques used in CONASYS are Key-Value and Markup Scheme as shown in Chapter 4.

Event registration process is triggered when any event related to the devices connected to middleware occurs. These events are linked to the data generated by the device. Every time that a device generate data an event is created. The event registration process is responsible for understand the event data and also to store it in the database. Every event generated has a *timestamp* that is stored and used in the real-time processing in order to know how long is the occurrence of the event. Both the device and event registration include database access in order to store the data.

The main goal of the tests is to discover how long these background functions take in order to verify if it is able to be inserted/used in an IoT environment, which is a dynamically place where the decisions and functions have to occur in the most fast way. Other objective is to test the functionality of the functions, in order to be sure that they are playing their role properly.

We made the tests with simulated data that correspond to the devices and events information. These data were the input of the background functions. In this sense, the function receives the data, process it (i.e., context modelling) and stores it in a database. The tests were performed in five phases. In each phase, the number of simultaneous data that were sent to the background functions vary. The tests were made with 10, 20, 50, 70 and 100 parallel devices and events data sent to the functions. We test device and event background functions separately.

Each execution of the test was made ten times in order to have a accurate time measuring. Table 5.2 shows the results of performed tests. The first column shows what background function

Table 5.2 – Execution time (ms) of the background functions.

Parallel Functions	10	20	50	70	100
Devices	276,2	305,7	356,8	414,5	498,4
Events	247,8	289,1	329,9	373,9	409,7

was performed (i.e., devices or events). The first row shows the amount of data that was sent to the functions in a parallel way. Figure 5.2 shows the test results in a graph way, in order to make easy the visualisation of the time spent in each function.

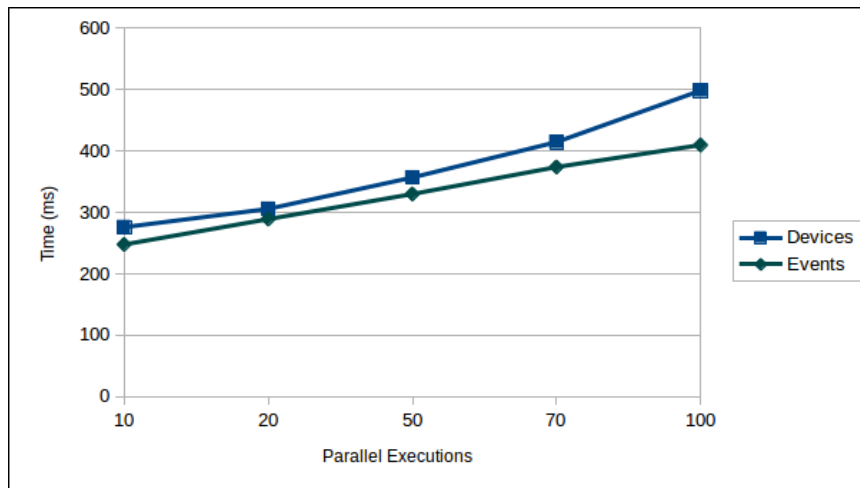


Figure 5.2 – CONASYS background functions execution time variation.

As we can see in Figure 5.2, the tests revealed that CONASYS is able to interpret and model events and devices data in a short time. For example, related to the function that interprets device data, in the worst tested case we have one hundred devices registering to the CONASYS in a parallel way. In this case, the average time taken for the background function interpret and store information of the devices is 489,4 ms for each device. In other words, even with a hundred of devices connecting to CONASYS at the same time, the CONASYS will perform hundred background functions simultaneously, and each function will be done in less than a half second.

For the event registration process, the time is even lower than the device registration. In this case, the best time that we achieve is 247,8 ms when 10 events occurs at the same time. For the worst test case, the time taken for model and register the event is 409,7 ms. The percentage of growing time for the event registration is lower than for device registration. This occurs because an event has less information than a device. An example of device information can be seen in the Figure 4.7. Unlike the device, event information only has an `<uri>` and `<value>` tags.

The tested features are related to the CONASYS background functions. In this sense, the user will not have direct access to these functions, which means that the time taken by these functions will not interfere in the user requests. However, provide efficient time for background functions is an important requirement for CONASYS in order to have a fully updated and functional system.

