ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

EVERTON DE MATOS

EDGE-CENTRIC CONTEXT SHARING ARCHITECTURE FOR THE INTERNET OF THINGS: CONTEXT INTEROPERABILITY AND CONTEXT-AWARE SECURITY

Porto Alegre

2020

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

EDGE-CENTRIC CONTEXT SHARING ARCHITECTURE FOR THE INTERNET OF THINGS: CONTEXT INTEROPERABILITY AND CONTEXT-AWARE SECURITY

EVERTON DE MATOS

Dissertation submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Fabiano Passuelo Hessel

Porto Alegre
2020

Ficha Catalográfica

D278e de Matos, Everton

Edge-centric Context Sharing Architecture for the Internet of Things Context Interoperability and Context-aware Security / Everton de Matos . – 2020.
144.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fabiano Passuelo Hessel.

1. Internet of Things. 2. Context-Awareness. 3. Context Sharing. 4. Edge Computing. 5. Context-aware Security. I. Hessel, Fabiano Passuelo. II. Título.

Everton de Matos

Edge-centric Context Sharing Architecture for the Internet of Things: Context Interoperability and Context-aware Security

This Dissertation has been submitted in partial fulfillment of the requirements for the degree of Doctor of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on March 26th, 2020

COMMITTEE MEMBERS:

Prof. Dr. Jó Ueyama (ICMC/USP)

Prof. Dr. Jorge Luis Victória Barbosa (PIPCA/Unisinos)

Profa. Dr. Sabrina dos Santos Marczak (PPGCC/PUCRS)

Prof. Dr. Fabiano Passuelo Hessel (PPGCC/PUCRS - Advisor)

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 incondicional e compreensão durante minha pesquisa. Obrigado por estarem ao meu lado durante todos os períodos importantes na minha vida.

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

Agradeço ao meu orientador e amigo, Prof. Fabiano Hessel, pelo incentivo e confiança no meu trabalho e pelos valorosos conselhos durante a caminhada do doutorado.

Agradeço aos meus colegas de projeto e amigos do GSE/PUCRS (Grupo de Sistemas Embarcados) pela ótima convivência durante os anos de projeto e pelas discussões que engrandeceram meu trabalho de doutorado e as pesquisas acadêmicas feitas no laboratório.

Agradeço à Fulbright Brasil pelo crucial apoio financeiro para a realização de meu estágio de pesquisa 'sanduíche' na University of Southern California. Tal período foi de extrema valia tanto pessoal quanto acadêmicamente.

Thank you Prof. Dr. Bhaskar Krishnamachari, Dr. Gowri Ramachandran, and all the Center for Cyber-Physical Systems and the Internet of Things team members at the University of Southern California in Los Angeles for receiving me and giving me the opportunity to work with you during my visiting period.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal Nivel Superior – Brasil (CAPES) – Código de Financiamento 001. Agradeço à CAPES pelo apoio financeiro prestado durante a realização do doutorado que originou esta Tese.

ARQUITETURA DE COMPARTILHAMENTO DE CONTEXTO BASEADA EM EDGE PARA TNTERNET DAS COISAS: INTEROPERABILIDADE DE CONTEXTO E SEGURANÇA ATRAVÉS DE CONTEXTO

RESUMO

A adoção da Internet das Coisas, *Internet of Things* (IoT) em ambientes inteligentes exige avanços para lidar com a grande heterogeneidade das entidades IoT (ou seja, sistemas, aplicativos e dispositivos). A informação de contexto é uma característica essencial dessas entidades, que podem armazenar detalhes relevantes sobre seu ambiente e eventos. No entanto, com a necessidade de integração de diferentes domínios (verticais) da IoT para fornecer soluções relevantes aos usuários, o provimento de contexto de maneira isolada não é suficiente. Os sistemas de contexto atuais fornecem as informações de contexto apenas localmente em domínios específicos (ou seja, abordagens verticais) e não compartilhadas com outras entidades distribuídas, o que é mandatório para se ter uma abordagem de arquitetura de informação horizontal e descentralizada. *Edge Computing*, ou computação de borda, surge como uma abordagem promissora para auxiliar a preencher a lacuna de compartilhamento de contexto, minimizando a sobrecarga de informações e reduzindo a latência de rede. Além disso, a informação de contexto compartilhada é prevista para ser utilizada na tomada de decisões seguras e também na preservação da privacidade do usuário.

Embora algumas abordagens para o compartilhamento de contexto tenham sido investigadas por pesquisadores, não há esforços em torno da definição de uma arquitetura de compartilhamento de contexto centrada na borda para ambientes IoT que cumpra os requisitos de compartilhamento de contexto e forneça recursos de segurança através das informações de contexto. Nesse sentido, este trabalho define uma arquitetura de sistema centrada na borda capaz de realizar o compartilhamento de contexto, que é baseado em uma abordagem de *edge-to-fog* para minimizar a sobrecarga e a latência de rede. Também é apresentada uma extensiva discussão sobre os requisitos para o compartilhamento de contexto, além dos trabalhos relacionados na área. Além disso, a arquitetura proposta é capaz de prover serviços de segurança através do contexto com as informações de contexto compartilhadas, o que é considerado um desafio na área.

Palavras-Chave: Internet das Coisas, *Context-Awareness*, Compartilhamento de Contexto, *Edge Computing*, Segurança baseada em Contexto.

EDGE-CENTRIC CONTEXT SHARING ARCHITECTURE FOR THE INTERNET OF THINGS: CONTEXT INTEROPERABILITY AND CONTEXT-AWARE SECURITY

ABSTRACT

The adoption of the Internet of Things (IoT) in smart environments demands advances to cope with the large heterogeneity of IoT entities (i.e., systems, applications, and devices). Context information is an essential characteristic of these entities, which can store relevant details about their environments and related events. However, with the integration of different IoT vertical domains to provide more relevant solutions to users, providing isolated context is no longer enough. Current context systems provide context information only locally in specific domains (i.e., vertical approaches) and not shared with other distributed entities, which is mandatory to have a horizontal and decentralized information architecture approach. Edge computing emerges as promising approach to help filling the context sharing gap by minimizing information overhead and reducing network latency. Also, the shared context information is envisaged to be used for secure decisions and privacy-preserving.

Although some context sharing approaches have been investigated by researchers, there are no efforts around the definition of an Edge-centric Context Sharing Architecture for IoT environments that mitigates the context sharing requirements and provides context-aware security feature. In this sense, this work defines an edge-based system architecture able to make context sharing, which is based on an edge-to-fog approach to minimize network overhead and latency. It also discuss the requirements for context sharing and the related work in the area. Moreover, the proposed architecture is able to provide context-aware security services with the shared context information, which is considered a gap in the area.

Keywords: Internet of Things, Context-Awareness, Context Sharing, Edge Computing, Context-aware Security.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

ABAC – Attribute-Based Access Control

ADSL – Asymmetric Digital Subscriber Line

API – Application Programming Interface

CARBAC – Context-Aware Role Based Access Control

CAS – Context-Aware Security

CEP – Complex Event Processing

COAP – Constrained Application Protocol

DAC – Discretionary Access Control

DTLS – Datagram Transport Layer Security

ECA – Event-Condition-Action

EMS – Emergency Medical Services

GDPR – General Data Protection Regulation

GPS – Global Positioning System

HIOT – Healthcare Internet of Things

HTTP – Hypertext Transfer Protocol

IIOT – Industrial Internet of Things

IOT – Internet of Things

IP – Internet Protocol

JSON – JavaScript Object Notation

LBS – Location-Based Service

LPWA – Low-Power Wide-Area

LTE – Long Term Evolution

M2M – Machine-to-Machine

MAC – Mandatory Access Control

NFC – Near Field Communication

NLTK – Natural Language Toolkit

OS – Operating System

OWL – Web Ontology Language

RAC – Rule-Based Access Control

RAM – Random-Access Memory

RBAC – Role-Based Access Control

REST – Representational State Transfer

RFID – Radio-frequency identification

SDN – Software-Defined Networking

SHS – Smart Home Systems

SOA – Service-Oriented Architecture

SOAP – Simple Object Access Protocol

SSN – Semantic Sensor Network ontology

TCP – Transmission Control Protocol

TLS – Transport Layer Security

TSDB – Time Series Databases

UDP – User Datagram Protocol

URL – Uniform Resource Locator

VPN – Virtual Private Network

W3C – Wide Web Consortium

WSN – Wireless Sensor Network

XML – Extensible Markup Language

CONTENTS

1. INTRODUCTION

The Internet of Things (IoT) has gained significant attention in academia as well as in industry. By embedding mobile networking and information processing capability into a wide array of gadgets and everyday items, the Internet of Things has been adding new dimensions to the world of information and communication technology [10].

The Edge Computing paradigm enables moving this IoT computation from the high-powered central Cloud to the edge of the network [131]. The benefits of Edge Computing result from its proximity to data sources and end users. It has the potential to address the following challenges: (i) low and predictable latency for end users and applications; (ii) secure and privacy-preserving services and applications; (iii) long battery life and low bandwidth cost; and (iv) scalability [122].

As miniaturization still continues and computing capacity still increases, edge sensors (IoT devices) become more powerful. There is a common sense that the devices of IoT environments generate a lot of data, and also that they are only useful if it is possible to analyze, interpret and understand these data in a proper way. In this sense, context-aware computing has played an important role in tackling this challenge in previous paradigms, such as mobile and pervasive computing, which implies that it would continue to be successful in the IoT as well [109].

Most of the systems or architectures for context management (context-aware platforms) are designed to facilitate applications in separate domains. However, in the IoT, most solutions are deployed in heterogeneous domains, which means different systems developed by various industries can be employed in the environment to connect different sensors and devices, and thus collect, model, and also reason about the environment context, producing context information. These information is defined as the data that characterizes the entities of the environment (e.g., location, status, updates). Therefore, sharing context information between different kinds of systems has become a mandatory requirement in the IoT ecosystem [1] [109].

There is a need to define an architecture that goes beyond vertical solutions by integrating all required technologies and components into a common, open and multi-application platform [18]. Some technologies as ontologies, lexical analyze, and semantic processing have been used for a similar processing that can also be used for such kind of processing as well. Although some approaches for sharing the context have been investigated, there are no efforts around the definition of an Edge-centric Context Sharing Architecture for IoT environments. Such definition demands the mitigation of requirements that are not addressed in existing studies, such as large heterogeneity, scalability and real-time sharing. Also, the use of shared context information for providing context-aware security decisions is not well explored by researchers [109].

1.1 MOTIVATION

The context information can be achieved in many domains (i.e., verticals) of IoT environments. In this sense, the context sharing processing occurs in different domains as well. Figure 1.1 presents a motivation scenario that shows the vision of the proposed Context Sharing Architecture in a smart city domain. Heterogeneity is always present in a smart city, once it is a very complex scenario that contains entities of different domains, such as transit, public services, users. There is a need for a Context Sharing Architecture to make a common context communication.

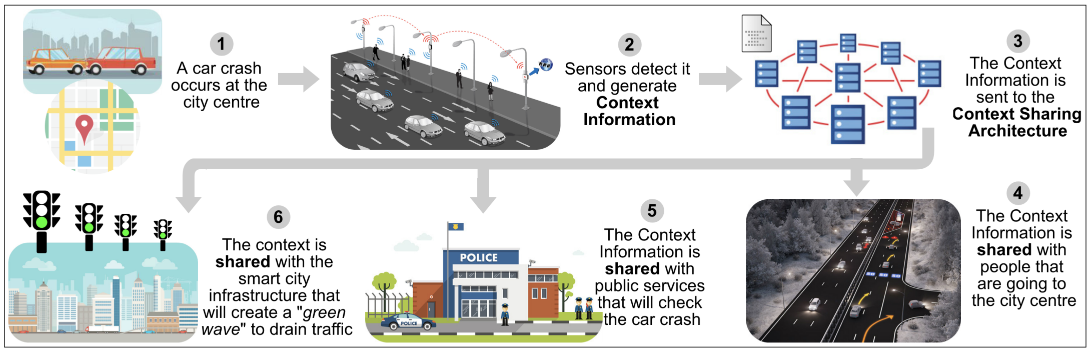

1
A car crash occurs at the city centre
2
Sensors detect it and generate **Context Information**
3
The Context Information is sent to the **Context Sharing Architecture**
4
The Context Information is **shared** with people that are going to the city centre
5
The Context Information is **shared** with public services that will check the car crash
POLICE
6
The context is **shared** with the smart city infrastructure that will create a "*green wave*" to drain traffic

Figure 1.1 – Context Sharing Architecture usability in a Smart City application scenario.

To make clear the organization of a context sharing platform, Figure 1.1 shows how the context sharing feature can fit in an Internet of Things environment (e.g., smart cities). It shows an example scenario in which the context information generated in one domain (e.g., traffic) is shared with different domains (e.g., public services and citizens). The context can be produced with data from different devices (e.g., monitoring camera, light pole) and it should be understood by the destination domain and its devices (e.g., ambulances and traffic lights). In Figure 1.1, the context sharing feature is provided by a context sharing platform. Such platform represents a software system able to interconnect different domains by sharing its context information.

By receiving context information of entities that are not strictly connected each other, the phases 4, 5, and 6 (see Figure 1.1) are directly related to context sharing process. The received context information can be used in a different kind of processing. For example, the entities of phases 4, 5, and 6 can process the received context, create a new context, and share it by the Context Sharing Architecture.

The use of context sharing enables computational entities such as agents and services in pervasive computing environments to have a common set of concepts about context while interacting with one another [143]. By reusing well-defined context of different domains, it is possible to compose large-scale context information without starting from scratch [18].

Besides of sharing context information, such platform can be used to reduce the processing effort of the entities, once they receive context information instead of reasoning about it.

The decentralization of such an approach by having multiple instances in different application domains fits with the Edge Computing principle. The Context Sharing Architecture can be present as a Fog Computing device in the smart traffic monitoring system, and as an Edge Device in each light pole of those systems. The Edge Devices will generate the context information, send it to the Fog, which will share with Fog instances in different domains.

Moreover, the shared context information can be used to provide secure decisions, called context-aware security. For example, information can have its privacy defined by the context of each domain that will receive it, as an ambulance near to the location may receive all the details of the patient. The received context information can be matched with a historical one in decision-making methods to provide context-aware security decisions, such as authentication, authorization, access control, and privacy-preserving.

1.2 HYPOTHESIS AND RESEARCH QUESTIONS

This dissertation seeks to investigate two hypotheses: (i) context sharing is well suited to provide context interoperability among different systems of IoT environments; and (ii) it is possible to provide security based on context to IoT entities through context sharing feature.

Research Question: *"Which requirements a system that produces context information must have to be prepared to share its context information with other entities?"*

Research Question: *"How can the context information interoperability between heterogeneous IoT platforms that produce different kinds of context be provided?"*

Research Question: *"What criteria will be taken into account to define which entities will receive the shared context information and which entity will perform this control?"*

Research Question: *"In which ways can shared context information be used for context-based security provisioning and how it can be implemented?"*

1.3 OBJECTIVES

The main goal of this research is to define an Edge-centric Context Sharing Architecture, thus providing context-aware security. The architecture must cover different points related to managing the context information and also to share it. To achieve the research goal, the following objectives were defined:

- Studying the existing studies for sharing information on Internet of Things environments and their fitness to the Edge Computing paradigm;
- Definition of a data model to express and store the context information in an interoperable way, and also some strategies, protocols, and communication languages to enable this interoperability;
- Implementation of an architecture able to manage and process the context information of different entities in an interoperable way to have a pool of abstractions for context management systems that are sharing their information;
- Definition of an approach to transport and store the context information in an organized and secure way;
- Development of an environment to run tests of the architecture, considering the possibility to use the shared context information to provide context-aware security to the entities;
- Evaluation of the proposed system in scenarios which the context information can be used to provide different categories of context-aware security;
- Documenting and reporting the study results, publishing them in scientific conferences and journal articles, making publicly available all source code involved in this research.

1.4 CONTRIBUTION

The vision of 2020 and beyond also includes a great deal of growing use cases with massive number of devices (e.g., sensors, actuators and cameras) with a wide range of characteristics and demands. Smart services will become pervasive in urban areas, and usage will also grow in suburban and rural areas. Among others, metering (e.g., gas, energy, and water), city or building lights management, environment (e.g., pollution, temperature, humidity, noise) monitoring, and vehicle traffic control represent prominent examples of services in a smart city. The aggregation of all these services leads to very high density of devices with very different characteristics expected to be combined in a common communication and interworking framework [71].

An Edge-centric Context Sharing Architecture will be of great impact in such scenarios. The architecture fits in the definition of a common framework, by exchanging context information of devices placed in different verticals (i.e., domains). Its feature on working at Edge Computing layer makes possible that devices context be shared without the necessity of a centralized server at a Cloud layer.

This research presents two main contributions: (i) the development of an Edge-centric Context Sharing Architecture, able to provide context information interoperability, and (ii) the provision of security based on context through the sharing architecture. To the best of our knowledge, a context sharing architecture that works with highly heterogeneous IoT environments was not defined yet. In this sense, the novelty will be reached at the scientific research community. The specific contributions are:

- State-of-the-art of the projects working with context sharing feature;
- An interoperable model allowing sharing of context information between heterogeneous IoT entities;
- An architecture to perform context sharing in Internet of Things constrained environments, including heterogeneity and security challenges;
- An Edge-centric module able to manage and process context information in constrained environments;
- A method to interpret shared context information in order to provide security and privacy for IoT entities in different ways.

1.5 DOCUMENT STRUCTURE

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, Edge Computing, and context-awareness. Chapter 3 presents the related work in sharing platforms, and context-aware security. Chapter 4 presents the proposed work architectural view, showing the developed modules overview. Chapter 5 presents the technical details of the developed Context Sharing Architecture. Chapter 6 presents the experimental results obtained by testing the proposed approach. Finally, Chapter 7 presents the conclusions, and author's publications in the last years.

2. THEORETICAL BACKGROUND

2.1 INTERNET OF THINGS AND EDGE COMPUTING

The Internet of Things (IoT) is a computing paradigm that is rapidly gaining space in scenarios of modern information and communication technologies. The idea of the 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 neighbor elements and systems to reach common goals through unique addressing schemes and reliable communication media over the Internet [10].

Implementation of IoT environments is usually based on a standard architecture consisting of several layers [13]: from the data acquisition layer to the application layer. Next, the IoT layers functionality is presented [13]:

- **Application layer:** This layer 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.
- **Middleware layer:** This layer acts as an interface between the hardware layer and the application layer. 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. There are different Middleware platforms developed for the Internet of Things environments [116]. It is common for the researchers to define its Middleware platform to meet their project‘s needs. However, some projects, as FIWARE[1], provides a curated open source framework for such kind of needs. Moreover, FIWARE has an infrastructure of different modules that can be used for a personalized Middleware service provision.
- **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.
- **Device 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

[1]https://fiware-orion.readthedocs.io/en/master/

are the primary goals of these devices, leaving the processing activities for the upper layers.

Device layer refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services. The constant increase in data volume elevates the complexity and costs of transporting, analyzing, and storing data at the Cloud. To mitigate these shortcomings, Fog and Edge Computing paradigms have been introduced [20][122].

IoT environments may have different processing layers, including Fog and Edge computing layers. Fog and Edge Computing are firmly related concepts, but they are not the same [133][22][108][107][94][99]. According to the OpenFog Reference Architecture [107], Fog computing extends Cloud computing into an intermediate layer close to IoT devices and enables data processing across domains, while Edge computing involves the control and management of a standalone endpoint device individually within the Fog domain, typically within a close proximity of the device [25][46][84]. It is prevalent in context-aware IoT environments to produce the context information at the Edge layer and enrich them in the Fog layer. It may happen because the Fog layer may access data from other nodes, making it possible to fuse different information.

To avoid any possible misunderstanding about the concepts of Fog and Edge, the definitions from the OpenFog project and Morabito et al. were adopted for the contextualization of those concepts in this work [107][94]. Thus, it is possible to define three entities for IoT environments: (i) edge devices, (ii) fog nodes, and (iii) cloud servers. Figure 2.1 illustrates these entities divided into layers. The edge devices make part of the Edge layer; It is composed of different IoT devices. These devices usually have sensors attached to it responsible for sensing the environment and for data generation. The edge devices may also have the responsibility of data pre-processing and sometimes local decision making. They are characterized by having limited processing power and in some cases, limited energy supply (i.e., battery). Some examples of edge devices are IoT platforms (e.g., Raspberry Pi, Arduino), sensors, actuators, smartphones. The fog nodes make part of the Fog layer. They have the main function of processing data that could not be processed at the Edge layer, which may occur due to the limited resource capacity of the lower layer (i.e., Edge). Also, fog nodes may store edge devices data. It is usual for the fog nodes to be placed physically close to the edge devices (i.e., from the same room to the same city). Some examples of fog nodes are small servers and personal computers. Even a powerful edge device sometimes may be considered a fog node (e.g., last generation smartphones/tablets connected to an energy supply). In the Edge-Fog approach, the cloud servers, placed at the Cloud layer, work as information storage. They usually do not process data, working as a larger database. The cloud servers may store data (e.g., context information) from a few fog nodes and several edge devices at the same time.

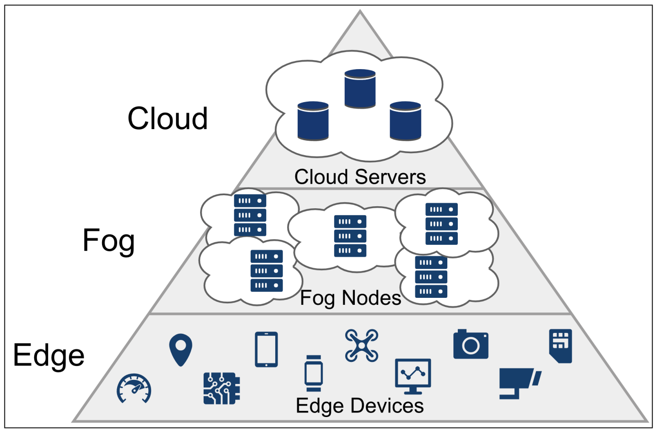


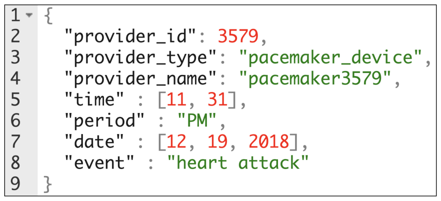
Figure 2.1 – Overview of different layers present in IoT environments.

2.2 CONTEXT-AWARENESS AND IOT

Context, sometimes referred as context information, is commonly represented semantically [39]. It is used to define the status of an environmental entity (e.g., person, place, application, or computing device), thus characterizing its situation [145][1]. Context information is highly related to the information that is easily understandable by humans when reading it [109].

Abowd et al. [1] have introduced a way to characterize the situation of the entities, that is used until these days by most of context information management solutions [109]. It is based on the “Five Ws” approach, which uses five questions: Who, Where, When, What, and Why. Those questions are made to build the context information. The question “Who” can characterize the identity of the entity. By asking “Where” it is possible to discover the location. The question “When” gives a notion of time. The “What” can characterize an activity. Finally, by asking “Why,” it is possible to determine the motivation. In light of this, the information is expected to have at least one of the “Five Ws” within it to be considered a context.

Even with the well-known definitions of what is considered context information, there is no standard format and representation for it [109][39][145]. Different researchers have identified different ways to present context based on different approaches. Abowd et al. [1] introduced one of the most popular ways to define the context (i.e., the “Five Ws” approach). They defined two types of context: primary and secondary. The primary context is identified as location, time, identity, and activity. Further, the secondary context can be achieved by using the primary context [109]. For example, when considering that the primary context is both the GPS (Global Positioning System) coordinates of a user’s device (e.g., smartphone) and the time of the day, it is possible to achieve the secondary context as being the events that may occur on that particular area, or the traffic status. Figure 2.2 shows an example of this representation of a primary context, representing the data from a patient's pacemaker connected to the patient's smartphone. Figure 2.3 shows an example of the secondary context, which is the enriched location information (e.g., city, zip code) that can be used to send an alert to the Emergency Medical Services (EMS). It represents a primary context enriched with a patient's phone information, thus creating a more complex context.



```
{
  "provider_id": 3579,
  "provider_type": "pacemaker_device",
  "provider_name": "pacemaker3579",
  "time" : [11, 31],
  "period" : "PM",
  "date" : [12, 19, 2018],
  "event" : "heart attack"
}
```

Figure 2.2 – Example of a primary context, generated by a pacemaker device, of a patient having a heart attack in JSON format.