### 5.3 SCENARIO 3 - REAL-TIME PROCESSING

This scenario tested the real-time processing capability of CONASYS. The real-time processing occurs when the user sends a *Query* request to CONASYS. Moreover, for a real-time processing, the answer for the user request must be registered in the database (i.e., Information Service from Figure 4.2) and in an updated way. The information that is in the CONASYS database is the event related to the devices. An event occurs when a device generates data. The parameter for measuring if the event is updated or not may vary by device to device. This parameter is extracted from the XML file that is used for registering the device in the middleare/CONASYS. In the Figure 4.7 we can see an example of a device XML file and the field `<generationTime>` that is used to check if the device information is updated.

This test scenario measures the time taken by the CONASYS to: (i) verify if there is in the CONASYS database information related to the user request, (ii) verify if the user request can be solved by the real-time processing by checking if the information is updated, and (iii) attribute the information to the response that will be sent to the user. This test do not measure the contextualization time (i.e., context reasoning). The main objective of this test is to measure the time taken by the system to interpret if the user request can be solved with the real-time processing and do the tasks that correspond to this processing.

To measure the time taken by the system to identify when is possible to do the real-time processing and do it in fact we have tested this scenario in different formats. The tests were made with 5 variations. Each of these variations correspond to the number of real-time processing that was fired at the same time in order to know how the function scale. The functions fired at the same time vary from 10, 20, 50, 70 and 100. To collect the time taken for the real-time processing we split this process in three phases. The first, called Preparation, refers to the connection with the database. Second phase is called Data Search, this phase verifies if the information requested by the user is present and updated in the database. Finally, the third phase, called Attribution, is responsible for setting the data into the response package, that will be contextualized (context reasoning) and sent to the user.

Table 5.3 – Execution time (ms) of the real-time processing function.

Real-time Processing	10	20	50	70	100
Preparation	213,1	232,6	282,7	311,5	384,1
Data Search	17,3	17,7	18,5	20,6	23,2
Attribution	7,9	8,4	8,5	11,2	16,5
Total	238,3	258,7	309,7	343,3	423,8

Table 5.3 shows the results of the performed tests. The first column shows the real-time processing divided in three phases. The first row shows how many real-time processing functions were fired at the same time. Figure 5.3 shows the test results in a graphic way in order to make

an easier visualisation of the time spent in each of the phases of the real-time processing, and also highlights how the increase in parallel executions affects the function performance.

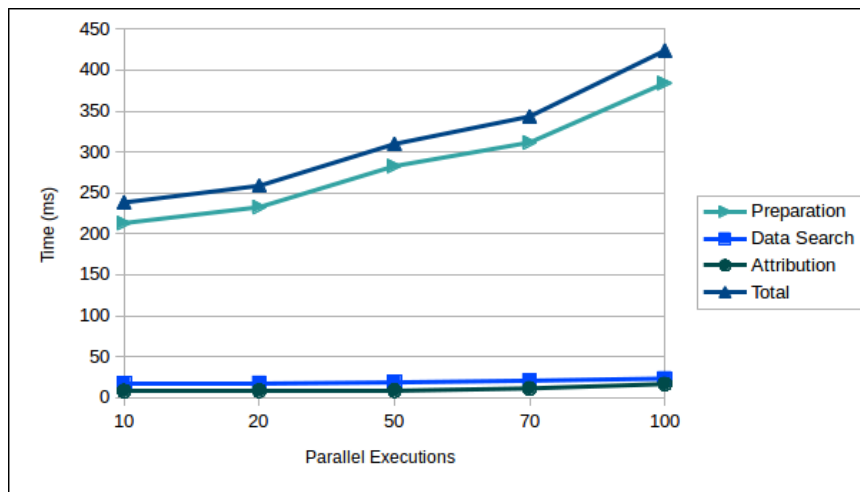


Figure 5.3 – Measuring time of real-time processing function divided in three phases.

It can be noticed in Figure 5.3 that the Preparation phase is the longest phase in the process. It happens due to the database connection step that is necessary in this phase. On the other hand, the other two phases (i.e., Data Search and Attribution) take less time when compared to the Preparation phase. Even in the worst tested scenario, these two phases perform their function in a short time.

With this test scenario we could see that the real-time processing function is scalable. It can be seen by comparing the first execution with 10 parallel functions and the last with 100 parallel functions. In this comparison, the number of parallel executions increased 10 times, while the processing time increased less than 2 times. In the worst case, we have the real-time processing function fired 100 times simultaneously. The time taken for this processing was 423,8 ms, which can be considered real-time [26], since the data (event) can be processed with a very small response time.

## 5.4 SCENARIO 4 - PROCESSING CYCLE

The Processing Cycle (PC) is tested in this section. PC is the heart of CONASYS. The phases of the PC can be seen in more details in Figure 4.13. The main function is the processing of the user request and giving a contextualized response. It is in the PC that the context reasoning process occurs, since it is the core function of CONASYS. In this test scenario, we collected the time taken by all the steps of the PC, from start (i.e., receives the request interpreted) to end (i.e., contextualize and notify observers).

Processing Cycle has many phases. One of them is when the PC must know if the request can be solved in real-time or not. In this sense, the test scenario 5.3 is part of this test scenario that measure the total time taken by CONASYS to do the PC. Besides, the PC also measure the case if

the request can not be solved by real-time processing. In this case, PC must create a specification in order to request data from middleware. Finally, regardless if the flow was normal or real-time processing, the PC must contextualize the data and send it to the user by notification (i.e., Observer pattern).

The main objective of this test is to measure the time taken by CONASYS to perform the PC. The PC is fired with a user request. This request can be a *Subscription* or a *Query*. In order to measure the PC time we use a *Query* request. This is made because with a *Subscription* request the PC must wait a certain period of time (i.e., repeat period and duration) and returns data periodically to the user. On the other hand, the *Query* request do the processing in a most simplified way, just collecting the data and send it to the user. Both *Query* and *Subscription* do all the CONASYS steps including context reasoning.

The tests were performed by collecting the time in the start and end of the PC. For a more realistic scenario, we simulated the users requests and fired it simultaneously. We tested four case scenarios with different numbers of users requests fired at the same time. The number of users requests vary from: 10, 20, 50 to 70. The time was collected in two executions of each case scenario: (i) when the real-time processing occurs (i.e., device information updated in the database) and (ii) when no real-time flow occurs, in this case there are no possibility of real-time processing. The middleware used for the test was an instance of COMPaaS that was deployed at the same computer. We made ten executions of each test scenario in order to have an average execution time.

Table 5.4 – Execution time (ms) of the Processing Cycle with real-time and no real-time flow.

Processing Cycle	10	20	50	70
Real-time Processing	343,7	504,9	1243,3	2232,7
No Real-time Flow	3442,1	4531,8	7560,9	10605,4

The Table 5.4 presents the results of the performed tests. The first column shows the PC type that was performed: real-time processing or no real-time flow. The first row shows how many users requests were fired at the same time, in this way, creating multiple PCs. Figure 5.4 shows the test results in a graphic way in order to make an easier visualisation of the different time taken by the CONASYS to do the PC.

As can be seen in Figure 5.4 the execution time of real-time processing and no real-time flow are very different. In the first test scenario, with 10 parallel executions and real-time processing the obtained time was 343,7 ms, that is almost an instantaneous function. For the others tests, the time taken by real-time PC increases linearly. In all the tests, the contextualization (context reasoning) time is included.