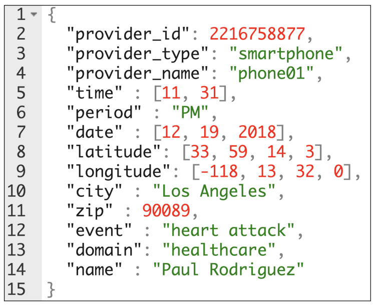
The FIWARE introduced another example of a popular way of representing context information, the Orion Context Broker[2] project. It organizes the context information more straightforwardly by just defining one type of context, that encompasses both primary and secondary ones when compared with the Abowd et al. representation [1]. Figure 2.4 shows an example of FIWARE - Orion Context Broker context representation.

Recently, Casadei et al. [27] stated that the context information is a fundamental piece of the IoT environments. The authors classify the IoT environment in three main classes of entities: IoT Entity, IoT Environment, and IoT Service. The IoT Entity is any subject that either produces or consumes IoT Services. The IoT Environment is the physical place where the IoT Entities are deployed. IoT Services are the cyber-physical services provided by the IoT Entities. The context is stated as the dependencies between those three classes of entities. It can express implicit or explicit information regarding them.

As shown in Figure 2.2, Figure 2.3, and Figure 2.4, the context information tends to be presented in an easily understandable form for the final user, for example, in a JSON (JavaScript Object Notation) or in an XML (Extensible Markup Language) format.

In this work, it is defined a formal way of representing context information, not considering the metadata format, but the content that it should have. Let's consider a set of Entities $E = \{e_1, e_2, ..., e_n\}$ and a set of Status $S_n = \{s_1, s_2, ..., s_m\}$. An entity e_n represents a person, place, application, or computing device, and has a set of Status S_n composed of information that characterizes the entity e_n. A common way of characterizing an entity is by using the "Five Ws" approach [1]. Thus, a status s_m should be represented by at least one of the "Five Ws" characterization about the entity e_n. Taking this into account, a context

[2]https://fiware-orion.readthedocs.io/en/master/

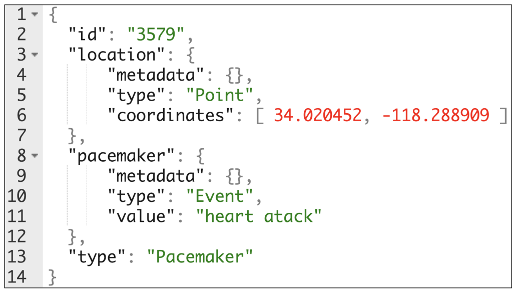


```
{
  "provider_id": 2216758877,
  "provider_type": "smartphone",
  "provider_name": "phone01",
  "time" : [11, 31],
  "period" : "PM",
  "date" : [12, 19, 2018],
  "latitude": [33, 59, 14, 3],
  "longitude": [-118, 13, 32, 0],
  "city" : "Los Angeles",
  "zip" : 90089,
  "event" : "heart attack",
  "domain": "healthcare",
  "name" : "Paul Rodriguez"
}
```

Figure 2.3 – Example of a secondary context from a patient having a heart attack in JSON format.

information is defined as ci_{nm}, where n is an entity id and m an information id about an entity e_n. A set of context information is denoted as $C = \{ci_{nm1}, ci_{nm2}, ..., ci_{nmk}\}$.

To provide context information, a system must follow some steps. Perera et al. [109] defined Acquisition, Modelling, Reasoning, and Distribution as the steps for a system to provide context information, naming as context life-cycle. The Acquisition refers to gather the raw data from a sensor, database, or from the environment. The Modelling process adjusts the data in a specific format to turn its input for the Reasoning step. There are many different techniques for Modelling already surveyed in existing literature (e.g., key-value pairs, ontology, markup scheme) [15][128]. The Reasoning process is the primary step in the context life-cycle. It transforms the information into a context, denoted as ci_{nm}, turning it understandable to the final users. The Reasoning, also called inferencing, may use different data enrichment techniques (e.g., business rules, ontology, probabilistic, data fusion, aggregation). Perera et al. also detail the main Reasoning techniques and show in detail how each technique works. Distribution is a straightforward step [109]. It is responsible for spreading the context information. Usually, it has the option to distribute context by direct query or subscription.



```
{
  "id": "3579",
  "location": {
      "metadata": {},
      "type": "Point",
      "coordinates": [ 34.020452, -118.288909 ]
  },
  "pacemaker": {
      "metadata": {},
      "type": "Event",
      "value": "heart atack"
  },
  "type": "Pacemaker"
}
```

Figure 2.4 – Example of a FIWARE - Orion Context Broker context from a patient having a heart attack in JSON format.

A system can be considered context-aware when it uses the context obtained through the context life-cycle to provide useful services/information to the user [1][90]. In this way, it is indispensable to the IoT environments to have a context-aware system able to reason about the environment to provide such kind of services/information. Thus, context-awareness is considered a must-have feature to IoT systems [39].

The context information may vary depending on the producer. It can vary in format, size, representation. Most of the context-aware systems produce such kind of context information and use it only locally for decision making or spread it directly to the final user. However, there are some systems that could share context information with whom may be interested. This process is called context sharing and is one of the main challenges of the context-awareness area [109][1][18].

2.3 CONTEXT SHARING

Before going into context sharing details, it is important to separate its definition from data interoperability, i.e., data sharing. Although having a similar concept of delivering common understanding information for two different entities, both data interoperability and context sharing differ in some aspects. There are some data interoperability platforms for IoT already studied and developed by the scientific community, such as FIESTA-IOT [2], and IoT-A [37]. Their main goal is to provide a way to make IoT device data interoperable between different applications and users. However, context sharing platforms may work with context, that can be considered a more complex information [109].

Context information sharing needs a more careful process than sharing regular data. This data may be sensitive, making it essential to care about its security. For example, if an attacker gets regular data from a communication channel, it can be tough to understand the meaning of such data without the right context. Differently, the context information represents a specific event, many times in a semantic manner. Thus, it is much more understandable for a possible attacker. Moreover, there are many different ways to provide context information [109][39]. In light of this, it is very common that heterogeneous context information providers act in different ways when generating context, varying in its format, length, data type, representation. Thus, these variations should be considered when providing context sharing feature. Therefore, context sharing platforms tend to have an enormous effort in providing context interoperability, many times by using different techniques (e.g., rules, ontology, decisions trees).

As IoT has many heterogeneous environments with different devices generating context information, it is essential to share context information between entities. More than helping in a common understanding of the context information, the context sharing feature may also help in reducing the effort of the entities. The receiving entity can get the context information without performing the reasoning process, which is considered the most demanding hardware operation in context life cycle [109].

The context sharing feature can be performed embedded in an IoT entity or by a third-party software. The software system that performs the context sharing feature can be called architecture, platform, tool or mechanism.

For example, an event that can occur in a smart city generates context information that is automatically shared by the context sharing platform to whom may be interested in it. The sharing process includes heterogeneous entities. Thus, the received context information can be used in different kinds of processing. The entities can process the received context, create a new context, and share it back with the platform.

A context sharing platform may vary in its characteristics (i.e., features). For example, it can have decentralized or centralized processing. Next section, explains in details the different features (i.e., the building blocks) that a platform must have to share context information.

2.4 SHARING BUILDING BLOCKS

It is well-known that IoT environments have complex application scenarios. A context sharing platform must deal with these scenarios by implementing some specific functions. These functions are also called building blocks. In this work, it is defined the context sharing building blocks taking into account some past research in the context-awareness and context sharing areas [109][103][74]. The following building blocks are used to compare the different context sharing platforms: Modeling (M), Reasoning (R), Data Dissemination (D), Privacy (P), Interoperability (I), Context Processing (CP), Infrastructure Configuration and Management (ICM), Scalability and Real-time sharing (SRT), Availability (Av), Communication technologies (C), History (Hi), and Architectural model (Ar).

The building blocks can be categorized as: (i) *Properties*, and (ii) *Architectural Components*. The *Properties* refer to the ones mainly related to the software side of the context sharing platforms. The *Architectural Components* refers to the architectural decisions when deploying a context sharing platform, mainly related to the hardware side (e.g., communication technologies, storage space, processing layers). Figure 2.5 shows the building blocks and its divisions in a taxonomy view. Moreover, as shown in Figure 2.5, some building blocks are strictly related to the context sharing characteristics, which means the ones exclusively need in such processing. On the other hand, some building blocks are related to regular IoT characteristics, which are common in IoT systems that do not necessarily originate from the context sharing feature.

Figure 2.6 shows all the different building blocks and how they fit and interact together in an Internet of Things Environment. It is common sense that most of the building blocks are a responsibility of the Context Source entity that will share the context information. However, it opens new possibilities when some building blocks are implemented by the

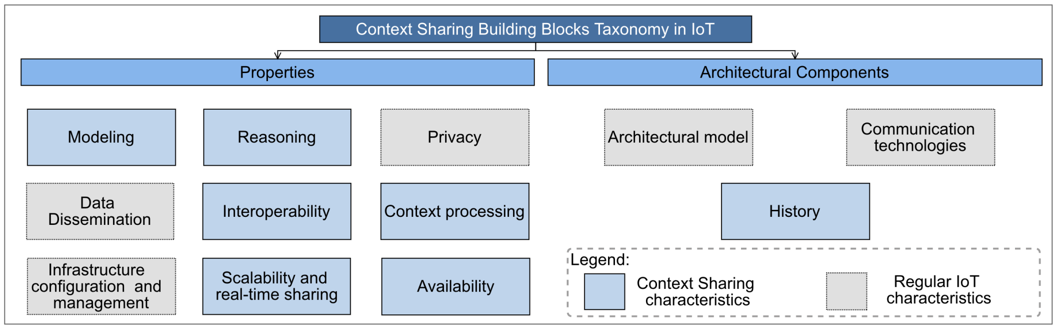

Context Sharing Building Blocks Taxonomy in IoT
Properties
Architectural Components
Modeling
Reasoning
Privacy
Architectural model
Communication technologies
Data Dissemination
Interoperability
Context processing
History
Infrastructure configuration and management
Scalability and real-time sharing
Availability
Legend:
Context Sharing characteristics
Regular IoT characteristics

Figure 2.5 – The context sharing building blocks taxonomy.

Context Destination, that will receive the context information. For example, if the Context Destination has a Reasoning (R) function, it can produce new context information to perform a new task.

The following paragraphs present the context sharing building blocks that a context sharing platform must have. In this sense, this section defines these building blocks and the enabling technologies for each one, while Table 3.2 (see Section 3) makes a comparison with many systems architectures through the defined building blocks.

2.4.1 Properties

- **Modeling (M)**: The modeling is the first step for context standardization. It is responsible for converting the context into a predefined format. The modeling process helps in a more straightforward interpretation of the context information. An efficient modeling process is essential for a context sharing platform. Researchers already surveyed the most popular techniques for modeling context information [30][130][109]. These surveys present the techniques and different ways of implementation for each one. To choose for a specific modeling technique will depend on the deployed site characteristics, once each technique may be suitable for a particular situation. A given system may employ one or more modeling technique. The works are classified by the following modeling techniques: key-value modeling (Key), markup schemes (Mrk), text-based modeling (Tex), graphical modeling (Gra), object-oriented modeling (Oob), logic-based modeling (Lob), and ontology-based modeling (Onb). The symbol (✓) is used to denote that the work employs the modeling feature, but it does not make clear the specific technique.

- **Reasoning (R)**: It is defined as the process to obtain high-level information from less enriched data, or even raw data. The reasoning process uses the available context to produce a more useful one. The outcome of the reasoning process could be semantic information, being easily understood by final users. Moreover, the reasoning also is defined as the inference process [16]. In the scope of context sharing, the reasoning is used to discover/infer the destination of the context information for sharing. For example, this feature may be used to discover that a context of a patient having a heart attack must be sent to the Emergency Medical Services (e.g., ambulances) of the city. Moreover, the reasoning can be also performed by the entity that will receive the context in order to do a new processing (e.g., decision making, inferencing). The most popular reasoning techniques are surveyed in [109]. The works are classified by the following reasoning techniques: supervised learning (Sul), unsupervised learning (Unl), reinforcement learning (Rel), rules (Rul), fuzzy logic (Fuz), ontology-based (Onb), and probabilistic reasoning (Pro). The symbol (✓) is used to denote that the work employs the reasoning feature, but it does not make clear the specific technique.

- **Data Dissemination (D)**: The context information is shared to who is interested in it. Data dissemination is a fairly straightforward task. Each system has a policy to disseminate the context information. There are only two ways of data dissemination: static (Sta), and dynamic (Dyn). In the static approach, there is a predefined list to whom the context must be sent depending on the specific situation. On the other hand, the dynamic approach needs a specific reasoning function to define the exact destination of the context information. For example, in the static approach, there will be a predefined list of the city ambulances, while in the dynamic, a reasoning method will find the nearest available ambulance. For both approaches, the dissemination may occur by groups, roles, or individually.

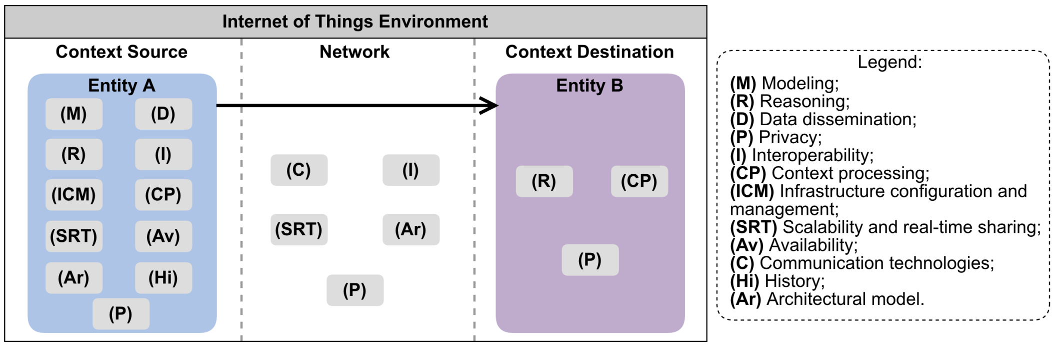


Figure 2.6 – The context sharing building blocks and its interaction.

- **Privacy (P)**: The context information includes private data in most situations such as location, activities, and preferences. Thus, privacy is an important role. Different approaches can be used to ensure privacy, such as authorization, access control policies, anonymization, cryptography [137]. Concerning privacy protection, it is necessary to specify what information may be disclosed, providing means to trace and destroy the information, if necessary. The symbol (✓) is used to denote works that employ techniques to ensure privacy.

- **Interoperability (I)**: Differently from the most context systems that comprise only an application-specific system (i.e., a vertical domain), shared systems tend to be more heterogeneous. In the context sharing platforms, the interoperability needs to appear in different ways, such as in the context production, format, and interpretation. The interoperability requirement is related to the capability of the platform to manage context information in different aspects, such as format, source, length, and representation. Currently, there is no standard context notation format. Although there is a wide range of applicable context information, it is very difficult to make a one size fits all context format. The complete interoperability only will be possible by the combination of many other factors, such as modeling, data dissemination, communication services, among others. The works are categorized in two groups: full interoperability (FuI), and partial interoperability (PaI). The full interoperability tries to address interoperability in different ways (e.g., data format and communication technologies) and between heterogeneous entities. Partial interoperability usually provides interoperability only in a predefined application domain or between entities that may have similar characteristics to define the context.

- **Context Processing (CP)**: Considering the development of the IoT and with the increasing number of devices being connected to the Internet, it will not be realistic for those requesting context information to enter an IP (Internet Protocol) address to a specific device. In this sense, the context sharing platform must have the smart capacity to obtain, produce, and share context information from a highier-level request. This feature is similar to the device discovery service already present in IoT environments [85]. The works are classified by the following context processing techniques: searching (S), filtering (F), and aggregation (A). The symbol (✓) is used to denote that the work employs the context processing feature, but it does not make clear the specific technique.

- **Infrastructure Configuration and Management (ICM)**: In an IoT environment, many different devices can connect to the network. It is essential to the context sharing platforms to provide ways for the new devices/applications to connect with the sharing infrastructure. The infrastructure configuration and management feature must facilitate the connection to the sharing platform to different kinds of devices and applications. Besides, to allow the sharing of information, context sharing platforms must be responsible for managing the connected ones in order to know a possible source and destination for context information. The symbol (✓) is used to denote that the work employs techniques to provide the infrastructure configuration and management feature.

- **Scalability and Real-time Sharing (SRT)**: In IoT environments, with many requests happening at the same time, enormous quantities of data/context information need proper processing with acceptable execution time. Moreover, this also reverberates to massive communication efforts. Taking into account this concept, the context sharing process must minimize the processing and the communication overhead in the sharing platforms. The symbol (✓) is used to denote that scalability and real-time sharing functionality is employed by the analyzed works in some perspective.

- **Availability (Av)**: As the context information can be produced every time that the IoT devices generate new data, the context sharing process may happen anytime in such dynamic IoT environments. Context sharing platforms must be always available for a possible sharing. In this sense, sharing platforms may run the sharing process automatically (Aut), that is, the sharing should occur without the need for setting up the system, and it will happen as soon as new context information is produced and defined as shareable. On the other hand, the sharing process may need to be triggered (Tri), that is, it will require some setup or modification in the system to start the sharing process.

2.4.2 Architectural Components

- **Communication Technologies (C)**: The context sharing platform is an entity of the IoT environment. In this sense, it must be able to communicate with many other entities. This communication may happen to gather specific data, or, in most cases, to share the context information. This topic is not directly related to the context sharing process, but the communication services feature refers to which network technologies the system supports, that is essential for the communication between the entities. The systems may offer local communication (Loc) (e.g., Bluetooth, Wi-Fi, NFC), external communication (Ext) (e.g., 3G, 4G, LTE), or both (LE). Moreover, network layer protocol aspects and messaging paradigms are also crucial for communication technologies in such systems. These kinds of communication technologies help to provide interoperability by their different ways of implementation. Some examples are Web services (e.g., REST, SOAP), socket and WebSockets. These kinds of technologies are well-known device-agnostic standards for communication [101][47]. Thus, the work is considered attending the network layer communication criteria (NeC) if it mentions the use of such kinds of technologies.

- **History (Hi)**: The shared context information may be stored in the context sharing platforms. It can be useful for probabilistic reasoning or to access the last record of some information. Also, the history can help in the real-time processing, when it is updated in storage. Context information may be stored locally or in the cloud. The symbol (✓) is used to denote that the work employs techniques to provide the history feature.

- **Architectural Model (Ar)**: The approach for the architectural perspective of context sharing platforms may vary depending on the application domain characteristics. There is a wide range of applicable perspective of systems architecture. However, considering the IoT environments, some architectures are more suitable than others. There are three main architectures for sharing platforms appropriate for IoT environments: cloud-based (Clo), centralized-edge (Cen), and decentralized-edge (Dec). In the cloud-based, the most processor demanding computation is executed in the cloud. It has a significant dependency on the network. The centralized-edge brings the computation near to the IoT devices, but still has a central point of computation, even to store information. It works based on groups. Finally, in the decentralized-edge, the computation is divided into devices. It may lack in some features, once the device may be limited in some ways, like processing power and storage capability. On the other hand, some systems do not follow a specific architectural perspective and can adapt (Adp) itself depending on the environment.

2.5 CONTEXT-AWARE SECURITY

The context-aware technology brings completely new experience for the applications operators and to users. Traditionally, security requirements are assumed to be relatively static since security decisions do not change with context, nor do they account for changing conditions in the environment [3]. However, the use of context information to provide security decisions is key feature to mitigate some security problems. There are only a few applications, which has security based on context [43][142].

The Context-Aware Security (CAS) is defined by Mostéfaoui and Brézillon [21][97] as: “a set of information collected from the user’s environment and the application environment and that is relevant to the security infrastructure of both the user and the application.” Also, CAS can be defined as a situation where a security solution considers a set of information (context) while making a specific security decision. For example, while detecting an intrusion during communication, security mechanism may adapt to strong authentication method [61].

The context unaware mechanisms can be inadequate for the Internet of Things due to its dynamic and heterogeneous environment. The context information can be used to reconfigure security mechanisms and adjust security parameters. The contextual information can be integrated into various security mechanisms such as authentication, access control, encryption, etc [61]. For example, access from City A can have different access rights then access from City B. They can even sometimes omit authentication because their context is trustworthy by itself (e.g., access from inner company network). Similar to users, also application operators can profit from the context-based authentication. They might define more strict security rules for suspicious users behavior (e.g., Internet access to system confidential resources at night). Using context allows system administrators for more fine-grained security rules, which would be otherwise tangled with multiple rules and make them unsustainable for maintenance [142].

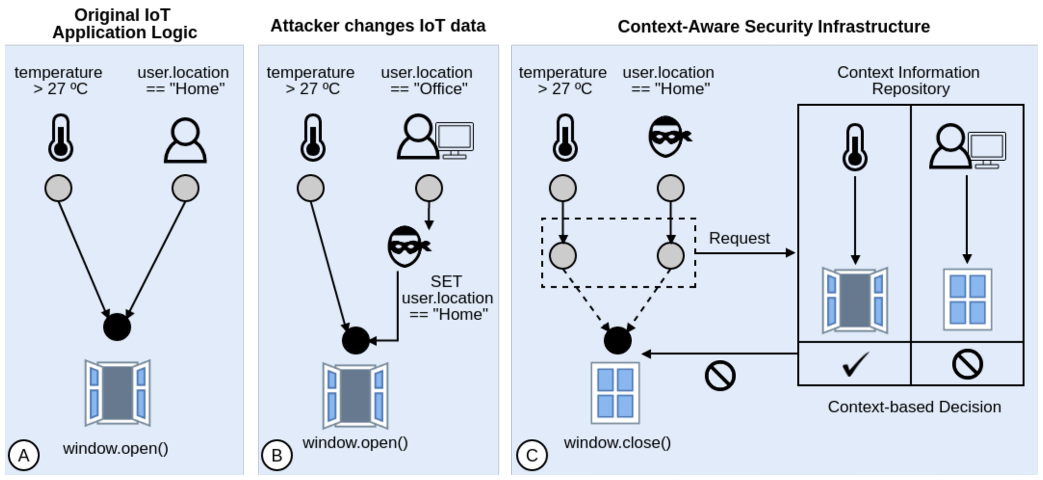

Original IoT Application Logic
Attacker changes IoT data
Context-Aware Security Infrastructure
temperature > 27 ºC
user.location == "Home"
user.location == "Office"
SET user.location == "Home"
Request
Context Information Repository
Context-based Decision
window.open()
window.close()
A
B
C

Figure 2.7 – Overview of Context-Aware Security in IoT environments.

Figure 2.7 shows an example of a possible attack in an IoT environment and how CAS deals with this issue [72]. Letter (A) shows a standard IoT application scenario, which has devices generating data. Let's consider that it may have an application deployed in this scenario that reasons about the device data and do some decision making. For example, *WHEN* the temperature is bigger than 27℃ *AND* the user location is defined as "Home," *THEN* a specific window is open.

In Letter (B), the temperature device still generating the same data, but the user location was changed. In this sense, when the user is at "Office," the pre-defined rule no longer works. The window will only be opened when the user location is "Home." In this sense, an attacker may *SET* the user location to "Home" in order to open the house window (i.e., get access to a determined resource).