The big difference between real-time and no real-time flow is that when the no real-time flow occurs, is mandatory to have a connection with the middleware. This process of creating or using a specification to access the middleware increases the total time. Moreover, the major fact that impacts the time is the communication with the middleware, which does not happen if it is a

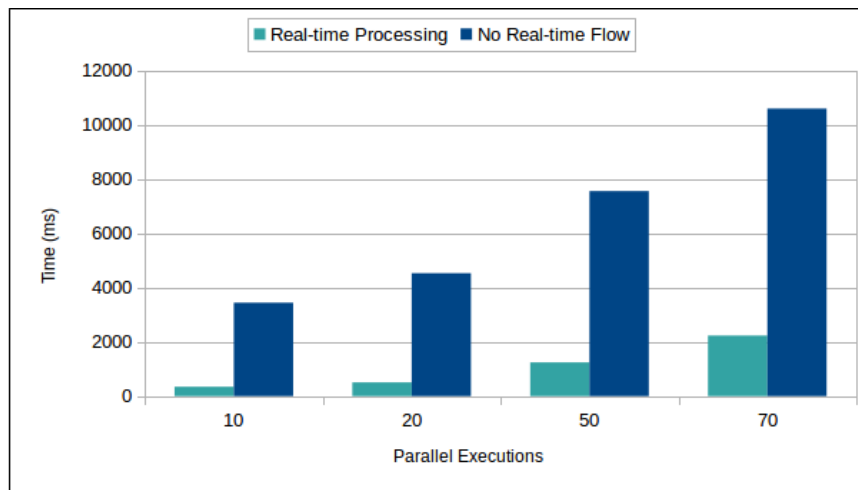


Figure 5.4 – Processing Cycle comparison between real-time processing and no real-time flow.

real-time PC. Even with a larger execution time, the no real-time flow is still a good alternative, taking into account that in the test scenario the device rate of data generating is 1000 ms.

According to [26], real-time processing solutions are focused on processing faster than traditional methods, which allows stream data processing. In the CONASYS scenario, even with a good execution time for parallel executions in the no real-time flow, the real-time processing shows up as an even better alternative, since it meets the IoT needs providing the user response with a very small delay time.

## 6. CONCLUSION AND FUTURE WORK

In this chapter we present the conclusions for this work. First, we present the publications obtained over the Master degree. All the published works have a connection with this work. We also present the conclusions about CONASYS. Finally, we present the future directions and perspectives regarding this work.

### 6.1 PUBLICATIONS

In addition to the study and the development of the proposed work, some papers were written in order to expose and validate the researched ideas. The following list shows all the published works (i.e., conference/journals to which they were submitted) and also their current status. The papers are ordered by publication date:

- Cooperative Middleware Platform as a Service for Internet of Things Applications [3], in *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC 15)*;
- Context-based Framework to Discovery, Search, and Selection of Computing Devices in the Internet of Things [30], in *Congresso da Sociedade Brasileira de Computação - Seminário Integrado de Software e Hardware (CSBC 2015 - SEMISH)*;
- Context-based Search Engine for Industrial IoT: Discovery, Search, Selection, and Usage of Devices [28], in *Emerging Technology and Factory Automation (ETFA 2015)*;
- Context-Aware System for Information Services Provision in the Internet of Things [31], in *Emerging Technology and Factory Automation (ETFA 2015)*;
- The Importance of a Standard Security Architecture for SOA-based IoT Middleware [42], in *Communications Magazine, IEEE*;
- Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G [2], in *Internet of Things (IoT) in 5G Mobile Technologies (Springer Book)*;
- Context-Aware Systems: Technologies and Challenges in Internet of Everything Environments, in *Beyond the Internet of Things: Everything Interconnected (Springer Book)* - To be published.

### 6.2 CONCLUSIONS

The Internet of Things (IoT) is a growing scenario for new applications in the academia and in industry. The IoT added new frontiers to the world communication by embedding processing

power to daily objects, called “Things”. We identified through literature that there are significant amount of middleware system solutions for sensor data management related to IoT, sensor networks, pervasive/ubiquitous computing, and context management fields. However, although the middleware can analyze, interpret, and understand these data, the data will keep useless and without meaning for users and applications.

Context is used to give meaning to the IoT data. With the context, we can give a semantic meaning to the generated data by the “Things”. A context-aware system uses the context in order to provide to users (i.e., requester) services with the context information. These services must be relevant to the user, since the relevance depends of the user needs. In this sense, a context-aware feature is required to address the middleware context challenge.