The CAS process takes care of the issue in Letter (C). Every time that a crucial (i.e., security) decision has to be made by the application, it may check the information with the Context-Aware Security Infrastructure. The CAS platform must have a repository with the last context information sensed (i.e., near real-time). Also, it must contains the context sensed in a determined situation, for example, every day at this time the user tends to be at "Office," so it is unusual to his location be "Home." In this sense, in this specific example, the window will be closed (or not opened), since this is an unusual activity and the last sensed context informs that the user locations was “Office.”

While the notion of context awareness has been well researched in recent years [1], currently there is a lack of security and privacy-preserving mechanisms that take into account dynamic context conditions for the IoT [34][115]. To the implementation of CAS in IoT environments four main areas can be considered: (i) authentication, (ii) authorization, (iii) access control, and (iv) privacy-preserving. Next items present an overview of each area [3][61][142].

- **Authentication:** Traditional authentication methods require much user interaction in the form of manual log-ins, logouts, and file permissions. These manual interactions violate the vision of non-intrusive ubiquitous computing. Traditional security mechanisms are context-insensitive (i.e., they do not adapt their security policies to a changing context). Reliable authentication is an essential requirement for secure systems. Today, passwords are the most common form of authentication. However, passwords are also a major source of vulnerabilities, as they are often easy to guess, re-used, forgotten, shared with others, and susceptible to social engineering [65][69][64]. Moreover, well-known technologies can be used for authentication, such as face recognition, iris scanner, and biometric technology. Besides these technologies, the use of context information strengthens the authentication process.

- **Authorization and Access Control:** Although different, these two areas are presented together since most approaches try to reach both. Many existing computer networks comply with allow and deny based access control policies. Allow means granting access when the user or device credential matches with pre-stored credentials and deny means blocking access when the user or device credential do not match with pre-stored credentials. This type of system can be considered static in nature because it does not take into consideration other factors such as, contextual information from the user or device environment while making allow and deny decisions. But the IoT has a dynamic environment, where flexible security policies using contextual information can potentially increase the effectiveness of security decisions.

  Two of the oldest principles for securing application resources are Mandatory Access Control (MAC) and Discretionary Access Control (DAC) [119]. Those two principles does not define how the application security should be implemented, but rather define core principles in authorization. In MAC there exists an authority that has the responsibility to grant permissions to access all resources. On the contrary, in DAC, the permission can be granted by anyone with sufficient permission for the resource. However, granting permissions to every user in the system is unpractical for larger amount of users. There are many principles for Authorization and Access Control as role-based access control (RBAC), rule-based access control (RAC), attribute-based access control (ABAC), and many others. Also, some authors merge one or more principles by creating a new one (e.g., Context-Aware Role Based Access Control (CARBAC)) to fits with its necessities [66].

- **Privacy-Preserving:** Since information reflecting users' daily activities (e.g., travel routes, buying habits), it is considered by many users as private it would be no surprising that one of the requirements to ubiquitous applications would be privacy preservation [79]. Location-based service (LBS) provides a user with contents customized by the context information, such as the user location and nearest restaurants/hotels/-clinics, which are retrieved from a spatial database stored remotely in the LBS server. LBS not only serves individual mobile users, but also plays an important role in public safety, transportation, emergency response, and disaster management. With an increasing number of mobile devices featuring built-in Global Positioning System (GPS) technology, LBS has experienced rapid growth in the past few years. Despite the benefits provided by LBS, users may not be willing to provide their current location to the LBS server due to concerns on location privacy [110]. The objective of a privacy-preserving LBS is to protect the privacy of a user's location while maintaining a high level of LBS accuracy. The context information can be used to determine when or not to keep user information private.

2.5.1 Taxonomy of Context-aware Security in IoT

Taking into account widespread published research in the context-awareness area [109][61], and considering particular features of heterogeneous IoT environments, such as processing power, storage capacity, network conditions, and different users/applications, we defined a taxonomy of context-aware security in IoT. The taxonomy presents the main characteristics of context-aware security solutions alongside with the possible deployment variations. It is depicted in Figure 2.8. The taxonomy is divided into three parts: (i) Context Modeling (i.e., how to manage with context), (ii) Key Architectural Components (i.e., architectural characteristics), and (iii) Applicability (i.e., in which way the context-aware security is provided). Next items discuss the taxonomy verticals.

- **Context Source**: Context-aware security solutions need context information to provide security services. It can be acquired from different sources, such as *local domain*, *shared domain*, and *outside domain*. The domain relates to the physical place where the solution is deployed (e.g., healthcare, smart city, industry 4.0). In the *local domain*, the context-aware security solution has access to context information only from its domain. It is the most common way to get context information. Regarding the *shared domain*, it refers to a context that can come from the same domain of the solution but in

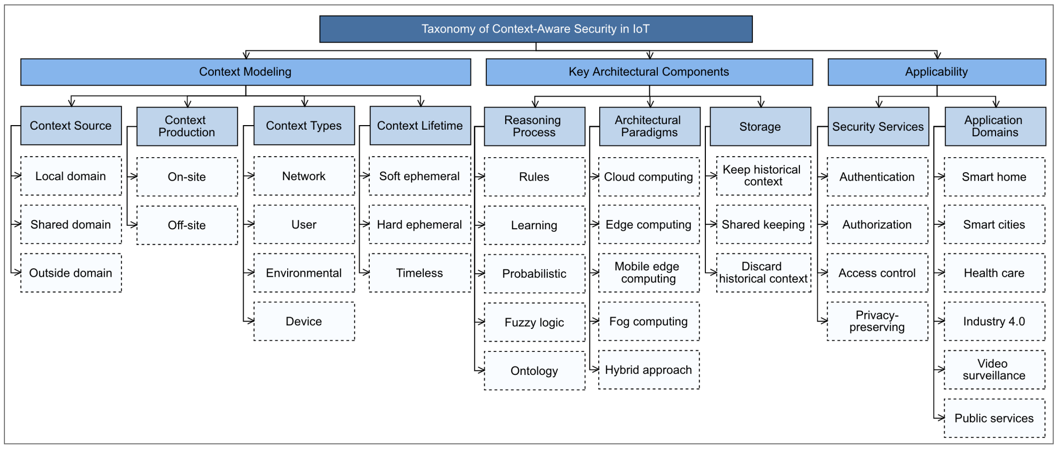


Figure 2.8 – A taxonomy representation of Context-Aware Security in IoT.

another deployment site (e.g., two instances of the same healthcare solution). Finally, the *outside domain* is the most challenging way to acquire context. It refers to a context from a different domain of the deployed one. Most times, the context of a different domain has distinct characteristics, such as size, length, formatting. It demands a processing to achieve interoperability among context information of different domains.

- **Context Production**: The context can be produced in two different ways: *on-site* or *off-site*. The *on-site* context production happens when the solution providing context-aware security is also responsible for the whole process of producing the context information, by acquiring the raw data from the IoT entities and turning them into context information. The *off-site* process happens when a third-party software entity is responsible for the context production. Both Context Production types allow the Context Source to be from *local domain*. However, Context Source from *shared* or *outside domain* is only possible with the off-site Context Production.

- **Context Types**: The Context Type is defined by the characteristics of the source that the context was acquired. The different Context Types of the context information are: *network*, *user*, *environmental*, and *device*. A context-aware security solution can have context information of one or more context types. It depends on how complex is the deployment environment. A *network* Context Type gives information about the status of the network, bandwidth situations, congestion, fault nodes. The *user* Context Type is linked mainly to location, activities, paths, preferences. *Environmental* Context Type represents information about the weather, crowding, time. Finally, the *device* Context Type is linked to physical characteristics, such as temperature, battery life, possible errors.

- **Context Lifetime**: Context can be a time-sensitive information depending on the deployment environment. If context information becomes old, it can lose value, as IoT environments tend to be highly dynamic and the data can change in a minimal amount of time. The Context Lifetime can be classified in: *soft ephemeral*, *hard ephemeral*, and *timeless*. The *soft ephemeral* context information is useful for a specific period, but it is not crucial to be always updating that information. The *hard ephemeral* context information needs substantial updates, once it can vary in each interaction. On the other hand, *timeless* context information represents the ones that do not need a frequent update and may not change along time. The amount of time for a context become useless depends on the deployment environment and must be set by the context-aware security solution.

- **Reasoning Process**: The Reasoning Process is responsible for transforming raw data into context as well as to turn the context information into security services by the context-aware security solutions. Many researchers have surveyed the different techniques that can be used in this processing [109]. The most popular reasoning techniques are *rules*, *learning*, *probabilistic*, *fuzzy logic*, and *ontology*. *Rules* are the most widespread technique. It is simple to use and is one of the most lightweight options, which should be considered for resource-constrained IoT environments. *Rules* are based on IF-THEN-ELSE conditions. The *learning* techniques as Bayesian Networks and Decision tree are also widespread techniques. However, they require a significant amount of data for more accurate reasoning. *Probabilistic* reasoning also needs an extensive data set to produce satisfactory results. However, it reasons numerical values only using past acquired data/context. *Fuzzy logic* allows a more natural representation of the environment, it is also simple to define and easy to extend. Even so, it can be error-prone considering that it is manually defined. The use of *ontologies* for the reasoning process in IoT environments is growing in the last years [109]. It allows more complex reasoning and representations, thus providing more meaningful results. However, the input data should be modeled in a compatible format (e.g., Web Ontology Language (OWL), Resource Description Framework (RDF)) and it tends to have low performance, by requiring more computational effort than usually found in resource-constrained IoT environments.

- **Architectural Paradigms**: The context-aware security solutions can follow different architectural paradigms depending on diverse requirements, such as resources availability, storage space, network conditions, processing power. These architectural paradigms can be categorized in *cloud computing*, *edge computing*, *mobile edge computing*, *fog computing*, and *hybrid approach*. *Cloud computing* is the most centralized approach. It focuses on performing the essential processing tasks at the cloud, and also use it to store information if necessary. Oppositely, both *edge computing* and *mobile edge computing* paradigms focus on processing the more critical tasks at the edge of the network, directly on the data sources devices. These paradigms help in decrease network latency and tend to be more scalable. The difference between edge and mobile edge approaches is mobility since the mobile edge entities can change its location frequently. An example of the *mobile edge computing* is bringing processing capabilities to the edge of a cellular network [63]. The *fog computing* paradigm represents an approach between cloud and edge. It extends cloud computing into an intermediate layer physically close to data sources devices [58]. Finally, some solutions integrate more than one architectural paradigm described, being characterized as *hybrid approaches*.

- **Storage**: The context-aware security solutions may have different ways to deal with the context information regarding storage. They can *keep historical context*, perform a *shared keeping*, or *discard historical context*. The ones that *keep historical context*, store internally all the context information used for the security services provisions. The history can also be used alongside with the context-aware security reasoning for probabilistic processing. They keep historical context locally, which turns the access time to the historical context fast. However, it needs a wide storage space. A different way of storage is the *shared keeping*. These solutions can also have access to previously used context information but in a shared way. The context is stored in another instance, different from the one performing the context-aware security process. The different entity can be a database in another domain, or a cloud storage service. It is necessary to use the network to access the shared keeping context information. They can benefit for possible reasoning using historical context, but one drawback is the network delay to get the information. Finally, some solutions may only use the context information during the execution time and *discard historical context*.

- **Security Services**: Context-aware security solutions can be used to provide different Security Services in the security and privacy area, such as *authentication*, *authorization*, *access control*, and *privacy-preserving*. The solutions use the Reasoning Process to reason about the context and provide a Security Service. A solution can provide one or more Security Services, it will depend on the deployment environment and the Reasoning Process used. In most cases, it is based on the traditional security (e.g., Role-based Access Control (RBAC)), but it suffers some modifications to work accordingly the context. An example of a Security Service is the Context-Aware Role Based Access Control (CARBAC) scheme proposed by Hosseinzadeh et al. [66]. It controls the access of the users to the system following their role, as the traditional RBAC, but it also considers the current context information for granting access. In CARBAC the access is granted depending on dynamic information, as location, time of the day, status, and even the role of a user can change based on the context. The most significant challenge to overcome for a Security Service be provided is to consider the context as input. It only can be achieved by the use of the Reasoning Process to understand the semantic context information.

- **Application Domains**: This subsection shows some examples of possible scenarios that can take benefit of context-aware security solutions. A number of different application domains can use context-aware security solutions. Some examples are *smart home*, *smart cities*, *health care*, *industry 4.0*, *video surveillance*, and *public services*. The major offering of context-aware security solutions in these scenarios is related to the Security Services previously defined. In a smart city, the privacy-preserving can be achieved when the people may share its personal context with a city infrastructure only at a determined context (e.g., out of the home, crowding environments). Context-based access control policies can be defined in an industry 4.0 environment to the employees only access some rooms depending on the context. Health care, public services, and video surveillance domains may contain secret and sensitive information. Authentication and authorization context-based factors may be defined for those scenarios to guarantee that who sees the information is only the persons with the right context (e.g., location and time). Also, considering the dynamism of these scenarios, the authentication policies may change depending on the context. For example, the authentication method may be enforced depending on the user's location.

2.5.2 Key Requirements of Context-aware Security in IoT

The taxonomy presented at the Section 2.5.1 showed the main characteristics of context-aware security solutions in IoT and how their application may vary. Taking this into account, the present section defines the mandatory features for a smooth provision of security services using context information [109][68]. There are different ways to fulfill the requirements detailed below. It can vary depending on the deployment environment and on its context.

Context acquisition: The acquisition comprises three characteristics of the defined taxonomy (see Figure 2.8): Context Source, Context Production and Context Types. It is crucial that the context-aware security solution maintain a source to acquire context. Also, the context should be produced acquiring information from different types (e.g., network, user, environmental, device). The acquisition process can be made in different ways, such as web services, web sockets, publish/subscribe, observer pattern, among others. It is important reiterating that the context-aware security solution should adapt to different ways of acquiring context to fit in heterogeneous IoT environments. For example, some entities may provide context information in different formats, or by different communication methods. Thus, it is important to the context-aware security be aware of the different scenarios. Also, the context-aware security solution should be able to acquire context from various sources depending on the deployment environment. For example, a context-aware security solution deployed in a smart city can acquire context from a traffic light by publish/subscribe and from an ambulance by web service at the same time.

Context processing: This component can also be called reasoning, or security services inference. It is strictly linked to the Reasoning Process, which examples can be seen in Figure 2.8. The context processing is the main component in a context-aware security solution. It is responsible for providing secure decisions using the context information as an input. There is no definition of how many context information can be processed to provide a secure decision, it can vary depending on the reasoning technique and the application domain. For example, some techniques may require more processing power while some application domains may be more resource constrained. The context processing needs to adapt itself depending on the IoT environment that it is inserted. Thus, the design of this component should consider the resource restrictions imposed by the IoT environment in order to have a smooth and functional context-aware security solution.

Interoperability: The great amount of different available IoT devices leads us to have a great heterogeneity as well. Context information can vary in many ways, such as format, data type, size, representation. Thus, context-aware security solutions for IoT should be interoperable by ensuring that different context information is compatible to be used as input for the security services provision. Also, there is no standard for context information representation [109], which makes this requirement more crucial and also hard to implement.

Privacy: As most of IoT systems, context-aware security solutions also deal with sensitive information. For example, if a context-aware security solution is placed at a hospital, it will deal with sensitive health context of the patient (e.g., health status). Thus, it is essential to keep all the context information protected from possible attacks trying to steel or modify them. These attacks can happen when the context information is stored or even when it is on the network. The communication channels must be protected with secure protocols to ensure data integrity and privacy. Also, authentication and access control methods should be used to protect the stored information, as well as cryptography/anonymization.

Reliability: As the context-aware security solutions provide security and privacy for IoT environments and can be used in many application domains, it is a requirement to be reliable. Reliability is a critical requirement since the context-aware security solutions may deal with sensitive data, such as healthcare and banking information. It is essential to ensure the reliability to foster the users/applications usage and also to improve the security level of the entire solution.

3. RELATED WORK

In this chapter, it is analyzed two types of related work. First, Section 3.1 presents a comparison among context sharing platforms. The challenges and open issues are shown in order to clarify the gap in the area. Second, Section 3.2 presents an overview of platforms with context-aware security feature.

3.1 CONTEXT SHARING PLATFORMS

There are already available systems/platforms that support data processing, data enrichment, and data sharing. Moreover, some systems/platforms perform the sharing of more complex information than raw data, thus having the context sharing feature (see Section 2.3). In this Section, it is presented the analysis of different works focused on sharing their contexts in different ways. The analyzed works were selected by its capability to provide or facilitate the context sharing in some way. Moreover, the selection was based on the works that could address an IoT environment application. Some works talk directly about IoT, while others talk about pervasive/ubiquitous computing application scenarios.

The systems are analyzed based on the requirements presented previously (see Section 2.4), which are essential to provide a Context Sharing Architecture. Table 3.1 summarizes all the aforementioned sharing building blocks and its abbreviations, helping in understanding the Table 3.2, which presents all the analyzed works. The proposed work appears in Table 3.2 as "**This work**". The architectural and implementation details of "**This work**" can be found in Chapters 4 and 5.

To provide a fair comparison between the works, they were categorized and grouped in two categories: (i) Model, and (ii) Middleware. The Models are lightweight approaches that has the goal of facilitate and assist in the context information sharing process (e.g., ontologies, Bluetooth-based systems, models for formalizing context information). The Models can act in classifying context information to model it, making easy the context interoperability. Also, some Models may serve as a broker, in which context can be posted and queried. Many times, Models may react to a determined event (e.g., generated context information) or situation (e.g., proximity to an entity) triggering the sharing process. On the other hand, a Middleware is a platform that provides diverse kinds of services, and has different methods to manage data [137]. The Middleware matches with the definition of a PaaS (Platform as a Service) [11], offering a platform that users can deploy different services, including the context sharing functionality.

The Models can be considered a more straightforward category for context sharing, offering only that feature or facilitating it in some way. The Middleware are more robust

Table 3.1 – Summary of Context Sharing Building Blocks.

Context Sharing Building Blocks	Possibilities
Category	Model (Mo), Middleware (Mid)
Modeling (M)	Key-value modeling (Key), markup schemes (Mrk), text-based modeling (Tex), graphical modeling (Gra), object-oriented modeling (Oob), logic-based modeling (Lob), ontology-based modeling (Onb). (✓) it is employed, but no specific technique is detailed
Reasoning (R)	Supervised learning (Sul), unsupervised learning (Unl), reinforcement learning (Rel), rules (Rul), fuzzy logic (Fuz), ontology-based (Onb), probabilistic reasoning (Pro). (✓) it is employed, but no specific technique is detailed
Data dissemination (D)	Static (Sta), dynamic (Dyn)
Privacy (P)	(✓) it employs privacy techniques
Interoperability (I)	Full interoperability (Ful), partial interoperability (Pal)
Context processing (CP)	Searching (S), filtering (F), and aggregation (A). (✓) it is employed, but no specific technique is detailed
Infrastructure configuration and management (ICM)	(✓) it employs infrastructure configuration and management techniques
Scalability and real-time sharing (SRT)	(✓) it employs scalability and real-time sharing techniques
Availability (Av)	Automatically (Aut), Triggered (Tri)
Communication technologies (C)	Local communication (Loc), external communication (Ext), local and external (LE), network layer communication (NeC)
History (Hi)	(✓) it employs history techniques
Architectural model (Ar)	Cloud-based (Clo), centralized-edge (Cen), decentralized-edge (Dec), adapt (Adp)

platforms that usually provide context sharing as an option and different features (e.g., data storage, device interconnection) as main goal. Both categories, Model and Middleware, are suitable for deployment in IoT environments. Even though, by its characteristics, Models are more suitable for lightweight environments and Middleware for those with more resource (e.g., processing power, memory). However, it is a common characteristic of both Models and Middleware to provide a stream processing of data (i.e., context information). The stream happens because the context is always changing (especially in IoT), and it is essential to share the different context when the events occur to keep the environment updated. Some approaches may need a previous connection and others stream context automatically. Moreover, some works also provide a batch processing.

Different works were selected to compare, differing in scale, from small-scale to large-scale projects. Moreover, it this Section analysis both well established works as well as new projects available in the past few years. The works are compared in different categories (i.e. Model, and Middleware) which have different ways to provide context sharing. The analyzed works are detailed in Section 3.1.1 and Section 3.1.2. Section 3.1.3 discusses how the works provide context sharing. Moreover, it is discussed the International Efforts in context sharing in Section 3.1.4. By covering the vast heterogeneity of context sharing works, it is provided a large vision of the area.

Table 3.2 – Evaluation of Surveyed Works.

Sharing Platforms	Ref	Year	Category	Properties									Architectural Components		
				(M)	(R)	(D)	(P)	(I)	(CP)	(ICM)	(SRT)	(Av)	(C)	(Hi)	(Ar)
CONON	[143]	2004	Mo	Onb	Rul, Onb	Dyn	—	Ful	✓	✓	—	Aut	—	—	Adp
ACC	[117]	2004	Mo	Oob, Lob	Onb	Sta	✓	Pal	✓	✓	—	Tri	—	—	Adp
DJess	[26]	2005	Mo	Tex	Rul	Sta	✓	Pal	—	✓	✓	Tri	Ext, NeC	✓	Adp
CoSM	[146]	2009	Mo	Onb	Onb	Dyn	—	Ful	—	✓	—	Aut	—	—	Clo
M3	[60]	2014	Mo	Onb	Rul, Onb	Dyn	—	Pal	S	✓	—	Tri	—	—	Adp
CS-Sharing	[144]	2016	Mo	Key	Rul	Dyn	—	Pal	A	✓	✓	Aut	Loc, NeC	✓	Dec
Bluewave	[38]	2016	Mo	Tex	Rul	Sta	✓	Pal	—	✓	✓	Aut	LE, NeC	✓	Cen
PSW	[118]	2017	Mo	Onb	Rul, Onb	Dyn	—	Ful	—	✓	✓	Aut	LE, NeC	—	Adp
LiO-IoT	[114]	2018	Mo	Onb	Onb	Dyn	—	Ful	—	✓	✓	Tri	—	—	Adp
SCS	[126]	2019	Mo	—	Rel	Dyn	—	Ful	—	✓	✓	Aut	NeC	—	Dec
SE-TSDB	[147]	2019	Mo	Onb	Rul, Onb	Dyn	✓	Ful	F,A	✓	—	Tri	—	✓	Adp
FRASCS	[78]	2008	Mid	Key, Mrk	Rul	Dyn	—	Ful	A	✓	—	Aut	—	✓	Clo
SharedLife	[77]	2009	Mid	Onb	Rul, Onb	Dyn	✓	Ful	✓	✓	—	Aut	—	✓	Adp
ConCon	[87]	2014	Mid	Key	Onb	Sta	—	Pal	F	✓	✓	Tri	LE, NeC	—	Cen
Grapevine	[31]	2015	Mid	Key, Tex	Pro	Dyn	—	Ful	F	✓	✓	Aut	—	—	Cen
HEAL	[88]	2015	Mid	Key	Pro	Dyn	—	Pal	A	✓	—	Tri	Ext, NeC	✓	Clo
Magpie	[83]	2015	Mid	✓	✓	Dyn	✓	Ful	✓	✓	—	Tri	LE, NeC	—	Dec
OIoT	[106]	2015	Mid	✓	Rul	Dyn	—	Ful	A	✓	—	Aut	LE, NeC	—	Adp
RCOS	[45]	2016	Mid	Onb	Onb	Dyn	—	Ful	✓	✓	✓	Aut	Ext, NeC	✓	Cen
Chitchat	[32]	2016	Mid	Key, Tex	Pro	Dyn	—	Ful	—	✓	✓	Aut	LE, NeC	✓	Adp
C2IoT	[52]	2017	Mid	✓	✓	Dyn	—	Ful	F,A	✓	—	Aut	NeC	✓	Clo
BigClue	[70]	2018	Mid	✓	Pro	Dyn	—	Ful	—	✓	✓	Aut	Ext, NeC	✓	Cen
CoaaS	[63]	2018	Mid	✓	Rul, Pro	Dyn	✓	Ful	A	✓	✓	Tri	NeC	✓	Clo
SCENTS	[82]	2019	Mid	✓	Rul	Dyn	—	Ful	—	✓	—	Aut	LE, NeC	✓	Cen
This work	—	2020	Mid	Mrk, Onb	Rul, Onb	Dyn	✓	Ful	S,F,A	✓	✓	Aut	LE, NeC	✓	Dec

In Table 3.2, the dash (—) symbol is used when the work does not provide the specific feature, or it is not mentioned in the available publications. Next, the definitions of analyzed platforms are presented.

3.1.1 Context Sharing Models

- **CONON:** It is an OWL (Web Ontology Language) ontology to model context information in ubiquitous environments. Besides the modeling, as being an ontology, CONON also facilitates the logic-based inferencing process [143]. CONON provides a flexible architecture, once it has a general ontology for modeling broader context concepts and also enables the coupling with other ontologies in a hierarchical manner. The coupled ontologies could be from any specific domain. To CONON, the context information of different domains shares common definitions and concepts. Thus, it uses a generic ontology to model these general concepts, while domain-specific ontologies deal with fine-grain context information.

- **ACC:** Agent Coordination Context (ACC) works to model the application environment context and the interaction among agents and the environment in Multi-Agent Systems (MASs) [117]. ACC provides tools to organize the access to the shared information (i.e., context). It organizes the shared information into "spaces" and regulates the access control in a role-based policy. Thus, ACC has two main functions: (i) it works to model the application environment characteristics to enable context sharing, and consecutively (ii) by creating sharing "spaces" for interaction considering the environment characteristics. In this sense, the ACC can be suitably understood as an infrastructural abstraction.

- **DJess:** It is a Java package that works in a distributed way and provides an infrastructure for context sharing between entities using it [26]. Only middleware that was implemented using Jess [55] for the inferencing/reasoning process can run DJess. Different nodes using DJess can communicate to have a common understanding of the context. DJess creates an abstraction of a single view of the application environment for the distributed systems running it. Thus, even if the systems were placed in different sites, they will share context as if it were in the same local domain.

- **CoSM:** The Context Sharing Message Broker (CoSM) can model the context in a standard way to enable the context sharing process [146]. CoSM helps in facilitating the common understanding of context between different applications even in dynamic environments. Applications can agree in defining a context model to share context information. CoSM's context model intermediates the context sharing processing by providing a common interface for a mutual understanding of different context information. The applications communicate with CoSM to send context information, then CoSM manages it and delivers the context to interested entities/applications based on the defined context model. CoSM acts as a plug-in to the application, making it an independent tool and not modifying the applications that use it.

- **M3:** The Machine-to-Machine Measurement (M3) is an approach that includes an ontology, a hub, and semantic domain rules, that can combine, and reason about IoT devices data that follow the M2M (Machine-to-Machine) standard [60]. The authors have proposed a semantic-based M2M architecture that is used as basis for M3 [59]. The M3 ontology main goal is to classify and unify the heterogeneous data sensed by devices running at the M2M standard. It is an extension of the Semantic Sensor Network ontology (SSN)[1], thus making easy the reuse of common concepts as appears

[1]https://www.w3.org/TR/vocab-ssn/

on the M2M standard. The authors claim that the M3 ontology is able to describe more than 30 sensor types (e.g., thermometer, accelerometer) from different domains (e.g., agriculture, health).

- **CS-Sharing:** It is a compressive sensing (CS) based scheme technique to provide context sharing in a decentralized way for vehicular networks. CS techniques enable the search for context information with basic queries [144]. The vehicles monitor the road conditions to acquire context and then share it using CS-Sharing. The vehicles store context locally. To avoid possible network overheads, CS-Sharing offers aggregation operations in the stored vehicle context before the sharing process. Thus, each vehicle can get context information about the road condition from a focused aggregated message that is broadcast by vehicles.

- **Bluewave:** It is defined as a Bluetooth-based technique that makes possible nearby mobile devices to share their context [38]. In Bluewave, the context information first needs to be uploaded to a server for enabling the context sharing, characterizing it as a centralized approach for storing but an edge in processing. For an accurate context sharing process, it also needs to set a sharing URL for each device and broadcast it, along with a temporary credential, to the devices in a 30 feet radius. Bluewave has two main components: (i) client, and (ii) context broker. The client component is embedded into mobile devices and has the functions of upload the context information to a server, and discover new devices by proximity. The context broker component runs in a web server and has the responsibilities to store context and to share it with the authorized clients (i.e., nearby mobile devices).

- **PSW:** The Physical Semantic Web (PSW) is a framework that makes use of Semantic Web technologies to knowledge discovery and sharing in IoT scenarios [118]. PSW has four components: (i) machine-understandable standard languages, (ii) objects exposing semantic annotations, (iii) knowledge discovery and sharing, and (iv) semantic matchmaking. The machine-understandable standard languages main goal is to provide a common language for a uniform communication. It uses an ontology for this goal. The objects exposing semantic annotations component is responsible for exposing the context information for sharing, and also to update this information when needed. The knowledge discovery and sharing components are responsible for discovering the neighbor devices with corresponding semantic annotations as the request. Finally, the semantic matchmaking component main goal is to make a rigorous semantic matching to rank the resources with the request by relevance.

- **LiO-IoT:** The Light-weight Ontology (LiO-IoT) tackles a challenge in IoT ontologies spectrum by considering sensors, actuators, and RFID as IoT concepts [114]. It uses concepts from both SSN and IoT-Lite[2] ontologies to help in the provision of semantic

[2]https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/

interoperability. The authors have evaluated the LiO-IoT ontology through experiments to verify the round trip time of a query. They have compared LiO-IoT with both IoT-Lite and SSN. The results have shown that it has a similar response time when compared with IoT-Lite and a better performance when compared with SSN. However, it is valid to mention that SSN is a massive ontology that covers different IoT sensors [33].

- **SCS:** The Smart Context Sharing (SCS) is an algorithm designed for facilitate context sharing among Fog nodes [126]. It focuses on sharing context information of different Fog nodes deployed on different IoT domains (e.g., smart agriculture, smart health, smart traffic). It considers a scenario of connected Fog nodes, in which one of the nodes may broadcast a message looking for a specific context. The Fog node with the wanted context information will respond to the broadcast message. SCS also has a load balancing algorithm that uses reinforcement learning techniques to predict the suitable node to migrate context information when sharing.

- **SE-TSDB:** Semantic-Enhanced Time Series Databases (SE-TSDB) is a tool suite that helps in the data management for IoT [147]. It can work either on the Cloud, Fog, or Edge architectural approach. The authors have developed the DS-Ontology as the main part of the SE-TSDB tool. The DS-Ontology works as a semantic model for the specification of data streams from IoT devices. In light of this, the SE-TSDB is the DS-Ontology applied to Time Series Databases (TSDB), as the traditional TSDBs process data streams, but they do not offer sufficient semantic processing as needed in heterogeneous IoT environments. On SE-TSDB, the similarity matching, and reasoning processing happens with the DS-Ontology alongside with pre-defined domain-specific rules.

3.1.2 Context Sharing Middleware

- **FRASCS:** The Framework Supporting Context Sharing (FRASCS) [78] enables entities to store and/or receive context information. The entities communicate directly with the FRASCS infrastructure. FRASCS has a Context Pool Manager (CPM) model responsible for receiving sensors connection and their raw data. It can deal with both physical (e.g., physical devices, hardware) and virtual (e.g., events from a software system) sensors. FRASCS also has the functionality of claim for context information directly from the context-aware applications when an entity requires it.

- **SharedLife:** It is defined as a framework to share information (i.e., context) of the users between different applications and/or other users [77]. SharedLife acquires the context from various sources related to a user and stores it in a set of knowledge about the specific user. SharedLife also allows the searching for context information about a particular user, taking care of the privacy defined for each user/information. SharedLife shares the data itself and also a meta-information regarding the data, such as information about potential partners for future sharing. SharedLife works in an event-based manner to describe the sharing interactions and the relationships between entities and the environment. The events can be stored to serve as a possible recommendation for future context sharing by the entities connected to SharedLife.

- **ConCon:** It is a middleware system that offers the context-aware feature in its architecture. ConCon [87] works based on the publish/subscribe pattern, enabling different entities to subscribe for specific context information. The entities connected to ConCon can act by producing and/or consuming the data (i.e., context). It can be embedded in everyday devices, such as smartphones. ConCon defines structures of similarity related to the context. Thus, the context information is matched semantically, avoiding possible data duplication. ConCon has two policies to manage context information: (i) time-sensitive, and (ii) quality measure. In the time-sensitive policy (i), the context to be shared receives a prefixed lifetime. Thus, the old context may lose its relevance. In the quality measure policy (ii), the context information is used to improve the quality of entities interactions, one time that a high-quality provider produces more relevant context (i.e., that is widely shared).

- **Grapevine:** It is a framework that works for context sharing in specific networks (i.e., connected peer devices) [31]. Grapevine is designed to work with pervasive devices. It forms groups of devices to perform context sharing. The situation of the entities (i.e., context) is the primary factor for the creation of the sharing groups. Grapevine uses conditions that define whether the entity context sharing occurs and how widely (i.e., for which groups) the context will be shared. Grapevine works to reduce the communication overhead in the context sharing process. As it works with pervasive devices that may have resource-constrained restrictions, a lightweight data structure models the context information.

- **HEAL:** The Healthcare Event Aggregation Lab (HEAL) is a middleware platform focused in smart healthcare environments [88]. It works in a cloud-based approach and allows different entities to connect to it by using REST web service and SPARQL endpoints. The sensors can connect to HEAL to provide context information. On the other side, smart healthcare applications will take benefit of the sensors data. HEAL detects similarity between sensors data, thus providing a filtered data to the applications. The context sharing occurs when HEAL acts as providing interoperability different platforms. It enables the provision of services based on context to heterogeneous applications as third-party systems and developers tools.

- **Magpie:** It is an approach in which context sharing is provided by opportunistically connections between entities [83]. Magpie works in pervasive computing environments, sharing context information of mobile and heterogeneous devices. Magpie connects devices through an opportunistic mobile network. It has policies to concern about the privacy-preserving in devices connections. Magpie allows the entities connected to it for sharing several types of context with different applications/users. The event of sharing context can also generate a log for future trust agreement between the entities. Magpie provides tools to enable an "useful sharing", that avoids the sharing of all the context information produced by the entity, making that sharing process to occur with only the useful context information. Both Magpie and the entities that send or receive context information can define a context as useful.

- **OIoT:** Opportunistic IoT Platform (OIoT) is a software infrastructure which helps developers in providing data (i.e., context) sharing by the creation of opportunistic IoT devices groups [106]. OIoT adopts a form of mobile ad hoc network concept defined as Opportunistic Mobile Social Networks. It exploits the characteristics of human social interactions (e.g., daily activities, mobility patterns, and interests) to route messages and to share data. In such networks, the communication between mobile devices happens by dynamic (i.e., on-the-fly) social networks.

- **RCOS:** Real Time Context Sharing (RCOS) is a publish/subscribe system that provides context-awareness features [45]. RCOS provides subscribe services to the entities connected to it. RCOS makes possible the creation of a subscription about a specific context. Thus, RCOS notifies the subscribed entities when it has new information posted. The devices can connect to RCOS through a REST interface to send their context. RCOS has functions to enable a contextual semantic matching based on the context information generated by the devices. One example of the semantic match is to relate two contexts by proximity based on the GPS coordinates. RCOS also acts like a plugin, being connected to preexisting publish/subscribe systems. Thus, the preexisting system can take benefit of the context-aware features and maintain a context history.

- **Chitchat:** It is considered a pool of context information. Chitchat takes advantage of the devices communication by the network and inferencing capabilities to provide different views and access to the context information [32]. The entities (e.g., devices, applications) can connect to Chitchat through an Application Programming Interface (API) to specify its filters (i.e., when to receive context). Chitchat introduces probabilistic data structures based on a Bloomier filter [29]. Chitchat introduces two Bloomier filter based structures to further reduce a context information size and add the ability to update the structure on-the-fly. The first one claims to reduce the size of the structure without impacting the false positive rate. The second one claims to guarantee a zero false positive rate under certain conditions and adds the ability to update context values in an already constructed context information.

- **C2IoT:** It is a cloud-based framework for providing context-aware services on Internet of Things environments. The main application scenario of C2IoT are smart cities [52]. It provides a three layer infrastructure for such kind of context-aware services provision: SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service). The authors claim that with those three layers, it is possible to reach the necessary flexibility and effectiveness that smart city scenarios require. Thus, dealing with the common challenges of those environments, including context information interoperability. On C2IoT, each layer has a specific function: SaaS - Result Visualization, PaaS - Data Management, and IaaS - Data Collection.

- **BigClue:** BigClue is a data processing platform that integrates existing frameworks on its architecture with the primary goal of providing a cross-domain data layer for IoT environments [70]. It has five main layers on its architecture: Processing, Messaging, Storage, Service registry, and Visualization. For the implementation of those layers, BigClue uses the following techniques: Apache Spark for the Processing layer, Apache Kafka for Messaging, Apache Hbase for Storage, and HashiCorp Consul for Service registry. They also use Hadoop Distributed File System for the underlying file system. Finally, BigClue has RESTful exposed APIs for communication, thus helping on the interoperability.

- **CoaaS:** Context-as-a-Service (CoaaS) is a platform able to exchange context information in IoT environments [63]. It shares context information from different context providers to context consumers. However, it does not have efforts in providing the context in a semantically interoperable way. It has four main components: Security and Communication Manager, Context Query Engine, Context Storage Management System, and Context Reasoning Engine. The Security and Communication Manager handles the incoming context and distributes it to the destination in a private way. The Context Query Engine component is responsible to manage the queries for context. The Context Storage Management System maintains a cache with past context information to provide a better response time. Finally, the Context Reasoning Engine component infers new information (i.e., context) from raw data.

- **SCENTS:** Sensing Collaboratively in Everyday Networks (SCENTS) is a framework that supports context sharing by proximity for heterogeneous IoT devices [82]. One of the SCENTS' motivation is the premise that nearby devices tend to have similar values for context information, thus being interested in the nearby context by its geolocation. On an architecture view, SCENTS framework sits between the hardware layer and the application layer. It has two main components: Neighborhood Agent and the Collaboration Agent. The main function of the Neighborhood Agent is to continuously detects whenever a node (i.e., neighborhood device) is arriving or leaving the network. The

Collaboration Agent manages the queries looking for context, process it, and delivers the right data (i.e., context information) to the applications.

3.1.3 Summary of Sharing Platforms

All the analyzed platforms have a common goal, which is to share context information between entities. However, each platform has its own peculiarities. They may differ in various aspects and even provide shared context information as well. Table 3.2 sums up all the analyzed platforms and shows the differences between them regarding the context sharing building blocks defined in Section 2.4. The next paragraphs show details of how the analyzed platforms address the building blocks described previously. For each feature, the description is focused on platforms that discuss it in detail in the literature.

As presented in Table 3.2, the ontology-based (Onb) appears as the most used technique for the *modeling (M)* feature. It is considered a trend on the IoT to use ontologies. It can increase the interoperability feature of the platform [109]. Both CONON and RCOS use various ontologies for the semantically represent the context information. The use of different ontologies makes easier the integration with other semantic applications, as well with web applications. M3 and LiO-IoT extract some definitions from traditional ontologies, as SSN, on its approaches, helping on the interoperability with already deployed devices/entities. However, in some application scenarios, context systems may need a more complex approach. In this sense, FRASCS stands out by combining two techniques. FRASCS uses the key-value technique to model the information detected by the sensors (i.e., raw data), which can be considered low-level context information. On the other hand, it uses the markup scheme model (Mrk) technique to model the information from the context-aware applications connected to it. The markup scheme model technique is used in this case as it offers a more flexible structure than key-value pairs. Another technique used by many platforms is the key-value modeling (Key). It is a technique of simple implementation and gives a unique key for each context information. Key-value is also a lightweight approach for context information modeling.

As in the modeling of context information, the ontologies appears as one of the most important techniques for the *reasoning (R)* feature. The reasoning by ontologies (Onb) is facilitated when the platform also models context by ontologies. One example of work that uses ontologies for both modeling and reasoning is CoSM. It makes possible that the reasoning agents can access the modeled context to understand the context of the environment. On the other hand, ontologies may be considered heavy for some application scenarios since IoT environments could be restricted in processing capabilities. In this sense, SharedLife proposes a hybrid approach by using ontologies together with rules (Rul), which is a lightweight technique. SharedLife allows the definition of rules for automatic sharing of predefined context. It also uses ontologies to the creation of a unified user profile representation, making possible that different user profiles to being classified by similar characteristics. DJess uses a Java-based rule engine in the reasoning process. It makes possible the creation of declarative rules for reasoning in Java-based applications. Rules are the most popular reasoning technique of the analyzed platforms because of their suitability for resource-constrained environments. In some platforms, it may appear as Event-Condition-Action (ECA) rules.

The *data dissemination (D)* process is fundamental for the delivery of context information to whom needs it. The majority of the analyzed platforms makes the data dissemination in a dynamic (Dyn) way. It is the most suitable way for sharing context information since it enables sharing accordingly to the characteristics of the environment. For example, an offline or not apt entity will not receive context. CS-Sharing, OIoT, and SCENTS use the concept of *Opportunistic Meeting* to share context information. It defines that entities can share context information when they are physically near each other, in a dynamic way.

Just a few of the analyzed platforms provide the *Privacy (P)* feature. Although crucial in IoT environments, most of the platforms do not mention privacy efforts on their architectures. However, ACC provides access control to environment resources while DJess provides privacy-preserving options in the registering process of the entities connected to it. Bluewave concerns about the privacy at the communication protocol level, which can reach a certain level of confidentiality, and Magpie provides privacy-preserving options to devices related to the provided context information so that the context sharing does not breach user privacy. In SharedLife, the context information repository is separated into different levels with access control. SE-TSDB makes use of the Time Series Databases (TSDB) that can include policies to secure the stored information.

As can be seen in Table 3.2, most of the analyzed platforms with context sharing feature address *interoperability (I)* in a full (Ful) way. It is an expected result, as providing interoperability is crucial for a context sharing process involving different entities. CONON and PSW are examples of platforms that use ontologies for a common understanding of a set of concepts regarding context information by the entities interacting through them. The use of ontologies is one of the most effective ways to reach full interoperability because it is developed to work with different situations (i.e., inputs). ConCon uses the WordNet [92] tool, that is very useful in achieving interoperability. WordNet is an extensive lexical database of English. It groups a vast set of words, that include verbs, nouns, adjectives, and adverbs, into logical synonyms. With the use of WordNet, it is possible to discover a correlation between two (or more) different words. For instance, with the input of the words *cold*, *icy*, and *freezing*, WordNet can infer that they may converge in the meaning of *low temperature*. It is very useful in heterogeneous IoT environments, which have many devices with different specifications. The partial (Pal) interoperability addressed by some platforms means that the interoperability is present only on a specific set of entities or in a limited local way. They fail to reach the desired interoperability for context sharing in IoT environments.

Context processing (CP) is addressed by most of the analyzed platforms. ACC executes operations (e.g., modification) in sets of context information. CS-Sharing, HEAL, OIoT, FRASCS, and CoaaS use aggregation (A) techniques. Also, one of CS-Sharing key issues is to aggregate messages to reduce the network overhead and message size in resource-constrained vehicle networks. The filtering (F) technique is addressed by Grapevine, and ConCon. Grapevine filters information based on his contextual social properties (i.e., users, their characteristics, preferences, and the correlation with the environment) and only display the context information that matches with the filter. M3 is the only analyzed platform that provides the searching (S) feature. It uses Sindice[3] as a search engine. SE-TSDB and C2IoT stands out by providing both filtering and aggregation context processing features. C2IoT provides those functions at the Data Collection layer, near to the devices.

Infrastructure configuration and management (ICM) is the most addressed feature by the analyzed platforms. This is expected, once the platform must accept the connection of entities to share context information. ACC proposes an approach in which an agent tries to join in the organization and the request can be refused or accepted. Chitchat is focused on providing the entry of devices to the device-to-device (D2D) network to share context. CONON provides a model that a specific ontology can be registered to the main ontology. Bluewave, CS-Sharing, and Grapevine use proximity to form temporary groups.

Scalability and real-time sharing (SRT) are not only linked to the performance of platforms and the time taken to do tasks. This issue is also related to what platforms can make to reduce the amount/size of context information exchanged by the network and provide more optimized communication. Chitchat fulfills this issue by creating a compressed representation of context information to enable a lightweight context sharing. It takes into account size and energy efficiency. DJess makes the inference process faster allowing better performance regarding both throughput and response time. Grapevine focuses on to be extended to real world scenarios where bandwidth and energy limit connectivity. It also minimizes the transmission overhead. Bluewave uses a lightweight communication protocol. Both CS-Sharing and Bluewave use short-range communication protocols for context sharing, avoiding network delay. BigClue adopts standards instead of customized solutions to help in providing scalability.

Regarding *availability (Av)*, most of the analyzed platforms enable context sharing process automatically (Aut), which is the most suitable technique for a dynamic environment as IoT. FRASCS automatically acquires and delivers the devices/applications context to the participant entities. In CS-Sharing, when entities meet each other, the context sharing process happens automatically. This is the same vision adopted by the opportunistically processing of OIoT and SCENTS.

[3]https://www.w3.org/2001/sw/wiki/Sindice

Communication technologies (C) implies on which network technology the sharing platform supports. Some platforms do not present this information because in some cases, the platform is embedded in an entity, thus not caring about the communication services. However, most of the analyzed platforms have the capabilities of sharing context information by local (Loc) and external (Ext) networks. All the analyzed platforms that detail the communication technologies make use of the network layer communication (NeC). It helps on the interoperability by providing a standard communication channel. Bluewave, and ConCon have an architecture associated to smartphones, thus they are capable of sharing by 3G/4G, WiFi, and Bluetooth.

The *history (Hi)* feature is essential to retrieve older context information. It also can be used for probabilistic inference, as on Grapevine, HEAL, Chitchat, BigClue, and CoaaS. Bluewave has a module named *Context Repository* to store all set of context information alongside with *Context Brokers*. Both modules combined act as a database for context information. They play the role of a trusted entity in which the devices can share context information independently and without having relation to external objects. RCOS enables a query request for all context information of an entity. HEAL has cloud-based storage for context information, events, and valuable data to predict future interactions.

The context sharing process occurs regardless of a specific *architectural model (Ar)*. Most of analyzed platforms may adapt (Adp) itself according to the environment. In this case, they do not follow a specific architecture and in most cases are embedded in other entities. Some platforms, like Bluewave and RCOS, store data in a cloud-based server and process context information at the edge of the network, thus being a centralized-edge (Cen). On the other hand, some platforms, as CoSM and HEAL, are developed only in a cloud-based (Clo) approach, in which all the processing occurs in the cloud. HEAL works with low processing power devices, so the most resource demanding processing needs to be made in the cloud. Finally, there are decentralized-edge (Dec) platforms, as Magpie and CS-Sharing, that does not have a central point-of-control. By working with vehicle networks, CS-Sharing distributes the processing by the decentralized nodes.

3.1.4 International Efforts

Interoperability in IoT environments is one of the focus of many research efforts programs. These programs can be composed by different countries making a consortium or even a large enterprise group looking for standardization. Some examples are EU FP7[4], Horizon 2020[5], ETSI[6], FIWARE[7]. These efforts may differ from the ones presented in Table

[4]https://ec.europa.eu/research/fp7/index_en.cfm
[5]https://ec.europa.eu/programmes/horizon2020/en/
[6]https://www.etsi.org/
[7]https://www.fiware.org/

3.2 in some aspects. The international efforts usually have a lot of different solutions and outcomes, as they try to meet different IoT challenges. In this sense, it is difficult to compare with platforms developed specifically for context sharing. Even so, as they also care about data/context/semantics interoperability, it is interesting to look deeper into that solutions. Table 3.3 shows an overview of the analyzed international efforts. Next paragraphs show their aspects that address context sharing in some way.

Table 3.3 – International Efforts Regarding Context Sharing.