We have presented a system that give the context-aware capability of IoT middleware solutions and enables to build a sensing-as-a-service platform. The presented system is called CONASYS, and it was joint with the COMPaaS middleware in order to have access to the middleware environment (e.g., devices, events) and also to provide context-aware information services to the middleware users. Each service is linked to business rules (e.g., Drools rules) that processes the IoT data and generates contextualized information.

The characteristics of CONASYS were presented in addition to the modules that compose it. We also have presented an example of use. Moreover, we have presented the technical details of the system implementation. We shown what techniques were used in each step of the contextualization (i.e., acquisition, modelling, reasoning and distribution) and argued about each technique. The Processing Cycle was also presented. This cycle starts when CONASYS receives a user request and triggers all the events in order to answer the request with the contextualized information.

Tests were performed in order to validate the system capabilities. We have collected the results for four scenarios of CONASYS usage. The first scenario was in the function that interprets the Drools rules of the system and use them to create the Knowledge Base. The second scenario was to test the capacity of the system to interpret and store events and devices information. The third test scenario was about the real-time CONASYS feature, that is responsible for understanding when it is possible to perform real-time and execute it. Finally, the fourth test scenario was about the Processing Cycle (PC), that is the heart of CONASYS. We made a comparison with normal flow PC and real-time PC. All the tests shown that CONASYS can be used in an IoT environment because of its usability, scalability, and real-time response.

We also shown some similar work in order to identify how other systems work with the context and how they use it. We have made a comparison between the other systems and CONASYS. The comparison was made through various items that represent the context-aware systems features. With this comparison we could see that CONASYS is a viable choice when compared to other systems as a complete alternative.



### 6.3 FUTURE WORK

In the future, we plan to make some improvements in the system in order to provide better Quality of Experience (QoE) to the users. These improvements are in the visual interface of the CONASYS. We plan to offer to the user more characteristics of the services provided with examples of use, to make easy the use of the services. We also plan to create a tool to enable the creation and update of rules in runtime. In this sense, the user will be able to change the rules scope dynamically.

Context Sharing is largely neglected in the context-aware middleware domain. Most of the middleware solutions or architectures are designed to facilitate applications in isolated factions. Inter-middleware communication is not considered to be a critical requirement. However, in the IoT, there would be no central point of control. Therefore, sharing context information between different kinds of middleware solutions or different instances of the same middleware solution is important. In this sense, we intend to study and to develop a context sharing feature in CONASYS. In this sense, first, we intend to define and to test possible techniques to be used for context sharing considering different environments aspects such as heterogeneity, interoperability, security, and real-time processing. Second, we intend to develop a new feature that will be able to make context sharing considering all the restrictions of the environments.



## BIBLIOGRAPHY

- [1] Abowd, G. D.; Dey, A. K.; Brown, P. J.; Davies, N.; Smith, M.; Steggles, P. "Towards a better understanding of context and context-awareness". In: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, 1999, pp. 304–307.
- [2] Amaral, L. A.; Matos, E.; Tiburski, R. T.; Hessel, F.; Lunardi, W. T.; Marczak, S. "Internet of Things (IoT) in 5G Mobile Technologies". Cham: Springer International Publishing, 2016, chap. Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G, pp. 333–367.
- [3] Amaral, L. A.; Tiburski, R. a. T.; de Matos, E.; Hessel, F. "Cooperative middleware platform as a service for internet of things applications". In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015, pp. 488–493.
- [4] Atzori, L.; Iera, A.; Morabito, G. "The internet of things: A survey", *Computer Networks*, vol. 54–15, 2010, pp. 2787 – 2805.
- [5] Badii, A.; Crouch, M.; Lallah, C. "A context-awareness framework for intelligent networked embedded systems". In: Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services (CENTRIC), 2010, pp. 105–110.
- [6] Bandyopadhyay, D.; Sen, J. "Internet of things: Applications and challenges in technology and standardization", *Wireless Personal Communications*, vol. 58–1, 2011, pp. 49–69.
- [7] Bandyopadhyay, S.; Sengupta, M.; Maiti, S.; Dutta, S. "Recent Trends in Wireless and Mobile Networks: Third International Conferences, WiMo 2011 and CoNeCo 2011, Ankara, Turkey, June 26-28, 2011. Proceedings". Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, chap. A Survey of Middleware for Internet of Things, pp. 288–296.
- [8] Bandyopadhyay, S.; Sengupta, M.; Maiti, S.; Dutta, S. "Role of middleware for internet of things: A study", *International Journal of Computer Science and Engineering Survey*, vol. 2–3, 2011, pp. 94–105.
- [9] Bazzani, M.; Conzon, D.; Scalera, A.; Spirito, M. A.; Trainito, C. I. "Enabling the iot paradigm in e-health solutions through the virtus middleware". In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 1954–1959.
- [10] Bettini, C.; Brdiczka, O.; Henriksen, K.; Indulska, J.; Nicklas, D.; Ranganathan, A.; Riboni, D. "A survey of context modelling and reasoning techniques", *Pervasive and Mobile Computing*, vol. 6–2, 2010, pp. 161–180.