Name	Category	Application Area	Main Goal
EU FP7	Organization	Different technological areas	To support the research in Europe. It has three main projects regarding semantic interoperability: IoT-A, COIN, and IDIRA
IoT-A	Project	Internet of Things	To provide data interoperability by the definition of a reference architecture
COIN	Project	Industry	To provide semantic interoperability by the definition of web interfaces
IDIRA	Project	Crisis management	To facilitate coordination of large-scale disaster situations by improved interoperability
Horizon 2020	Organization	Science and innovation	To support the research in Europe. It has different project related to data interoperability
FIESTA-IoT	Project	Internet of Things	To provide data interoperability across different Internet of Things domains
INTER-IoT	Project	Internet of Things	To provide interoperability for Internet of Things environments by a multi-layered approach
INTER-Health	Project	Smart Healthcare	To share contextual information in a pub/sub manner on smart healthcare domain
Wise-IoT	Project	Internet of Things	To provide interoperability concerning context information
SEMIoTICS	Project	Internet of Things	To provide secure semantic interoperability
BIG IoT	Project	Internet of Things	To provide a unified Web API for IoT platforms. It also provides a way for monetizing shared data
symbIoTe	Project	Internet of Things	To provide an abstract layer for IoT platforms through a Web API
FIWARE	Organization	Internet of Things	To provide a development platform for Internet of Things
Orion	Project	Internet of Things	To provide a context information broker
ETSI	Organization	Information and communications	To provide standards for Internet technologies and systems in communication, which include mobile networks, radio, fixed, broadcast
ISG CIM	Project	Smart cities	To share data/context between different spheres of a smart city
OMA SpecWorks	Organization	Internet of Things	To act as a “specifications factory” for Internet of Things
LWM2M	Project	Sensor networks	The management of low power devices for secure data transfer
IPSO	Project	Smart objects	To foster the use of Internet Protocol (IP) by smart objects
OCF	Organization	Internet of Things	To provide device interoperability
OneM2M	Organization	Internet of Things and M2M	To provide technical specifications for an interoperable M2M service layer

The IoT-A (Internet of Thing Architecture)[8] of the EU FP7 program tries to achieve data interoperability by providing a data format developed for resource-constrained environments, being able to minimize the traffic and the number of interactions by the network. The COIN [51] and IDIRA[9], both from EU FP7, also try to achieve interoperability. COIN[10] provides a semantic based interoperability web solution. It uses the ontology-based approach for the semantic processing. COIN also maintains a knowledge-based system that holds information of distinct entities (i.e., devices, resources). IDIRA provides information sharing between various sources in crisis management scenarios. It uses ontologies to model context and discovery the destination of the shared context information. Even that it works with different sensors, it was developed for a specific crisis management domain.

[8]https://cordis.europa.eu/project/rcn/95713_en.html
[9]https://cordis.europa.eu/project/rcn/98968_en.html
[10]https://cordis.europa.eu/event/rcn/128988/en

One of the Horizon 2020 goals is making data interoperable allowing exchange and re-use between researchers, institutions, organizations, countries, etc. FIESTA-IoT[11] is an example of a project from Horizon 2020 that primes for interoperability. FIESTA-IoT is an infrastructure to provide data interoperability among already deployed IoT systems, platforms, and testbeds. It uses a common ontology to guarantee semantic conformity among different providers. It also provides a standard API for communication giving access to the information by the IoT systems connected to it. INTER-IoT and INTER-Health are also projects that received funding from Horizon 2020. The INTER-IoT[12] project goal is to provide interoperability on heterogeneous IoT platforms. INTER-IoT encompasses other projects to reach the interoperability in different layers: device level, networking level, middleware level, application service level, data and semantics level, integrated IoT platform level, and at the business level. On the data and semantics level, INTER-IoT developed the Generic Ontology for IoT Platforms (GOIoTP[13]) to support the semantic matching in IoT scenarios, facilitating the context sharing process. INTER-Health is a specific project under the INTER-IoT umbrella that presents an application scenario with context sharing [54]. On INTER-Health, messages and entities are described semantically using domain ontologies, facilitating the interoperability. It also has a specific communication bus for sharing context information to the subscribers.

Wise-IoT[14], SEMIoTICS[15], BIG IoT[16], and symbIoTe[17] are also projects from the Horizon 2020. Wise-IoT proposes a Global IoT Services (GIoTS) layer with semantic interoperability ensuring reliability, and end-to-end security. It has a Morphing Mediation Gateway (MMG) component, which translates different protocols and data representations, working with different ontologies. SEMIoTICS is an in developing project that aims to provide a pattern-driven solution for semantic interoperability in IoT environments. It claims to support cross-layer adaption for heterogeneous smart objects. BIG IoT proposes the BIG IoT API, a Web API to be used by the IoT platforms, thus providing interoperability. Also, it provides the BIG IoT Marketplace, making possible for the platforms to share and monetize their data. The symbIoTe project, an abbreviation for *symbiosis of smart objects across IoT environments*, provides an abstract layer for a unified view of different IoT platforms. It has a standardized API for the interconnection of heterogeneous IoT solutions.

FIWARE provides a modular open source framework to foster the development of IoT solutions. The FIWARE framework acts as a middleware in the IoT environments, making possible the interconnection of devices and applications through different options of

[11]http://fiesta-iot.eu/
[12]https://inter-iot.eu/
[13]https://inter-iot.github.io/ontology/
[14]http://wise-iot.eu/en/home/
[15]https://www.semiotics-project.eu/
[16]http://big-iot.eu/
[17]https://www.symbiote-h2020.eu/

communication services. The Orion Context Broker[18] is one of the FIWARE's modules. It has the primary goal of managing context information. Orion acts as a context broker, receiving context from IoT environments and providing options to query or subscribe to a specific context. It also provides mechanisms to query/subscribe to a context by geolocation, type, and format. By acting as a pool of context, Orion may work for context interoperability.

European Telecommunications Standards Institute (ETSI) has established a special interest group to develop context management systems standards [48]. The group is called Industry Specification Group on Context Information Management (ISG CIM)[19] and focuses on smart city applications. The ISG CIM specifics a standard API to provide access for a context management system that focuses on smart cities environments containing heterogeneous data sources. They claim that their approach works for providing real-time access to the context information [49]. The ETSI specification does not try to replace the ways to exchange data between software platforms but offers standards to facilitate the cooperation among different platforms.

Looking deeper into the standardization for IoT, some efforts are made having the goal of standardizing the interactions with IoT devices. Some examples are projects and organizations such as LWM2M[20], IPSO[21], OCF[22], OneM2M[23]. They all make attempts to come up with well-defined protocol stacks, data models and data representations, often using REST as messaging paradigm. They come up with models that describe how to model lights, buttons, inputs, outputs in IoT environments. By doing so, the meaning of data can be easily understood, or additional context can be retrieved from the device itself. This helps in achieve a better level of interoperability and facilitates further processing.

Lightweight M2M (LWM2M) is protocol created for remote managing lightweight and low power devices on a variety of networks [112]. It has an architectural design based on REST, and builds on the Constrained Application Protocol (CoAP) data transfer standard. The Internet Protocol for Smart Objects (IPSO) Alliance focuses on popularizing and incentive the use of Internet Protocol (IP) by smart entities (i.e., devices, systems). It also works to the definition of a framework considering privacy-preserving issues. It makes part of the OMA (Open Mobile Alliance) SpecWorks[24] organization that has efforts on interoperability for IoT. OMA SpecWorks also coordinate the LWM2M protocol.

The Open Connectivity Foundation (OCF) is a group whose solutions are to minimize communication effort between IoT devices. It provides a standard communication platform and data models that allow the communication among devices regardless of their characteristics, such as transport layer technology, application environment, operating sys-

[18]https://fiware-orion.readthedocs.io/en/master/
[19]https://portal.etsi.org/tb.aspx?tbid=854&SubTB=854
[20]https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/
[21]https://www.omaspecworks.org/ipso-alliance/
[22]https://openconnectivity.org/
[23]http://www.onem2m.org/
[24]https://www.omaspecworks.org/

tem. The oneM2M is a well know global standards initiative for Machine-to-Machine (M2M) communications that also considers IoT environments. It provides technical specifications for requirements, architectures, APIs, security solutions and interoperability in IoT technologies. The oneM2M has the primary objective of providing technical specifications to achieve the demand for an interoperable M2M service layer to be used by different systems and devices.

The presented standardization efforts for IoT may help in reaching the context sharing feature. However, even with the growing amount of different IoT devices generating heterogeneous data, there is no standard for context information description [109][18][39]. Moreover, it is hard to impose a context format, one time that it may have different characteristics and providers. Thus, it remains a necessity for the implementation of a context sharing platform.

3.1.5 Challenges and Future Directions

In IoT, there is always a need for interoperability in different aspects. It is common in IoT environments to have various entities such as software systems and physical components from different producers. A platform to provide the horizontal integration of those heterogeneous entities is essential for the proper function of IoT environments. The context information plays a significant role in IoT. It is desired a platform able to provide context information interoperability in such environments [18][109]. Despite some of the analyzed works try to overcome the context interoperability issues providing the context sharing feature, there is no a platform that meets all the context sharing building blocks. Most of the analyzed works focus on a specific application domain/scenario. Thus, they fail in addressing some essential features to provide the complete context interoperability for IoT.

The development and strengthening of the context sharing field in IoT will only occur with the continuation of the ongoing research efforts. Thus, the next items present the major challenges to be addressed in the context sharing for the consolidation of the area.Also, future directions are giving, allowing readers to know which are the next steps in developing context sharing platforms for the Internet of Things environments.

1) Interoperability: It remains a challenge to overcome in the context sharing field. In a significant amount of works, context sharing occurs locally or within a small group of similar entities. They try to care only about local sharing, not concerning the vast heterogeneity present in IoT environments. In most cases, they fail to provide an inter-domain context interoperability. Even that ontology works for the standardization by mitigating the interoperability challenge, and other technologies need to be employed in order to tackle such complex issue. Web services can be used to hide context systems patterns in order to standardize the communication channel used for context information transport. As RESTful technology is commonly used by IoT middleware [102], it can also be used by context sharing platforms as well. Also, WordNet [92] can be more explored to mitigate interoperability issues. WordNet is a lexical database and an open source tool that works with the English language. It can correlate words and expressions semantically. By using WordNet, the context sharing platforms can relate two different entities and their events (i.e., context) [93]. Another concept that can be used to enhance interoperability in such scenarios is the Virtual Object. It appears in the IoT as the digital/virtual representation of the service(s) of a cyber-physical object. The Virtual Object can provide interoperability among heterogeneous objects by using semantic descriptions and context sharing techniques [104]. Moreover, Virtual Objects are gaining momentum in the IoT scenarios as secure lightweight virtualization solutions are being proposed [139][135].

2) Communication Technologies: The use of Low-Power Wide-Area (LPWA) networks must be considered in IoT environments and for context sharing as well. LPWA represents a set of technologies able to provide network communication for a considerable distance with low energy consumption [125]. LPWA networks support (i) low power devices such as the ones that can last for several years on battery, (ii) devices with low data throughput requirements, and (iii) long range operation [132]. The use of LPWA networks by the context sharing platforms, such as SigFox, LoRa, and NB-IoT, has potential to enable new forms of communication, making possible the sharing of context information in most IoT scenarios.

3) Fog and Edge Computing: Manashty et al. discuss how cloud-based context sharing platforms lack in fulfill context sharing requirements [89]. In this sense, the adoption of Fog and Edge computing paradigms towards the architectural perspective can help to cover these requirements. Fog Computing paradigm brings the cloud applications physically closer to the IoT devices. It works in a distributed way. Fog Computing leverages cloud and edge resources along with its own infrastructure [36]. There is an urgency to minimize network communication in IoT by optimizing the systems that exchange data, and also context. The Fog and Edge computing paradigms can tackle this issue as well by improving the scalability of the systems. The Edge Computing paradigm reduces the amount of data (i.e., context) exchanged by the entities. It is related to perform some processing tasks of the systems at the final node of the communication layer (i.e., edge device), directly embedded into the devices themselves minimizing the communication with cloud instances [123]. Context sharing platforms can take advantage of both Fog and Edge computing paradigms to decrease latency and network overhead. Moreover, the decentralization provided by these paradigms helps in solving the heterogeneity problems of IoT environments in terms of systems and devices.

4) Hybrid Reasoning: The implementation of both Fog and Edge Computing paradigms facilitate a hybrid reasoning feature. The use of different reasoning techniques, accordingly to the environment, is a challenge for context sharing platforms. IoT environments tend to be heterogeneous by varying its characteristics (e.g., processing power, entities manufacturer). They can take benefit of a hybrid reasoning approach by adapting easier for each situation [109][86]. Following the hybrid Fog and Edge approach, a lightweight reasoning mechanism (e.g., rule-based system) can be embedded directly into IoT devices (edge) with resource-constrained characteristics. On the other hand, fog devices have more resource capabilities (e.g., processing power, unlimited energy) and can implement more complex reasoning (e.g., machine learning, ontologies).

5) Security and Privacy: Considered a leading challenge towards the definition of a context sharing platform. First, there is a need to protect contextualized information exchanged between entities. In this case, the use of lightweight communication protocols is seen as a viable choice [135]. Also, the context may include private information, such as medical and location data. In this sense, there are some efforts in defining a security architecture for IoT systems [137]. Moreover, there is still a lot to be explored to provide security at the hardware level, especially regarding the context-generating devices [95]. They are the primary targets for attacks and must be protected from the low level of hardware to the high level of software to ensure the integrity of the generated context.

6) Context-Aware Security: Besides the provision of privacy to the platform, context information may also be used for context-aware security decisions [35][5]. Shared context information is often used to improve the knowledge of the receiver, but context information should be used as well in smart environments for authentication, authorization, access control, and privacy-preserving services provision [72]. As analogy, let's consider the scenario of a door. A simple door will open with a key. A door using context-aware security functionalities will adapt itself depending on the environment (i.e., context). For example, a door may require different access control policies depending on the geographical place. While in a specific country, the door will need a physical key for the access, in another country it may require a secret password. Bringing this scenario to the IoT, it is possible to replace the analogy of a door by a computing device, or a system. Furthermore, the context information can improve the communication channel security, by strengthening the network security, sending reports to a manager, or making data anonymous when some event is detected (e.g., an intruder on the network). Context-aware security ensures decision to be made according to the actual environmental context. Some efforts provide context-aware security services to IoT application scenarios [4][42][148]. However, those solutions do not care in providing the context-aware security feature considering the use of context information from heterogeneous application domains. The union of context sharing with context-aware security can leverage new frontiers in the development of security solutions to IoT environments.

7) Context Economy: An emerging trend is the development of marketplaces for IoT data. These marketplaces are community-driven software systems that allow device owners to sell their sensor or the actuator data for a monetary benefit [105]. There are some recent researches that present different kinds of IoT marketplaces [50][141][100][76].

However, most of these works only deal with raw IoT data. Even when the IoT marketplace uses a kind of context [100], it only uses the information of one domain and does not use it to provide new decisions. No IoT marketplace allows sellers and buyers to deal with high-level information (i.e., context) from different domains. To be able to perform such function, the IoT marketplace needs to add a layer of semantic interoperability, to deal with the shared context information. More than commercializing the high-level information, such Context Marketplace can avoid buyers' entities to perform a reasoning process, that is considered a performance demanding task. Moreover, such Context Marketplace has the potential to leverage the development of the area.

3.2 CONTEXT-AWARE SECURITY PROVISION

The context-aware security concept has been in focus of the researchers since the pervasive computing era [68][61]. They have introduced important definitions of the area that are carried along the years. Due to the technological limitations of that time, the context-aware security solutions were used to have a very specific applicability. However, with the recent popularization of the IoT applications, the context-aware security topic become more likely to be applied. Nowadays, many efforts are providing context-aware security solutions in different ways considering the characteristics of IoT environments. Subsection 3.2.1 presents an overview of context-aware security solutions by the application area. Subsection 3.2.2 analyzes the context-aware security solutions by the requirements previously presented in Subsection 2.5.2.

3.2.1 Context-Aware Security Application Areas

This Section analyzes context-aware security systems based on possible application areas: (i) authentication, (ii) authorization, (iii) access control, and (iv) privacy-preserving. Moreover, the scope of each work is presented and if it uses shared context information for CAS (i.e., performs a kind of context sharing).

Table 3.4 presents the comparison of analyzed works. The proposed work appears in Table 3.4 as "**This work**". The architectural and implementation details of "**This work**" can be found in Chapters 4 and 5. In Table 3.4, a dash (—) symbol is used across all columns to denote that the feature is either missing or not mentioned in related publications that are available. The symbol (✓) is used to denote that the feature is employed by the analyzed work in some perspective. For the use of shared context information issue, the analyzed works can be categorized in two groups: full interoperability (FuI), and partial interoperability (PaI). The ones of full interoperability comprise the solutions which use context information of different sources, domains and/or formats to provide security decision. The partial interoperability ones use context information of local or similar groups. Next paragraphs present the definitions of analyzed solutions.

Table 3.4 – Overview of Context-Aware Security Solutions by Application Area.

Solutions	Ref	Year	Authentication	Authorization	Access Control	Privacy-Preserving	Scope	Uses Shared Contex
CASA	[64]	2013	✓	—	—	—	Mobile	—
ConXsense	[91]	2014	—	✓	✓	✓	Mobile	—
Mowafi et al.	[98]	2014	—	✓	✓	—	Mobile	—
SocIoTal	[115]	2015	✓	✓	✓	✓	IoT	PaI
Rachid et al.	[113]	2015	—	—	—	✓	IoT	—
Gansel et al.	[56]	2015	✓	—	✓	—	Automotive	—
SVM-CASE	[81]	2015	✓	—	—	—	VANET	PaI
CAS RBAC	[142]	2016	—	✓	✓	—	User	—
CARBAC	[66]	2016	✓	✓	✓	✓	WSN	PaI
ContexIoT	[72]	2017	—	✓	✓	—	IoT	—
CAPP	[148]	2017	—	—	—	✓	Smartphones	—
CRBAC	[6]	2018	✓	✓	✓	✓	Healthcare	—
CSIP	[5]	2018	✓	—	✓	—	Industrial IoT	—
Aegis	[124]	2019	—	✓	✓	—	Smart Home	PaI
Gheisari et al.	[57]	2019	—	—	—	✓	Smart City	PaI
This work	—	2020	✓	✓	✓	✓	IoT	FuI

- **CASA:** Context-Aware Scalable Authentication (CASA) [64] envisions using multiple passive factors to modulate active factors to authenticate users. CASA embodies two concepts. First, the digital sensors data combined with models of people and places can yield multiple passive factors about users' identities. Passive factors are those that can be acquired without explicit interaction from the end-user (e.g., a user's location or time since last login). Second, CASA is based on the idea that this passive multi-factor data can be used to modulate the strength of active authentication needed to achieve a given level of security. CASA makes authentication easier or harder based on passive factors rather than making it uniformly hard for all cases.

- **ConXsense:** It is a framework that provides context-aware access control decisions by performing automatic classification of the context with regard to its security-relevant properties [91]. The classification is based on machine learning models and user feedback providing ground truth information for training these models. The framework architecture is driven by context data observed with the sensors of the mobile devices.

- **Mowafi et al.:** The authors propose a context-aware adaptive security framework for eliciting context information and adapting this information with security control measures [98]. The framework uses context information and dynamically adapts the security settings of mobile applications for different situations and user actions. Context entails a variety of aspects (location, time, network, etc.) that are dynamically combined together to create a certain enriched context. Context aspects are utilized to adapt the security level required by each mobile application.

- **SocIoTal:** The authors present a framework developed under the foundations of the EU FP7 SocIoTal project [115]. The Architecture Reference Model (ARM) is used as a reference architecture in the development of the framework. It addresses security and privacy concerns by instantiating and extending the security functional group of ARM. The proposed framework has two main modules: Group Manager, and Context Manager. The former being responsible for dealing with data sharing within a group of devices. The latter is responsible for the context-aware security provision by a Complex Event Processing (CEP) technique.

- **Rachid et al.:** The authors propose a context-aware architecture for privacy preservation in IoT [113]. It has three main layers: (i) Sensing layer, (ii) Network layer, and (iii) Application layer. The first one (i) is responsible for device data acquisition, the second one (ii) to transmit the sensed data, and the third one (iii) to offer the data as a service. It has the possibility of offering an ontology as a service, that can be used for privacy processing.

- **Gansel et al.:** The authors present a solution focused in automotive scenarios [56]. They propose an access control model that is inherently aware of the context of the car and the applications. The context-aware access control allows for adapting access permissions based on the context of the car or the applications without compromising on safety. Their model grants permissions to exclusively access certain display areas to applications depending on the current context.

- **SVM-CASE:** SVM-based Context-Aware Security Framework (SVM-CASE) [81] uses the Support Vector Machine (SVM) learning technique to automatically determine the boundary between the misbehaving nodes and well-behaved nodes in VANETs. SVM-CASE uses the vehicles context information, such as velocity, temperature, and altitude to detect malicious nodes in a network. It also uses the network context to provide security services. For example, if a node in a network is dropping packets, it can be caused by a network context of a busy channel and will not be considered a misbehaving node. In a context-unaware solution, it would be treated like a misbehaving node without any further investigation on context. Thus, SVM-CASE works for context-aware security authentication.

- **Trnka et al.:** The authors propose a solution that extends role based access control (RBAC) with certain context awareness elements [142]. It is based on using security levels, which are granted to user based on his context. Their solution enables different contexts to have different security methods. For example, an entity with a specific role can have different access rights in different cities, and the authentication process can be even omitted depending on the context (e.g., access from internal company network). Moreover, it can be used alongside a legacy system, in which security policies need to be defined for granting access to users depending on its context (e.g., location, status, time).

- **CARBAC:** The authors propose a Context-Aware Role Based Access Control (CARBAC) scheme [66]. It controls the access of the users to the system in accordance to their role in the system and the current context information. For example, a user may have access restrictions in different locations and time of the day. CARBAC roles can also change at run-time, depending on the context. Moreover, it can protect the privacy of the data by defining business rules. CARBAC access control scheme is modeled using ontological techniques and Web Ontology Language (OWL), and implemented via CLIPS business rules tool. CARBAC can support context-aware security in authentication, authorization and access control.

- **ContexIoT:** It is a context-based permission system for IoT platforms that provides contextual integrity [72]. ContexIoT has the premise that each permission granted to the user may happen under a specific context. It is a smartphone app developed using the Samsung SmartThings platform that is able to analyze the context data flow to provide context-aware security. ContexIoT analyzes the runtime context together with a backend repository for inferring security decisions.

- **CAPP:** Zhang et al. [148] designed a context-aware privacy-preserving algorithm (CAPP) for smartphones applications to decide in which way the context of a user can be disseminated. The context is acquired through different sensors embedded in the smartphones. CAPP uses the temporal correlation between contexts of the users for security services provision. The main focus of CAPP is to provide privacy-preserving security service, but the reasoning process of the algorithm can serve as a basis to provide authentication and access control as well.

- **CRBAC:** The authors propose a new context-sensitive role-based access control (CRBAC) [6]. The proposed solution is inserted on a security and privacy architecture for the Healthcare Internet of Things (HIoT) area. The proposed architecture is responsible for data acquisition, management, and secure communication. The CRBAC model defines context conditions involving roles and attributes to describe policies that can be applied in critical situations. The term “context condition” (CC) is used by the authors as a key-value pair to achieve safety, security, and privacy of the CRBAC users.

- **CSIP:** Context-sensitive seamless identity provisioning (CSIP) is a mutual authentication framework for Industrial Internet of Things (IIoT) [5]. The authors defines the inhabitants of IIoT scenarios as people, devices, services, systems, sensors, 5G smartphones, etc., in varying applications in daily life. CSIP builds an inhabitant profile by using his activities' history and usage patterns of the environment's resources, based on that, it can build disposable customized virtual inhabitant profile (DCVIP), then it creates an identity proxy to perform the verification required during the interaction for the authentication process.

- **Aegis:** It is a context-aware security framework to detect malicious activity in Smart Home Systems (SHS) [124]. Aegis captures sensor-device data in smart home scenarios to understand the context of the user activity. With this data, it is possible to detect malicious behavior and alter users about it. Aegis observes the changes in device status based on user activities and builds a contextual model to differentiate benign and malicious behavior. Moreover, the authors introduced an adaptive training model to improve the malicious behavior detection mechanism.

- **Gheisari et al.:** The authors first equip IoT-based smart city with Software Defined Networking paradigm (SDN). Then, they mount a privacy-preserving method on top of it that manages flowing data packets of IoT devices data [57]. The SDN classifies all connected IoT devices data based on the context. The authors claim that privacy is preserved through the SDN function of splitting sensitive data and sending split parts through a secure route and a VPN.

Although all the analyzed works provide solutions related to context-aware security, they may differ in its architecture and how they provide security. Each work has its focus, one time that ones have the objective to protect the whole infrastructure, there are systems with a specific goal. By using context information to provide security, most of the systems were deployed for dynamic situations, where the location and status are an important element.

The works [115], [81], [66], [124], and [57] use shared context information for the security decisions. Aegis [124] has a Data Collector Module that creates an uniform context format for all contexts present in its architecture. SocIoTal [115] also provides a common format for the context. SVM-CASE [66] approach uses an ontology to facilitate the interoperability between shared context. Gheisari et al. [57] builds an SDN controller to connect all devices through two OpenFlow switches, creating an interoperable communication. However, the source of this context information is from the same or similar entities of the systems deployed many times in a near location. In this sense, they could not reach full interoperability (FuI). To reach it, the system must use context information from heterogeneous sources deployed in different locations or networks.

The majority of examples related to the context-aware security area are linked to the Access Control feature. Most of the analyzed works reach this feature. Moreover, most of the techniques for Access Control also provide Authorization feature. On the other hand, only a few works focus on offer Privacy-Preserving. Although analyzed works provide context-aware security features in their architecture, they do not care about the heterogeneity of the

IoT environments. A solution that considers this requirement and reaches a full interoperability is needed to mitigate the challenges of the area.

3.2.2 Summary of Context-Aware Security Solutions

A comparative summary of recent context-aware solutions focused in IoT environments is presented in Table 3.5, a dash (—) symbol is used across all columns to denote that the feature is either missing or not mentioned in related publications that are available. The context-aware security requirements (See Subsection 2.5.2) are consider for the analysis. For all the requirements, it is analyzed the characteristics implemented by the context-aware security solution and not the specific technologies that it uses. For example, a solution can address the Privacy requirement if it has a process in its architecture for hiding the user data in some context. If a solution only uses a secure communication protocol (e.g., DTLS, TLS, wolfSSL) for protecting data, it is not considered meeting the Privacy requirement.

Moreover than context-aware security, the context information also has been used for anomaly detection even before the IoT era [28]. Some of the analyzed works act in context-aware security field by the detection of anomalies [81][72][124]. Such approaches use context both from users and the environment to detect abnormal network/system activity based on the historic data and/or in an unusual behavior. The concept of anomaly detection was successfully employed to the Service-Oriented Architectures (SOA), that are common in IoT environments [149]. When properly implemented, anomaly detection can deliver significant security improvements to IoT environments. However, it represents a subset of the security provision possibilities achieved by the use of context-aware security solutions. In a context-aware security solution, the anomaly detection can act as a trigger to the provision of different Security Services.

The only two requirements performed by all the analyzed works were *Context acquisition* and *Context processing*. It is expected for the solutions to do that, as the context-aware security solutions should get the context from a source (i.e., Context acquisition) and process it for taking the security decisions. This process also can be called inferencing (i.e., Context processing). A considerable amount of analyzed works performs the Context acquisition by getting context from user‘ mobile device [64][91][98][66][72][5]. Rachid et al. [113] proposes a different approach by getting context trough a sensing layer from a wireless sensor network. Gheisari et al. [57] has defined a Software-Defined Networking (SDN) for acquiring context. On the Context processing requirement, the use of rules is the most popular technique [98][56][142][66][72][57]. Rules are simple to define, easy to use, and lightweight, being popular in the context-aware field [109]. The usage of both ontologies and machine learning techniques can be considered a trend in this field [64][91][113][66]. Moreover, such techniques can help in providing interoperability.

Table 3.5 – Overview of Context-Aware Security Solutions by Requirements.

Solutions	Ref	Year	Context acquisition	Context processing	Interoperability	Privacy	Reliability
CASA	[64]	2013	Gets context from the user's mobile device	Uses Naïve Bayes classifier to combine multiple factors, calculating the authentication	It can be used by different smartphones	—	—
ConXsense	[91]	2014	Gets context data with the sensors of mobile devices and user feedback	Utilizes machine learning to classify contexts according to their properties	It can be integrated with the FlaskDroid architecture [24] on Android phones	It limits the adversary to gather information with high privacy exposure	It uses more than one font for acquiring the context
Mowafi et al.	[98]	2014	Gets the context from the smartphone sensors	Uses rules to make the appropriate security decision	It can be integrated with the mobile device OS	—	—
SocIoTal	[115]	2015	Gets context from internal or external device's sensors	Uses key-value pairs and markups for modeling and Complex Event Processing for reasoning	Translates raw context data into a proprietary common format	A repository of privacy rules is used to define privacy preferences of users	It has a component that allows smart objects to obtain data from other entities in a reliable way
Rachid et al.	[113]	2015	Gets context trough a sensing layer from a WSN	Uses an ontology-based reasoning technique	An ontology acts as a common vocabulary for the context across the various system components	Uses an ontology that describes the privacy of users	—
Gansel et al.	[56]	2015	Gets context about the status of the car, the environment, and user	Uses rules for a dynamical access control	—	An application requires a permission restricted to certain contexts	It uses a microkernel based hypervisor
SVM-CASE	[81]	2015	Gets context from network and nodes/devices	Uses the Support Vector Machine (SVM) algorithm to classify nodes	Provides interoperability between different automotive nodes	—	Verifies if a node becomes misbehaving using context information
CAS RBAC	[142]	2016	Uses both real-time and historical users context (e.g., location, time)	Uses rules to determine access control levels	It is interoperable with solutions using Role-based Access Control (RBAC)	—	It has mechanisms to validity check the context information
CARBAC	[66]	2016	Gets context from users devices (e.g., smartphone)	Uses OWL and CLIPS rules to perform reasoning	By using ontologies, it is able to deal with generic data	Provides data privacy by security rules	—
ContexIoT	[72]	2017	Collects context from the smartphone in installation and runtime	Uses rules to compare contexts in a key-value pair	It uses the Samsung SmartThings platform to prototype the solution	—	It keeps a runtime logging for the context events
CAPP	[148]	2017	It acts as a middleware for the smartphone and can get context directly from smartphones sensors	It uses Markov Chain for both model and reasoning on context to provide privacy	It can be used by different smartphones	It protects user's privacy context from untrusted smartphone apps	It performs check methods before release the context of a user
CRBAC	[6]	2018	Gets context from previously connected devices	Uses a rule-based processing for access control	—	Replaces the original data identities with unique privacy labels	It records the history of every data exchange between the entities
CSIP	[5]	2018	Gets the context from sensors/smartphones of medical domain	Uses data mining techniques to extract patterns for future reasoning by comparison	Synchronizes data blocks with a cloud-based side	Uses a secure session-key for access to private data	Stores the processed data for learning purposes
Aegis	[124]	2019	Collects the state of smart home devices and sensors (active or inactive) autonomously	A Markov Chain-based machine learning model is used to detect malicious activities	It has a data array format for parsing the collected context	It assigns an anonymous ID for each user to ensure privacy	—
Gheisari et al.	[57]	2019	It has a Software-Defined Networking (SDN) for getting the device's data	It uses rules to determine the sensitivity level of each data	Different devices can participate a defined SDN	It defines that sensitive data not be disclosed unintentionally	It splits sensitive data and sends split parts through a secure route

The *Interoperability* requirement can be linked with the capacity of the solutions to use shared context information (see Table 3.4). Also, it is related to the solution effort in dealing with different entities (e.g., data, devices). Some analyzed works provide interoperability only with the same kind of entities [64][91][81][72][148]. SocIoTal [115] and Aegis [124] work in providing a common data format, making easy the interoperability after a parsing processing. Rachid et al. [113] and CARBAC [66] perform a similar processing, using an ontology to create a common vocabulary between the entities.

Privacy and *Reliability* are the less addressed requirements by the analyzed works. It is import to keep context information private, as many times it relies on sensitive user information [42][109]. For *Privacy*, most of the analyzed works perform a kind of processing, either by rules or different condition manager, to set the privacy level of context data [115][113][66]. CRBAC [6] perform an anonymization privacy processing by replacing some elements of the context information for unique data, keeping it private. Aegis [124] does not store user data from smart home devices which reduces the privacy risks and concerns from prior solutions. There are different approaches employed by the analyzed works to ensure the *Reliability* requirement. Most works perform a redundancy processing to minimize failures [91][72][5][57].

It is essential to research, study, and develop context-aware security solutions towards the evolution and consolidation of the IoT. The analyzed solutions provide context-aware security in different ways in their applications, but they still fail to cover all the key requirements of context-aware security in IoT. The most common drawbacks of the context-aware security solutions are related to data protection (i.e., context information). It is important to follow the new legal regulations such as the General Data Protection Regulation (GDPR) [140] to ensure that privacy-sensitive data is not leaked. Also, mostly context-aware security solutions do not care about the high heterogeneity of IoT environments by providing a full interoperable mechanism for context information.

4. ARCHITECTURAL APPROACH

This chapter presents the architectural view of developed work. Section 4.1 presents the Context Sharing Architecture modules. Section 4.2 presents the modules responsible for the context-aware security provision. Section 4.3 presents an application scenario to the architecture.

4.1 CONTEXT SHARING ARCHITECTURE

The importance of having a context sharing architecture is strongly related to the need of users and applications to share information between different entities (e.g., devices, places, users). Although the context sharing concept is used in the pervasive computing area by some systems, the sharing mostly occurs locally with a small group of similar entities. To the best of our knowledge, a context sharing architecture that uses shared context information for context-aware security provision and works with IoT environments was not deployed yet. An Edge-centric Context Sharing Architecture could be used as a reference in order to define a context sharing architecture to share contextualized information between heterogeneous entities and/or environments. The architecture may provide an infrastructure that could be used by other systems to develop their sharing platforms following the proposed architecture definition in full or in parts.

An Edge-centric Context Sharing Architecture takes benefit of fog and edge computing approaches to minimize network communications, thus reducing failure points and improving scalability. The definition of the architecture is illustrated in Figure 4.1. It is composed of three main layers: Context Storage, Context Sharing, and Context Provider. Context Storage is placed at the cloud level. It is responsible for storing historical context information that may be used by the architecture itself or by accredited third-parties software. Context Sharing System is placed at the fog level and is responsible for sharing the context information with other entities or Context Sharing Systems instances, including different fogs. Context Provider Systems are placed at the edge layer, embedded or connected directly to IoT devices, and are responsible for providing context information based on IoT devices data. The users and applications communicate directly with the Fog Layer. They do not have direct access to the Cloud storage infrastructure.

A real-world scenario can have multiple instances of Context Sharing and Context Provider systems. In most cases, the number of Context Providers systems will be greater than Context Sharing systems. For example, let's consider the scenario of a smart city. It can have one instance of the Context Sharing system responsible for the Emergency Medical

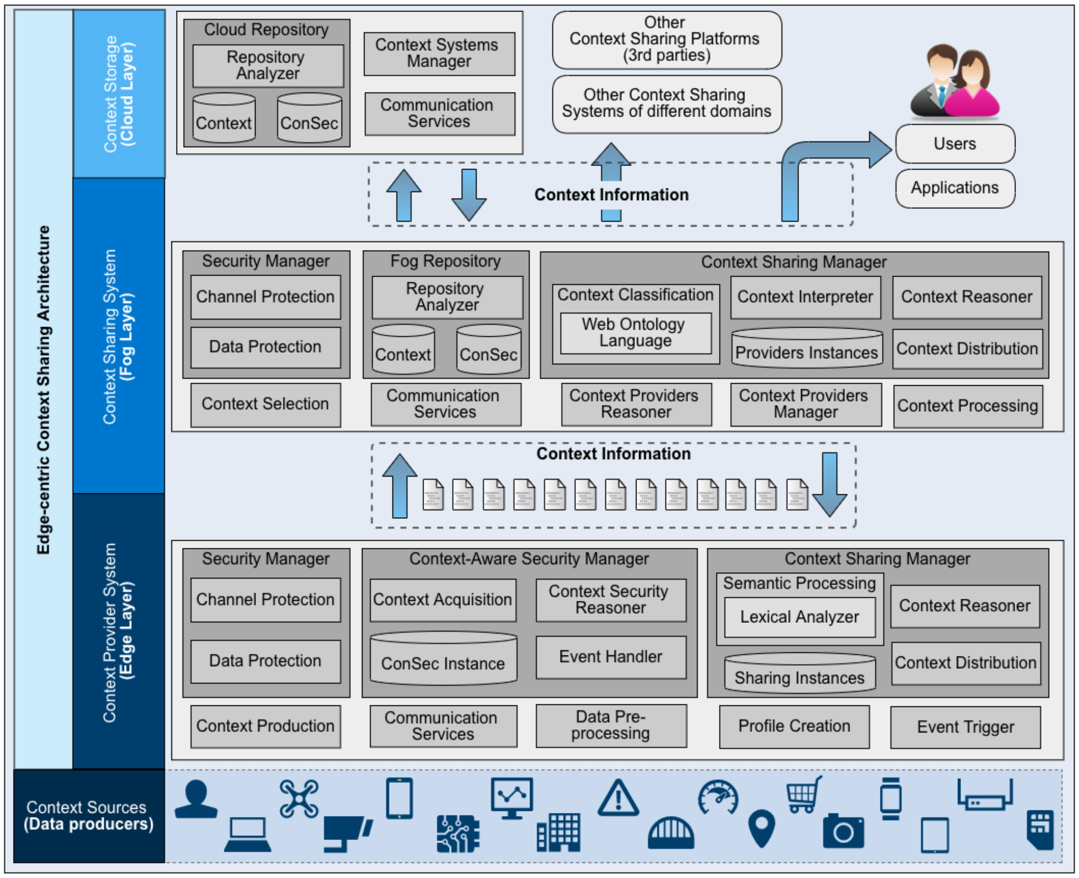


Figure 4.1 – A layered Edge-Centric Context Sharing Architecture.

Services (EMS) of the city and one instance of the Context Provider system deployed on each ambulance of the city.

Both Context Sharing and Provider Systems exchange context information. This context is acquired by the Context Provider Systems and shared with Context Sharing Systems. An example of a context can be seen at Figure 2.3. It contains all the information that will be shared and can be also presented in XML format.

Besides the Fog and Edge layers of the Edge-centric Context Sharing Architecture, it also has a Cloud layer (i.e., Context Storage) for store all the context information, context-aware security information, and Context Sharing and Providers Systems information. The Context Sharing Systems acts as a light Cloud with only information about the deployed scope, one time that the Cloud has information about all the instances. Next, the Context Sharing and Provider System modules are detailed. Also, an Application Scenario with the communications by the systems is presented. Technological details can be found in Chapter 5.

4.1.1 Context Storage

The Context Storage is a layer placed at the Cloud level. Figure 4.2 details all the layer modules. It has the main responsibility of store all the relevant context information of the whole architecture. Next items details the modules of the layer.

- **Cloud Repository:** It is composed of a Repository Analyzer and two databases. The Analyzer works for query and operations in the databases. The first database is related to the context information. The second one is related to security decisions taken using context. It is up to the Fog layer to decide which information can be stored locally and which is stored at this level (i.e., cloud). The Repository Analyzer has the directives to access the stored information.
- **Context Systems Manager:** It is responsible for maintaining information about the Context Sharing Systems that have data stored at the Cloud Repository. Every time that a new Context Sharing System store/gather information, it is created a log with its ID, URL for communication, and possibly other details that characterize such entity.
- **Communication Services:** This module is essential for the communication between cloud-to-fog. It has all the communication interfaces to guarantee the data exchange in an interoperable way.

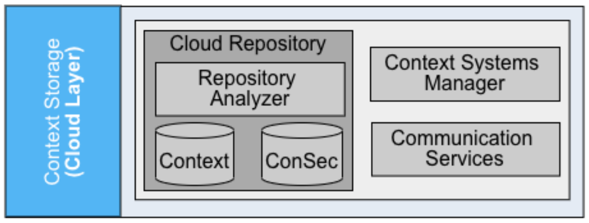


Figure 4.2 – Context Storage modules.

4.1.2 Context Sharing System

The Context Sharing System is a software system placed at the Fog layer. Figure 4.3 details the system modules. It has two main responsibilities: (i) coordinate the Context Providers Systems of his domain, and (ii) receive context information and share it with whom it may concern. Next items details the modules of the system.

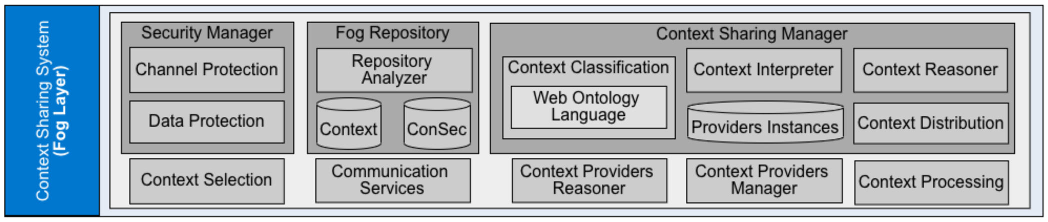


Figure 4.3 – Context Sharing System modules.

- **Security Manager:** A Context Sharing System must ensure its protection against attacks on data and communication channels [83]. Authentication, authorization, confidentiality, and integrity must be used to protect the proposed architecture. Researchers have proposed a security architecture for IoT middleware [115]. In this sense, that architecture can be deployed in the Context Sharing System as well.

- **Fog Repository:** It is composed of a Repository Analyzer and two databases. The Analyzer works for query and operations in the databases. The first database is related to the context information. Every updated context information regarding an instance (e.g., context provider) is stored in this database. The second one is related to every security decision taken using context. Both databases are replicated in a Cloud (i.e., working like a log system), and only the newest information stays at the Fog level. Fog Repository module also works for scalability. When an entity claims for context information, it first verifies if the information is updated in the Context database. If yes, there is no need to go for it through the Context Provider System (i.e., Edge Layer).

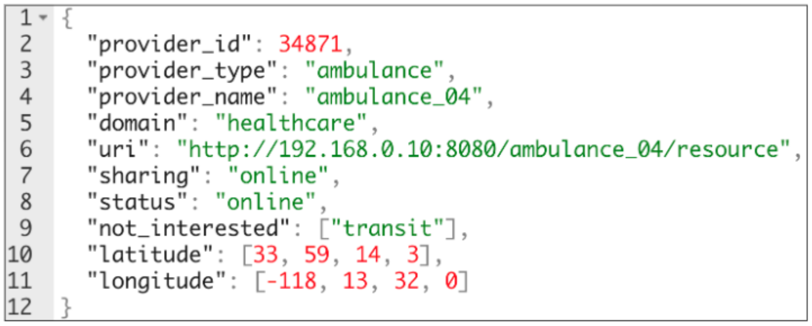
- **Context Sharing Manager:** It is responsible for the context information sharing process. It has the functionality of interpreting the context information sent by the Context Provider, reasoning over it to discover whom may be interested on in, and disseminating it to the destination by using the Fog Repository. The reasoning process happens alongside with the **Context Classification** module, using its ontology. The Context Classification output is the domain of the entities that may be interested in such context. The Context Interpreter and Context Reasoner modules are responsible for the interaction with the **Context Classification**. The Providers Instances database module has the address of the context providers systems inserted on its domain. Those address can be used to share context information. A large database with all the address is deployed at a Cloud level. The database of the Fog level (i.e., Context Sharing System) contains only the entities address of a defined location (e.g., city, state, neighborhood), depending on the application scenario needs. The process of sending context information to other entities is performed by the Context Distribution module.

  Context Classification: This module works regarding interoperability. As differences in network connectivity should be hidden by the use of standardized web service interfaces, the context interoperability can be hidden trough the use of OWL (Web Ontology Language). An ontology can be used to analyze, interpret, and model context information in most IoT environments. The Context Classification module holds an ontology developed for classifying the context according to its domain. Different already developed ontologies can be merged with it, as there are a considerable amount of ontologies in the literature defining the characteristics of different IoT application domains [109][39].

- **Context Selection:** It works with the Fog Repository. The main function of this module is to select the context information requested by an entity (e.g., Context Provider System or another Context Sharing System). It takes place when an entity is not interested in share context information but only to get it from the Fog.

- **Communication Services:** This module is essential for the communication between cloud-to-fog, edge-to-fog, and fog-to-fog. It has all the communication interfaces to guarantee the data exchange in an interoperable way. This module works alongside the Context Sharing Manager for communication with other entities.

- **Context Providers Manager:** Context Providers must register to the Context Sharing System by informing its characteristics and protocols. The registration happens when the Context Providers system sends its profile to Context Systems following a pre-defined pattern. The profile must have the basic characteristics of the provider. Figure 4.4 shows a simple example of a Context Provider profile representing an ambulance. The profile follows standards of widespread sources, such as FIWARE[1], and oneM2M[2].



```
{
  "provider_id": 34871,
  "provider_type": "ambulance",
  "provider_name": "ambulance_04",
  "domain": "healthcare",
  "uri": "http://192.168.0.10:8080/ambulance_04/resource",
  "sharing": "online",
  "status": "online",
  "not_interested": ["transit"],
  "latitude": [33, 59, 14, 3],
  "longitude": [-118, 13, 32, 0]
}
```

Figure 4.4 – Example of a Context Provider profile.

- **Context Providers Reasoner:** It works alongside with the Context Sharing Manager for understanding who are interested in the shared context. It is possible to register the range of each service that can receive or send context information. For example, the EMS can be composed of 15 ambulances, and each one is responsible for a sector

[1] https://www.fiware.org/
[2] http://www.onem2m.org/

of the city. Moreover, the city may have four hospitals of different specialties (e.g., transplantation, emergency, radiology, etc.). Therefore, the city will be geographically separated in sectors. In this sense, when an event occurs (e.g., home-care patient needing medical assistance), the context sharing architecture automatically knows to which services share the context information based on the environment conditions (i.e., patient severity and its location, ambulances location).

- **Context Processing:** The operations on context are essential for the flexibility of the architecture. This module provides the operations of aggregation and filtering in the context information. It works alongside the Context Sharing Manager, once the context can suffer modifications before be shared with the destination.

4.1.3 Context Provider System

Besides a request by the user/application at the Fog layer for context information, an automatic sharing action is started by the Context Provider System when new context information is generated. This system is essential for the architecture workflow. It enables the decentralization, one time it is placed at the edge of the network. The Context Provider System can be deployed alongside or within an IoT data producer device (i.e., Context Sources, see Figure 4.1). Figure 4.5 details the system modules. It has two main responsibilities: (i) provide context-aware security decisions, and (ii) trigger the context sharing process. Next items details the modules of the system.

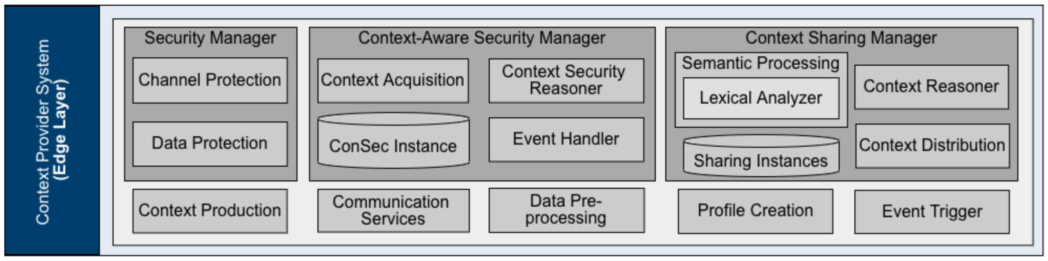


Figure 4.5 – Context Provider System modules.