- [11] Bikakis, A.; Patkos, T.; Antoniou, G.; Plexousakis, D. "Constructing Ambient Intelligence: Aml 2007 Workshops Darmstadt, Germany, November 7-10, 2007 Revised Papers". Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, chap. A Survey of Semantics-Based Approaches for Context Reasoning in Ambient Intelligence, pp. 14–23.
- [12] Capra, L.; Emmerich, W.; Mascolo, C. "Carisma: context-aware reflective middleware system for mobile applications", *IEEE Transactions on Software Engineering*, vol. 29–10, Oct 2003, pp. 929–945.
- [13] Chen, G.; Kotz, D. "A Survey of Context-Aware Mobile Computing Research", Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, NH, 2000, 16p.
- [14] Chen, H.; Finin, T.; Joshi, A.; Kagal, L.; Perich, F.; Chakraborty, D. "Intelligent agents meet the semantic web in smart spaces", *Internet Computing, IEEE*, vol. 8–6, Nov 2004, pp. 69–79.
- [15] Conan, D.; Rouvoy, R.; Seinturier, L. "Distributed Applications and Interoperable Systems: 7th IFIP WG 6.1 International Conference, DAIS 2007, Paphos, Cyprus, June 6-8, 2007. Proceedings". Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, chap. Scalable Processing of Context Information with COSMOS, pp. 210–224.
- [16] Corradi, A.; Fanelli, M.; Foschini, L. "Implementing a scalable context-aware middleware". In: IEEE Symposium on Computers and Communications, 2009. ISCC 2009., 2009, pp. 868–874.
- [17] Dey, A. K. "Understanding and using context", *Personal Ubiquitous Comput.*, vol. 5–1, Jan 2001, pp. 4–7.
- [18] Dey, A. K.; Abowd, G. D.; Salber, D. "Managing Interactions in Smart Environments: 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99), Dublin, December 1999". London: Springer London, 2000, chap. A Context-Based Infrastructure for Smart Environments, pp. 114–128.
- [19] Dohr, A.; Modre-Oprian, R.; Drobics, M.; Hayn, D.; Schreier, G. "The internet of things for ambient assisted living". In: Information Technology: New Generations (ITNG), 2010 Seventh International Conference on, 2010, pp. 804–809.
- [20] Filho, J.; Agoulmine, N. "A quality-aware approach for resolving context conflicts in context-aware systems". In: Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on, 2011, pp. 229–236.
- [21] Giusto, D.; Iera, A.; Morabito, G. "The internet of things: 20th Tyrrhenian workshop on digital communications". Springer, 2010, 442p.
- [22] Guo, B.; Sun, L.; Zhang, D. "The architecture design of a cross-domain context management system". In: 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010, pp. 499–504.