- **Security Manager:** As the Context Sharing System, the Context Provider system also offers protection to the data (e.g., context information) and communication channels. This system can employ lightweight protocols such as Datagram Transport Layer Security (DTLS).

- **Context-Aware Security Manager:** It is responsible for the context-aware security decisions. It can acquire new context information to match with the most updated one to reason over it and provides access control, authorization, authentication, or privacy-preserving. It may contact the Fog instance for updated context information. The reasoning process occurs with lightweight techniques taking into account the limitations of edge computing environments. More details of this complex module can be seen next in Section 4.2.

- **Context Sharing Manager:** Responsible for the context sharing process at the edge. It has two main functions: (i) modeling the context information to share it with the Context Sharing System infrastructure at the Fog level, and (ii) interpreting the received shared context information through the **Semantic Processing** module for the context sharing processing and/or to make a context security decision alongside the Context-Aware Security Manager. It also has the address of the Context Sharing System instance that the context information must be sent by the Sharing Instances module. A Context Provider can share the context information to one or more Sharing Systems.

  Semantic Processing: It is responsible to interpret context data, enabling context sharing by explicitly defining contexts in a semantic manner. The Lexical Analyzer splits the context data into parts and gathers all the words present in such context information. It extracts the words and retrieves their synonyms and related terms. With this information, it is possible to infer the domain of the context information by comparing the words set with the possible IoT domains present in a pre-defined scope (i.e., ontology). Only the extraction process occurs at the Edge layer, as it is a more lightweight process than the comparison that happens at the Fog layer.

- **Context Production:** It is a subsystem that has the function of produce the context information. The Context-Aware System (CONASYS) is a context production platform that can be used as a Context Production in the Edge-centric Context Sharing Architecture. Although the proposed architecture accepts any Context Production, CONASYS can be used as an example since it was developed by the GSE/PUCRS [90][40].

- **Communication Services:** This module works only for the edge-to-fog communication. It has all the communication interfaces to guarantee the data exchange in an interoperable way. It works alongside the Context-Aware Security Manager and Context Sharing Manager for communication with other entities.

- **Data Pre-processing:** The architecture addresses scalability and real-time sharing in two ways. First, data acquisition from Context Providers at the edge should be minimized by reusing common data accessed by multiple applications through the Repository Analyzer module at the Fog. Second, a data pre-processing method should avoid data stream from the Context Providers. This module has methods to concatenate multiple context information with near generation time to avoid various communication. Moreover, it can be programmable to send context information only if it was different from the last sharing. Also, most of the systems do not take advantage of the decentralization approach provided by the edge computing paradigm that helps to mitigate the scalability issues.

- **Profile Creation:** Responsible for creating the Context Provider profile to be registered in the Context Sharing System. An example of profile can be seen at Figure 4.4.
- **Event Trigger:** One time that context information is produced, this module guarantee it to be shared with the Context Sharing System automatically. Event Trigger acts alongside with other two modules: (i) Context Production to receive the context information, and (ii) Context Sharing Manager to start the sharing process.

4.2 PROVIDING CONTEXT-AWARE SECURITY

Context should be a first-class security component in order to drive the behavior of IoT devices. This would allow smart objects to be enabled with context-aware security solutions, in order to make security decisions adaptive to the context in which transactions are performed. At the same time, context information should be managed by taking into account security and privacy considerations. In particular, current IoT devices (e.g. smartphones) can obtain context information from other entities of their surrounding environment, as well as to provide contextual data to other smart objects [115]. These communications can be performed through lossy networks and constrained devices, which must be secured by suitable security mechanisms and protocols.

Additionally, the IoT is composed of sensitive domains, such as healthcare, transportation, home-care, etc. It is highly significant to protect the privacy and confidentiality of personal data from unauthorized access while stored or transmitted. It is even more crucial and difficult to administrate the information and physical security in ubiquitous environments with numbers of participants continuously joining and leaving the space [66]. Moreover, high-level context information can be reasoned and inferred by considering privacy concerns. Thus, a smartphone could be configured to provide information about a person's location with less granularity (e.g. giving the name of the city where he is, but not the GPS coordinates), or every long periods of time in order to avoid daily habits of that person could be inferred by other entities [115].

The proposed solution for Context-Aware Security (CAS) can provide authentication, authorization, access control, and privacy-preserving to edge computing environments (i.e., Context Providers) by using shared context information. Figure 4.6 shows an overview of the proposed CAS solution, the Context-Aware Security Manager module. The core operation to provide context-aware security is by using pre-defined security rules. These rules are mostly defined for a specific domain that the system is deployed. The system basically works as follows steps: (i) it receives the shared context information, (ii) matches the received context with the historic one, (iii) infers security decisions by the rules. There are some details of these steps that are defined in the next paragraphs. Next items present the components of the Context-Aware Security Manager module.

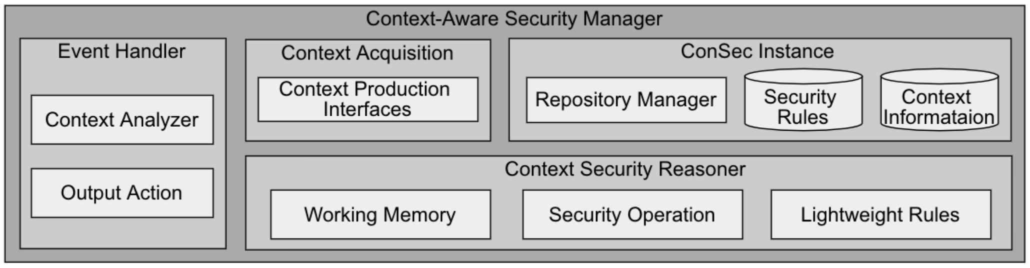


Figure 4.6 – Overview of Context-Aware Security Manager.

- **Event Handler:** It is responsible for receiving the shared context (i.e., new event) information and analyze it alongside with ConSec Instance and Context Security Reasoner modules. The Context Analyzer manages the context information. It can interpret the context information to extract any data (e.g., source domain, type of data, source device) to help in the selection to which kind of rules the context information must be submitted. After this process, the Output Action notifies the Context Security Reasoner with the event (i.e., context information).

- **Context Security Reasoner:** This module is responsible for the reasoning process of the Context-Aware Security Manager. The Security Operation classifies the context with a type of operation (e.g., authorization, access control, etc.), then the reasoning process starts to provide a security action (e.g., give access to an entity, change a status, acts on a device). The Working Memory module fire the Lightweight Rules that infer possible security decisions. These rules scheme is defined via business rules implemented at the Edge layer. The rules can be pre-defined at the development phase or inserted in the Security Rules database in the ConSec Instance module. The Context Security Reasoner module acts alongside the Security Rules of the ConSec Instance module by having a direct connection to it for the context-aware security provision.

- **ConSec Instance:** As reasoning process needs context information to match with the rules, it uses this module to query for it. This module is composed of two databases: (i) Context Information and (ii) Security Rules. The first one has a historical set of the context of past events. The second one is composed of security directives (e.g., IF *contextA* AND *contextB* THEN *giveAccess*) that will be converted into rules by the Context Security Reasoner module. The Security Rules module is composed of pre-defined information defined at the architecture‘s implementation time. The rules may vary depending on the application domain. The Repository Manager helps in access the two databases and update it when needed.

- **Context Acquisition:** Context information has a short lifetime once the IoT is composed of devices that eventually move or change status. The primary function of this module is to get new context information when needed. It has the Context Production Interfaces to make easy the connection with the subsystems that will produce the context information. In this sense, it could perform a request, receive new context information from the Context Production (see Figure 4.5), and update the Context Information database of the ConSec Instance module. Moreover, it can get context information from the Fog Repository module at the Fog layer (see Figure 4.3).

4.3 SMART CITY APPLICATION SCENARIO

A smart city is a complex IoT environment for having heterogeneous systems placed at different domains. It encompasses many different application domains (e.g., healthcare, home-care, urban traffic, EMS) interacting with each other to provide efficient services to the population.

This Section aims to provide a practical view on the application of the developed modules aforementioned. Figure 4.7 shows the representation of an Edge-centric Context Sharing Architecture application scenario. The main focus of this scenario is on sharing the context of a home-care patient when some important events related to the health condition occurs. Moreover, it also shows how the context information can be used for context-aware security decisions.

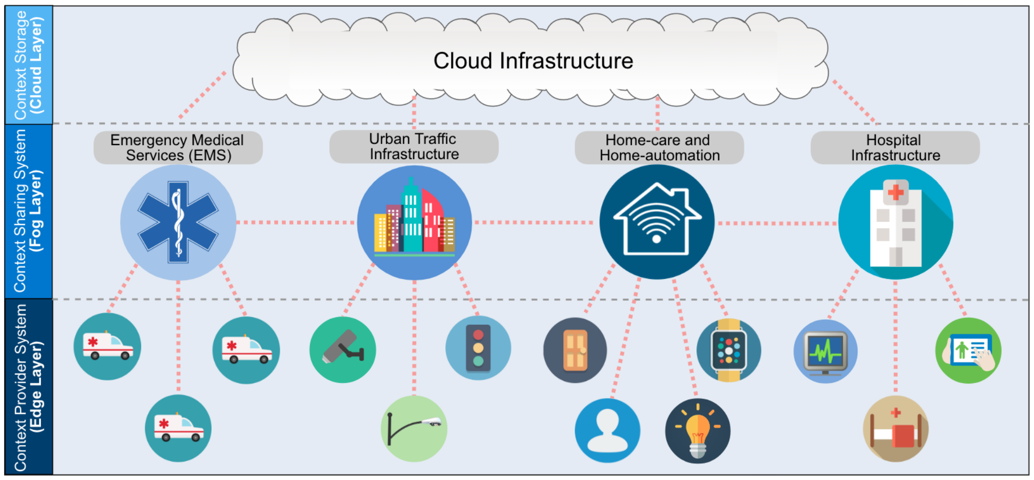

Context Storage (Cloud Layer)
Context Sharing System (Fog Layer)
Context Provider System (Edge Layer)
Cloud Infrastructure
Emergency Medical Services (EMS)
Urban Traffic Infrastructure
Home-care and Home-automation
Hospital Infrastructure

Figure 4.7 – Edge-Centric Context Sharing Architecture instances when applied to a Smart Healthcare scenario.

In this scenario, there are four deployed instances of the Context Sharing System: (i) home-care and home-automation, (ii) EMS, (iii) hospital infrastructure, and (iv) urban traffic infrastructure. Every instance is aware of the environment and knows with whom they must share context information. For every instance of the Context Sharing System it will be various instances of the Context Provider System. For example, the Context Sharing System instance of the home-care and home-automation will have one instance of the Context Provider system connected on it for every device of the house (i.e., automatic door, monitoring camera, etc.). In the EMS Context Sharing System, every ambulance has a Context Provider System.

Figure 4.8 presents a flowchart simulating how the architecture works when an event occurs. The context information acquired by home-care sensors triggers the event of a patient having a heart attack. Context Providers, placed at Edge Layer, send it to Context Sharing System. Having this context information, the system cares in sharing it with whom are interested in such context. The shared context information is formatted like the one presented in Figure 2.3. The instances of the architecture (i.e., Fog and Edge layers) have methods to decompose the context information in parts, thus extracting the necessary data for matching with rules or ontologies to infer decisions.

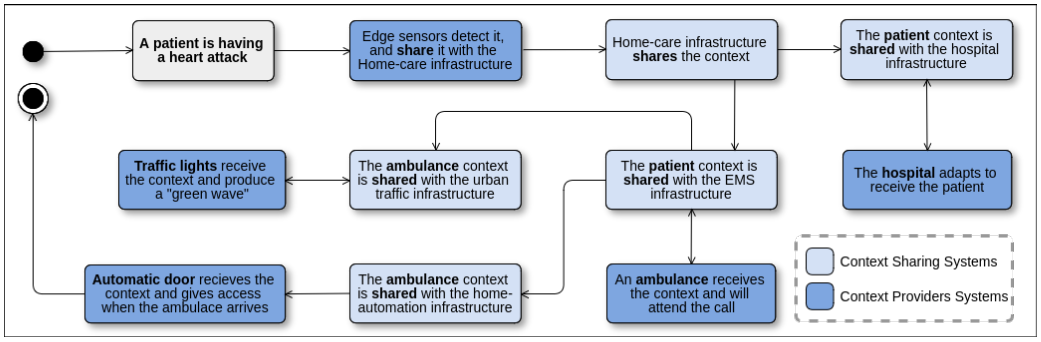

A patient is having a heart attack
Edge sensors detect it, and share it with the Home-care infrastructure
Home-care infrastructure shares the context
The patient context is shared with the hospital infrastructure
Traffic lights receive the context and produce a "green wave"
The ambulance context is shared with the urban traffic infrastructure
The patient context is shared with the EMS infrastructure
The hospital adapts to receive the patient
Automatic door recieves the context and gives access when the ambulace arrives
The ambulance context is shared with the home-automation infrastructure
An ambulance receives the context and will attend the call
Context Sharing Systems
Context Providers Systems

Figure 4.8 – Flowchart for a Smart Healthcare scenario.

The Fog layers (i.e., Context Sharing System) (see Figure 4.1) instances of the architecture are responsible for sharing context information with other fogs. This sharing occurs three times in this application scenario: (i) the patient context is shared with EMS, and hospital infrastructures, (ii) hospital context is shared with the EMS, and (iii) EMS context is shared with home-automation and urban traffic infrastructure.

The received context information can be used in new processing. In this scenario, the ambulance context is shared with the urban traffic infrastructure, thus the city adapts itself by creating routes to drain the traffic with a “green wave” in traffic lights. Moreover, context sharing is not only useful for live data provision, but also for other systems that might lack the necessary historical context required to detect and predict unforeseen anomalies successfully [89].

A home-automation infrastructure provides context-aware security decisions based on the shared context information. It receives context information of the EMS system that may contain an ambulance location and arrival time at the patient's home for medical care. The home-automation infrastructure may give access to the paramedics when they arrive by unlocking the automatic door, one time that the patient could not be able to do that. This process describes a possible context-aware access control function.

5. CONTEXT SHARING ARCHITECTURE IMPLEMENTATION

This chapter presents the technical view of developed work. Section 5.1 presents the Reference Platform used to develop the Edge-centric Context Sharing Architecture. Section 5.2 details the modules responsible for the context sharing process. Section 5.3 presents the used techniques for the context-aware security provision.

5.1 REFERENCE PLATFORM

Figure 5.1 presents the proposed context sharing architecture in a simplified view. The highlighted (i.e., red with dashed borders) modules present in Figure 5.1 represents the ones adapted and extracted from the Reference Platforms for the proposed Context Sharing Architecture.

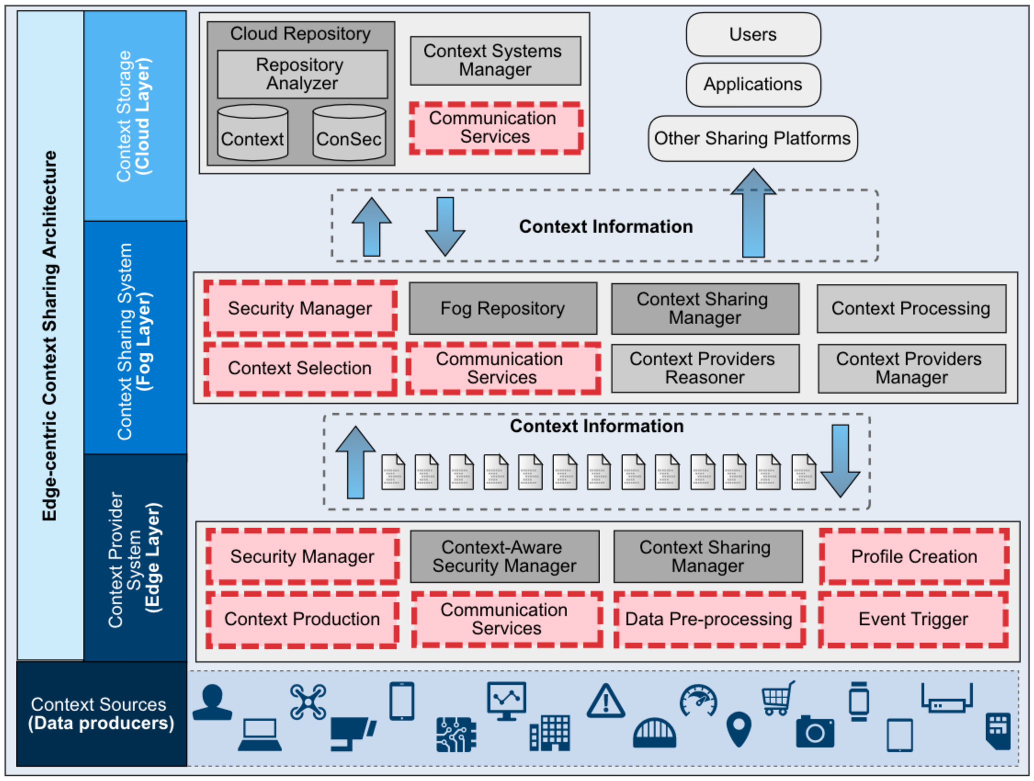


Figure 5.1 – Reference Platform modules of the Context Sharing Architecture.

Two main complementary Reference Platform were used: (i) COMPaaS, and (ii) CONASYS. Both were developed by the GSE/PUCRS research group. The choice for COMPaaS and CONASYS was made by the easy access to the source code, architectures that support services provision, and because CONASYS supports context-aware features. The implementation of proposed architecture for context sharing use parts of the source code of both COMPaaS and CONASYS.

COMPaaS (Cooperative Middleware Platform as a Service) is an IoT middleware [8][7]. 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 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 5.2 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.

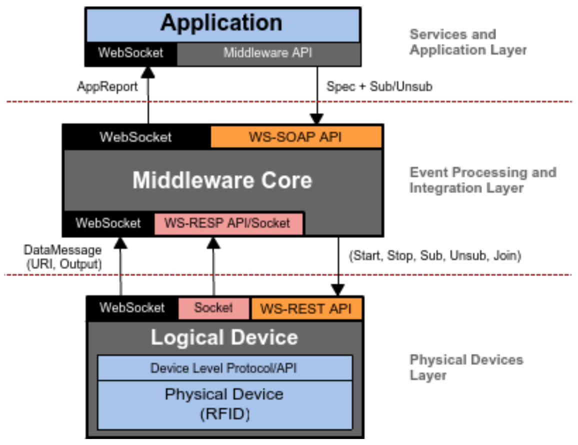


Figure 5.2 – COMPaaS architecture overview.

The following modules presented at Figure 5.1 inherited COMPaaS features: Communication Services, Security Manager, Data Pre-processing, Profile Creation, Event Trigger. The Logical Device present in COMPaaS can be abstracted as an Edge Layer system. 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 IoT environments. It uses 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 without a synchronous request.

COMPaaS also provides a security data management by implementing a security architecture for IoT environments [137]. Security for IoT middleware encompasses procedures that include: embedding keys in system entities; establishing access control policies together with authentication to allow access to networks; usage of security services to protect data against tampering and eavesdropping; and the development and selection of efficient cryptographic methods. The Datagram Transport Layer Security (DTLS) protocol can provide protection for the communication channel, while Constrained Application Protocol (CoAP) can ensure a secure interoperation between entities or systems

COMPaaS encapsulates a “low-level API of the device (device driver)” in order to define the devices “profile” for further connection with other entities. It also offers data preprocessing functions to the devices, as data aggregation. COMPaaS systems architecture is based on “Subscribe/Notify” and “Observer” communication pattern. Thus being able to trigger events every time that new data is produced, such as a context information.

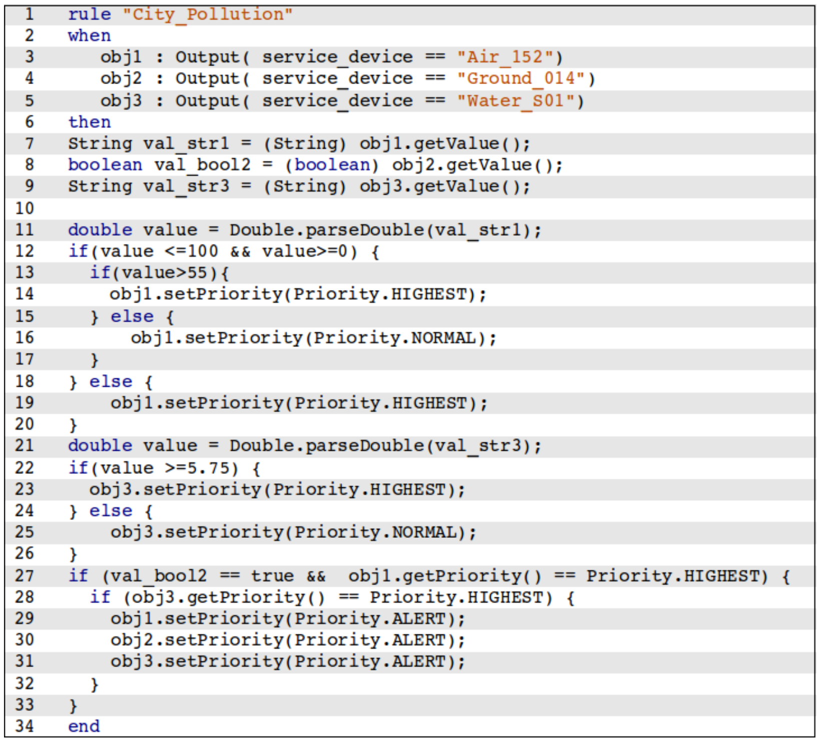
Although COMPaaS has many features in its architecture, it is not able to work according to the context in which it is inserted. In light of this, CONASYS context-aware functions were used by this work. The following modules presented at Figure 5.1 inherited CONASYS features: Context Production, Context Selection.

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) [40]. 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.

CONASYS implements the “Observer” pattern, so beyond the *query*, the user can also uses CONASYS by *subscription*. CONASYS receives the user request, by a specific communication API (through SOAP web service), understands it and creates a cycle to acquire the context information. As it have direct access to the devices connected to it (i.e., by COMPaaS), it is able to get device data and produce context.

CONASYS uses business rules to produce a context. It analyzes the environment data to create a context information depending on the environmental situation. CONASYS uses the Drools framework for the formulation of the rules. 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 5.3. 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.



```
rule "City_Pollution"
when
   obj1 : Output( service_device == "Air_152")
   obj2 : Output( service_device == "Ground_014")
   obj3 : Output( service_device == "Water_S01")
then
String val_str1 = (String) obj1.getValue();
boolean val_bool2 = (boolean) obj2.getValue();
String val_str3 = (String) obj3.getValue();

double value = Double.parseDouble(val_str1);
if(value <=100 && value>=0) {
  if(value>55){
    obj1.setPriority(Priority.HIGHEST);
  } else {
      obj1.setPriority(Priority.NORMAL);
  }
} else {
    obj1.setPriority(Priority.HIGHEST);
}
double value = Double.parseDouble(val_str3);
if(value >=5.75) {
  obj3.setPriority(Priority.HIGHEST);
} else {
    obj3.setPriority(Priority.NORMAL);
}
if (val_bool2 == true &&  obj1.getPriority() == Priority.HIGHEST) {
  if (obj3.getPriority() == Priority.HIGHEST) {
    obj1.setPriority(Priority.ALERT);
    obj2.setPriority(Priority.ALERT);
    obj3.setPriority(Priority.ALERT);
  }
}
end
```

Figure 5.3 – Example of a Drools rule.

5.2 CONTEXT SHARING MANAGER

The highlighted (i.e., red with dashed borders) modules present in Figure 5.4 represent the ones implemented by this work with the explicit functionality of providing context information sharing. The following modules represent the core of the sharing process: Context Sharing Manager (both Fog and Edge instances), Context Processing, Context Providers

Reasoner, and Context Providers Manager. The Context Storage layer and the Fog Repository module acts as important auxiliary functions to be detailed next under this Section, at Subsection 5.2.3.