- [23] Hall, D.; Llinas, J. "An introduction to multisensor data fusion", *Proceedings of the IEEE*, vol. 85-1, Jan 1997, pp. 6-23.
- [24] Huaifeng, Q.; Xingshe, Z. "Context aware sensor net". In: *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing*, 2005, pp. 1-7.
- [25] jboss.org. "Drools - the business logic integration platform". Accessed: 2015-05-15, Capturado em: <http://www.jboss.org/drools>, 2001.
- [26] Kwon, O.; Song, Y. S.; Kim, J. H.; Li, K. J. "Sconstream: A spatial context stream processing system". In: *2010 International Conference on Computational Science and Its Applications (ICCSA)*, 2010, pp. 165-170.
- [27] Le-Phuoc, D.; Polleres, A.; Hauswirth, M.; Tummarello, G.; Morbidoni, C. "Rapid prototyping of semantic mash-ups through semantic web pipes". In: *Proceedings of the 18th international conference on World wide web*, 2009, pp. 581-590.
- [28] 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: *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, 2015, pp. 1-8.
- [29] Martin, D.; Lamsfus, C.; Alzua, A. "Automatic context data life cycle management framework". In: *2010 5th International Conference on Pervasive Computing and Applications (ICPCA)*, 2010, pp. 330-335.
- [30] Matos, E.; Lunardi, W.; Amaral, L.; Tiburski, R.; Hessel, F.; Marczak, S. "Context-based framework to discovery, search, and selection of computing devices in the internet of things". In: *CSBC 2015 - SEMISH*, 2015, pp. 6.
- [31] 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: *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, 2015, pp. 1-4.
- [32] Nurmi, P.; Floree, P. "Reasoning in context-aware systems". In: *Department of Computer Science, University of Helsinki*, 2004, pp. 6.
- [33] ormfoundation.org. "The orm foundation". Capturado em: <http://www.ormfoundation.org>, Nov 2015.
- [34] Patel, P.; Jardosh, S.; Chaudhary, S.; Ranjan, P. "Context aware middleware architecture for wireless sensor network". In: *IEEE International Conference on Services Computing*, 2009. SCC '09., 2009, pp. 532-535.

- [35] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. "Context aware computing for the internet of things: A survey", *Communications Surveys Tutorials, IEEE*, vol. 16–1, First 2014, pp. 414–454.
- [36] Pietschmann, S.; Mitschick, A.; Winkler, R.; Meissner, K. "Croco: Ontology-based, cross-application context management". In: Third International Workshop on Semantic Media Adaptation and Personalization, 2008. SMAP '08., 2008, pp. 88–93.
- [37] Reetz, E. S.; Tonjes, R.; Baker, N. "Towards global smart spaces: Merge wireless sensor networks into context-aware systems". In: 2010 5th IEEE International Symposium on Wireless Pervasive Computing (ISWPC), 2010, pp. 337–342.
- [38] Román, M.; Hess, C.; Cerqueira, R.; Ranganathan, A.; Campbell, R. H.; Nahrstedt, K. "A middleware infrastructure for active spaces", *IEEE pervasive computing*, vol. 1–4, 2002, pp. 74–83.
- [39] Santucci, G.; et al.. "From internet of data to internet of things". In: International Conference on Future Trends of the Internet, 2009, pp. 1–19.
- [40] Shaeib, A.; Cappellari, P.; Roantree, M. "A framework for real-time context provision in ubiquitous sensing environments". In: 2010 IEEE Symposium on Computers and Communications (ISCC), 2010, pp. 1083–1085.
- [41] Strang, T.; Linnhoff-Popien, C. "A context modeling survey". In: In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004, pp. 8.
- [42] Tiburski, R.; Amaral, L. A.; Matos, E. D.; Hessel, F.; et al.. "The importance of a standard security architecture for soa-based iot middleware", *Communications Magazine, IEEE*, vol. 53–12, 2015, pp. 20–26.
- [43] uml.org. "Unified modeling language (uml)". Capturado em: <http://www.uml.org/>, Nov 2015.
- [44] Wibisono, W.; Zaslavsky, A.; Ling, S. "Comihoc: A middleware framework for context management in manet environment". In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010, pp. 620–627.
- [45] Zaslavsky, A. B.; Perera, C.; Georgakopoulos, D. "Sensing as a service and big data", *CoRR*, vol. abs/1301.0159, 2013.
- [46] Zhang, W.; Hansen, K. M. "Towards self-managed pervasive middleware using owl/swrl ontologies". In: Fifth International Workshop on Modelling and Reasoning in Context. MRC 2008, 2008, pp. 12.