The Context Sharing Manager module is the most crucial piece in the context sharing function. It interconnects with the other aforementioned modules responsible for the core of the sharing process. In light of this, the next paragraphs details the Context Sharing Manager functionalities and its relation with Context Processing, Context Providers Reasoner, and Context Providers Manager modules.

The Context Sharing Manager (ConShar) was developed to provide the context information sharing feature between different entities in IoT environments. The importance of having context interoperability is strongly related to the need of users and applications to share information between different sites (i.e., domains). There are two instances of ConShar on the proposed solution: one at the Fog layer, and other at the Edge layer.

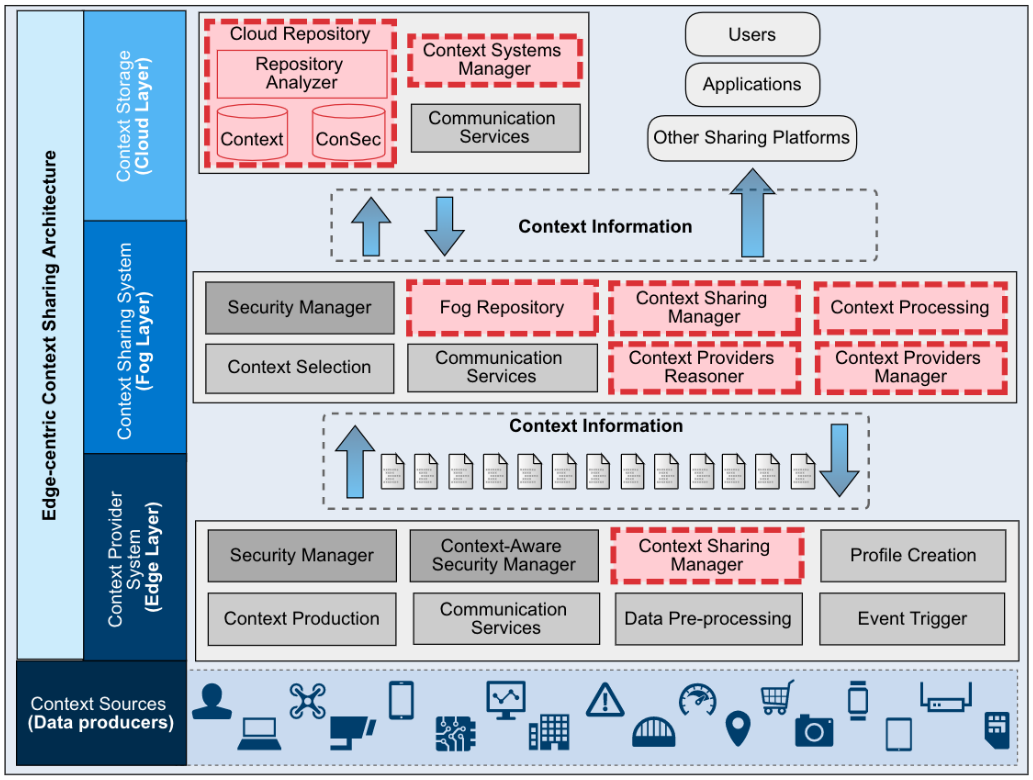


Figure 5.4 – Context Sharing modules of the Context Sharing Architecture.

The proposed ConShar takes benefit of Fog and Edge computing approaches to minimize network communications, thus reducing failure points and improving scalability [122][19]. Fog and Edge Computing are firmly related concepts, but they are not synonyms [107][94]. According to the OpenFog Reference Architecture [107], Fog computing extends

Cloud computing into an intermediate layer close to IoT devices and enables data processing across domains while Edge computing involves the control and management of a standalone endpoint device individually within the Fog domain.

The ConShar modules overview is presented in Figure 5.5, in a hypothetical scenario in which an IoT Entity of a specific domain shares context information with other IoT Entities in different domains. It is possible to divide the ConShar and the environment that it is inserted into two architectural layers: (i) Edge Layer, and (ii) Fog Layer. Next paragraphs explain the duties of each layer.

The Edge Layer comprises the IoT Entities that may use ConShar to share context information. Those entities can be considered either IoT physical devices (e.g., car, smart pacemaker, smartwatch) or IoT software running in any hardware (e.g., IoT devices, management system). Moreover, some ConShar modules may also make part of the Edge Layer, those modules are placed embedded or directly connected to IoT devices, and are responsible for the first round of context information processing and interpretation (i.e., Semantic Processing) and also for the distribution of the shared context to the entities that may have interest in receiving it (i.e., Distribution), at the end of the sharing processing.

Fog Layer comprises a heavier computational processing effort when compared with the Edge Layer. It is responsible for the reasoning part of the context sharing processing. First, it classifies, by the domain group, the interpreted context (i.e., Classification). Second, it notifies a Context Broker with the classified context (i.e., Repository). Thus, turning possible that the distribution process occurs by a specific domain.

To provide the context sharing as a service, the ConShar acts as a black box by the user (i.e., IoT Entity) side. The context information is shared between the domains, and the users/applications only receive the final output (i.e., shared context information).

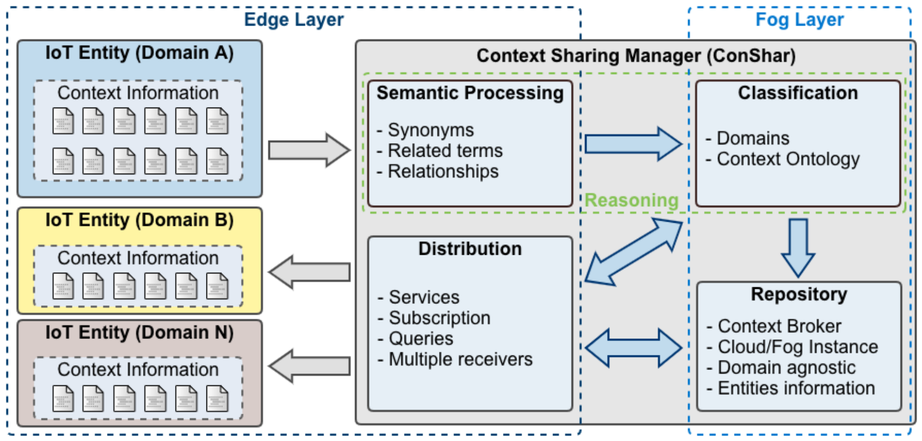


Figure 5.5 – Context Sharing Manager overview.

Let's consider the following scenario example for a better understanding of ConShar's applicability and how Fog and Edge entities relate themselves. It is possible to divide the IoT entities in two layers: Fog, and Edge. Each Fog entity is directly linked to some Edge entities under its domain. One Fog entity can be responsible for a Emergency Medical Services (EMS) and one Edge entity deployed on each ambulance of the city. One Fog entity can be placed at the urban traffic infrastructure and be responsible for different devices of this environment (e.g., traffic light, monitoring camera, public light) that are Edge entities. In a home-care or home-automation environment, a Fog entity can manage the IoT devices of the environment (e.g., smart door, smart light, smart-watch, health monitoring devices) that are Edge entities. The hospital infrastructure can hold a Fog entity and diverse hospital devices (e.g., multi-parameter patient monitor, hospital bed, patient information system) can hold an Edge entity.

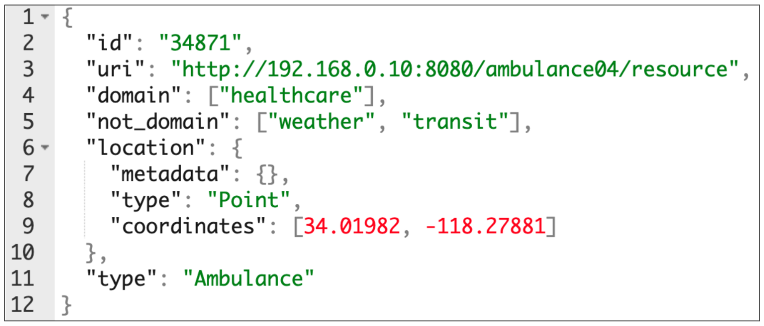
The network that constitutes the possible entities that are able to receive context information is created as the Fog and Edge entities share context, making it entirely decentralized. More than present the modules overview, this Section presents in details the context sharing process and its phases. This process is the core function of ConShar in order to reach the semantic interoperability. Next, the four main steps to reach the context sharing function are presented. Each step relates to a module presented in Figure 5.5.

5.2.1 Semantic Processing

The context sharing process starts on the Edge of the network, as it is responsible for the context information production by being directly connected to the IoT devices. The IoT Entity must send the context information to the ConShar module to trigger the context sharing process. ConShar offers an API for performing such a step. It was developed in Python programming language, so the IoT Entity can either import ConShar as a library or use its Web Service interface. The web address for the communication must be previously known by the IoT Entity.

It is a responsibility of ConShar to define what entity will receive the shared context information. The IoT Entity sharing context information does not need to determine the destination. This process is called Reasoning, and it is the more important step in context sharing [109][43]. In the proposed approach, the reasoning phase is divided into two modules: Semantic Processing, on the Edge Layer, and Classification, on the Fog Layer. Following this approach, it is possible to provide hybrid reasoning, with more substantial processing in the Fog Layer and a lighter processing effort in the Edge Layer.

There are two ways for an IoT Entity to receive shared context information. In the first way, the IoT Entity can share any context information, ConShar interprets its domain and put that entity in the list of entities that may have interest in context information of the same domain. In the second way, the IoT Entity can specify the domains that it wishes to receive context and the domains that it does not wish to receive context information. The second way depends on the IoT Entity to specify its domain, but it is a more accurate process. An example of a context information containing the domains specification can be seen in JSON format in Figure 5.6. The field *domain* presents the IoT Entity‘s domain that it wishes to receive context information. The field *not_domain* informs the domains that such IoT Entity does not wish to receive context information from.



```json
{
  "id": "34871",
  "uri": "http://192.168.0.10:8080/ambulance04/resource",
  "domain": ["healthcare"],
  "not_domain": ["weather", "transit"],
  "location": {
    "metadata": {},
    "type": "Point",
    "coordinates": [34.01982, -118.27881]
  },
  "type": "Ambulance"
}
```

Figure 5.6 – Context information of an IoT Entity with domain specification in JSON format.

```
from nltk.corpus import wordnet as wn

function extract_syn(word)
syns ← wn.synsets(word)
synonyms ← [ ]
for each syn in wn.synsets(syns[0].lemmas()[0].name() do
    for each l in syn.lemmas() do
        synonyms.append(l.name())
    end for
end for
set(synonyms)
return synonyms
end function

function extract_hyp(word)
syns ← wn.synsets(word)
return wn.synset(syns[0].name()).hypernyms()
end function
```

Algorithm 5.1 – Functions to gather the synonyms and hypernyms of a word.

The Semantic Processing is responsible for extracting the semantic values of the properties presented in the context information (i.e., JSON file). It will ignore the properties *domain* and *not_domain*, as they are used in a different processing to be detailed next, in

Subsection 5.2.3. Considering the Figure 5.6, the values extracted would be: *Point*, *Ambulance*. This process was made with the *JSON* library for Python[1].

From each one of the extracted values, the proposed approach gathers its synonyms (i.e., words having the same meaning) and hypernyms (i.e., more abstract terms of a word). The WordNet tool is used to discover the synonyms and hypernyms of a word. WordNet is a large lexical database of English [111]. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms. The Algorithm 5.1 shows the developed functions using WordNet, Pyhton and the NLTK library [17]. Finally, it is created a list with all the synonyms and hypernyms of the word and send it to the Classification module.

5.2.2 Classification

This step begins with the list with all the synonyms and hypernyms of a word extracted from the context information. ConShar performs the Classification process to associate the shared context information with an IoT domain. The Classification makes possible to infer with whom the context information must be shared.

A similarity method was used to compare the words extracted from the context formation and its synonyms and hypernyms with different IoT domains. The IoT domains are extracted from a context ontology defined in an author previous work [41]. The defined ontology (see Figure 5.7) is composed of three main subclasses: (i) *Context_Domain* subclass defines which domain the context came from (e.g., health, urban traffic, industry); (ii) *Context_Format* should be used to represents the context format (e.g., semantic, numeric); and (iii) *Context_Source* subclass denotes who is the context information source (e.g., user, network). The defined context ontology can also be composed or linked to another already developed ontologies for specific domains. The number of third-party ontologies used varies depending on how constrained is the environment in which ConShar is deployed. With a more detailed domain ontology, more similarity checks are made with the context information words. Technically, the developed ontology is called as an upper-level ontology and the domain-specific as domain ontologies.

Different domain-specific ontologies can be manually inserted into the Context Sharing Manager for a specific domain processing. It is necessary to inform during the development phase the URL address for the ontology specification. The domain-specific ontologies used in this work were mostly selected based on the lists available in the World Wide Web Consortium (W3C)[2]. Some examples of well-known ontologies used in this work for different domains are:

- Sanya et al. [120] ontology for Aerospace Context;

[1]https://docs.python.org/3/library/json.html
[2]https://www.w3.org/

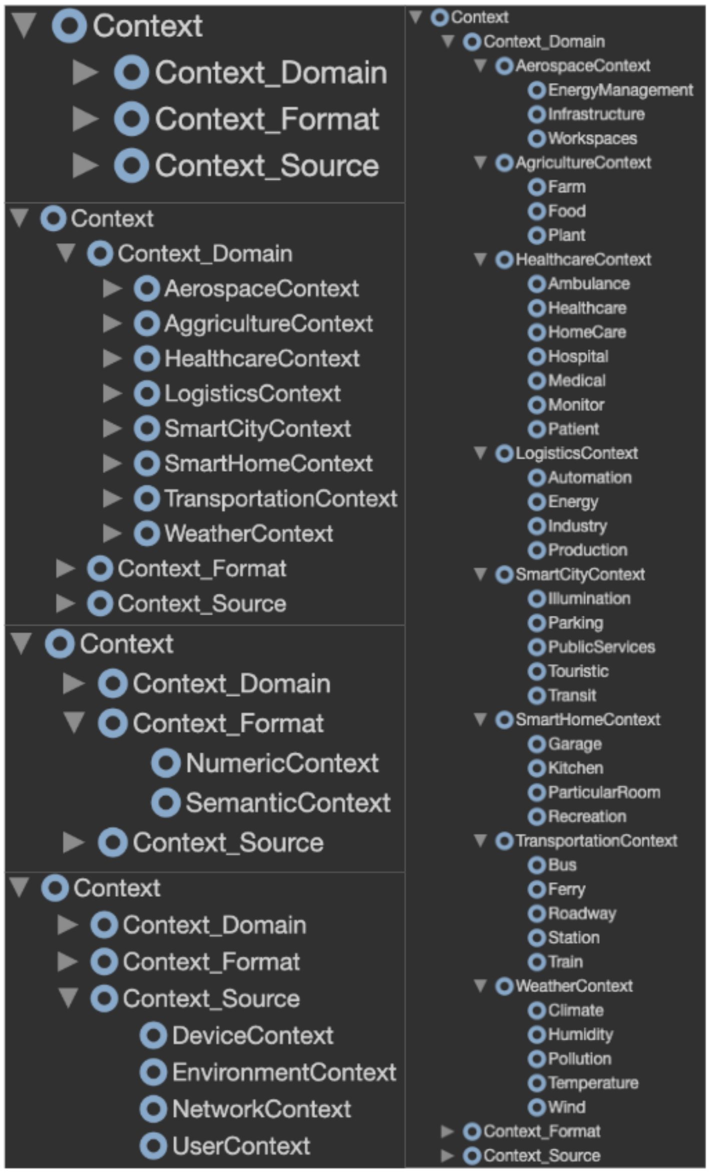


Figure 5.7 – Context ontology overview.

- Global Agricultural Concept Space (GACS) [12] for Agriculture Context;
- U-healthcare [75] for Healthcare Context;
- GenCLOn [9] for Logistics Context;
- Km4City [14] for Smart City Context;
- Tao et al. [134] ontology for Smart Home Context;
- Houda et al. [67] ontology for Transportation Context;

- WeatherOntology [129] for Weather Context.

There are different methods for similarity check [23]. This work uses the Leacock-Chodorow Similarity [80], that return a score denoting how similar two word senses are, based on the shortest path that connects the senses on the WordNet classification words tree. Leacock-Chodorow Similarity method is a robust and accurate method for similarity check [23][127]. Algorithm 5.2 shows the function that makes the similarity check. It receives two lists of words. The *word_list* represent the data about the context information. Each word extracted from the context has its own *word_list* with its synonyms and hypernyms. The first item of the list (i.e., index 0) is the original word from the context information. The *domain_list* comprises the words extracted from the context ontology, through Owlready2 Python library[3]. The first item of the *domain_list* is the most generic definition of a domain (e.g., healthcare), followed by more specific terms (e.g., ambulance, hospital, patient).

```
from nltk.corpus import wordnet as wn

function check_similarity(word_list, domain_list)
score ← [ ]
for each i in range(len(word_list) do
    for each j in range(len(domain_list) do
        if i == 0 and j == 0 then
            score.append(word_list[i].lch_similarity(domain_list[j]))
        else
            score.append(word_list[i].lch_similarity(domain_list[j])*75/100)
        end if
    end for
end for
score.sort()
synonyms ← [ ]
return score[-1]
end function
```

Algorithm 5.2 – Function to check the similarity between words.

The process to verify the similarity of a context information is made with different domains, running the *check_similarity* function for each domain present on the context ontology. Each run returns a *score* number. The bigger the score, the more similarity the context has with the compared domain. It is considered 100% of the *score* in the first similarity check, that comprises the actual word extracted from the context information and the most generic definition of the domain (i.e., index 0,0 on *if* clause on Algorithm 5.2). For the other similarity checks, it is considered only 75% of the *score* value, as it starts to compare the synonyms and hypernyms with more specific domain words.

[3]https://pypi.org/project/Owlready2/

5.2.3 Repository

After verify the domain that has more relation with the context information, this domain is added as a property to the context file (i.e., JSON, XML). With this classification done, it starts the repository process, in which it makes use of the FIWARE Orion Context Broker[4] system. A Python project was developed to facilitate the usage of the FIWARE Orion Context Broker. The source code was uploaded to GitHub[5]. It has functions for different types of upload and queries for context information.

The FIWARE Orion Context Broker acts as a pool of context. It is possible to upload the context information to its infrastructure, which can either be local or in a Cloud/Fog instance. In the present project design, it is adopted the view of a repository that could be accessed by any IoT system through the web. FIWARE Orion Context Broker makes it possible to the IoT system to query for context, or even subscribe to context information, receiving the updates related to that context. It also has options to manage the context by any property. In the proposed approach, considering the steps presented previously, a domain of interest for each context information is set. In light of this, it is possible to have heterogeneous context information of different domain in the same FIWARE Orion Context Broker infrastructure with a easy management policy controlled by FIWARE system.

Taking into account the scope of the proposed work, the implemented FIWARE Orion Context Broker infrastructure has two levels: one at the Fog layer and the other at the Cloud layer. The one at the Fog only stores context information about its domain. The one at the Cloud layer (i.e., Context Storage) stores context information about the different domains present in the hole system.

The FIWARE Orion Context Broker also provides some features when getting context information from its repository, like filtering or aggregation (i.e., Context Processing module). Moreover, as it maintains a record of each shared context information, it is possible to manage and query for the entities that had produced context information (i.e., Context Providers Reasoner, and Context Providers Manager modules).

It is possible to use FIWARE Orion Context Broker by downloading it package and running locally (e.g., in the Fog device) or by using the Docker[6] virtualization technology. Implementation details. In the proposed approach it is possible to visualize both implementation, as the Context Storage layer at the Cloud is related to the Docker virtualization approach. Next paragraphs shows the source code of FIWARE Orion Context Broker integration implementation used by this work.

[4]https://fiware-orion.readthedocs.io/en/master/
[5]https://github.com/evermatos/orion_integration
[6]https://www.docker.com/

The implemented methods make possible the insertion and search for context information. The Algorithm 5.3 shows the functions responsible for inserting the context information (i.e., an entity) in the FIWARE Orion Context Broker instance. The address will vary depending on the implementation instance (i.e., Fog or Cloud).

```
import requests

function put_entity(entity)
url ← 'http://'+address+':1026/v2/entities/'
response ← requests.post(url, json=entity)
if response.status_code == 201 then
    return 'Entity inserted'
else
    return 'Action failed'
end if
end function

function update_specific_attribute(entity_id, entity_attribute, entity_context)
url ← 'http://'+address+':1026/v2/entities/'+entity_id+'/attrs/'+entity_attribute+'/value'
response ← requests.put(url,data=entity_context,headers='Content-Type':'text/plain')
if response.status_code == 204 then
    return 'Context updated'
else
    return 'Action failed'
end if
end function
```

Algorithm 5.3 – Functions to insert and modify context information.

The Algorithm 5.4 shows the query options to get context information from the FIWARE Orion Context Broker infrastructure.

The FIWARE Orion Context Broker infrastructure also makes possible the subscription for a specific context. Every time that such context is updated at FIWARE Orion Context Broker infrastructure, it will be sent to the entities that have subscribed to it. It is necessary a JSON script informing the details for the subscription, such as the conditions to receive an update, the URL for notification, and the subscription expiration date. Algorithm 5.5 shows the subscription options to get context information from the FIWARE Orion Context Broker infrastructure.

Any time that the FIWARE Orion Context Broker infrastructure receives context information, the Distribution module is triggered.

```
import requests

function get_entities()
url ← 'http://'+address+':1026/v2/entities?options=keyValues'
response ← requests.get(url)
if response.status_code == 200 then
    return response.json()
else
    return 'Query failed'
end if
end function

list_of_filters ← [["type", "=Store"],
                   ["georel", "=near;maxDistance:1500"],
                   ["geometry", "=point"],
                   ["coords", "=52.5162,13.3777"]]

function get_entities_filter(list_of_filters)
url ← 'http://'+address+':1026/v2/entities?options=keyValues'
for each x in list_of_filters do
    url_aux ← '&' + x[0] + x[1]
    url ← url + url_aux
end for
response ← requests.get(url)
if response.status_code == 200 then
    return response.json()
else
    return 'Query failed'
end if
end function
```

Algorithm 5.4 – Functions to query for context information.

5.2.4 Distribution

The Distribution module has the main function of sending context information to the destination entities through the network. As it is notified by the Repository module when a new context information is available, it shares that context to the entities of context's domain, excluding the one that produced that context information.

It is a very straightforward task. Once each entity has its *URI (Uniform Resource Identifier)* address, the ConShar can send the context information by this address in different ways, as Web Service, socket, WebSocket. It uses the communication channels defined by COMPaaS and CONASYS [8][41].

```
import requests, json
from sys import argv

subscription_json ← """{
        "description": "A subscription to get info about Room1",
        "subject": {
            "entities": [
                {"id": """+'"'+argv[1]+'"'+""",
                "type": """+'"'+argv[2]+'"'+"""}
            ],
            "condition": {
                "attrs": ["""+'"'+argv[3]+'"'+"""]
            }
        },
        "notification": {
            "http": {"url": """+'"'+argv[4]+'"'+"""},
            "attrs": ["""+'"'+argv[3]+'"'+"""]
        },
        "expires": "2040-01-01T14:00:00.00Z",
        "throttling": 5
    }"""
detailed_subs ← json.loads(subscription_json)

function new_subscription(detailed_subs)
url ← 'http://'+address+':1026/v2/subscriptions'
response ← requests.post(url, json=detailed_subs)
if response.status_code == 201 then
  return 'Subscription done. Subs ID: '+response.headers['Location']
else
  return 'Subscription failed'
end if
end function
```

Algorithm 5.5 – Function to subscribe for a context information.

5.3 CONTEXT-AWARE SECURITY IMPLEMENTATION

The highlighted (i.e., red with dashed borders) module present in Figure 5.8 represent the one implemented by this work with the explicit functionality of providing context-aware security functionality to the Context Sharing Architecture using shared context information.

It is necessary a context information to infer a security decision. The context information can be acquired in two ways: (i) from the Fog Repository module at the Fog layer, and (ii) from the Context Sources (i.e., IoT devices, data producers) directly from the Edge layer. In most cases, the context information used for the security decisions came from the

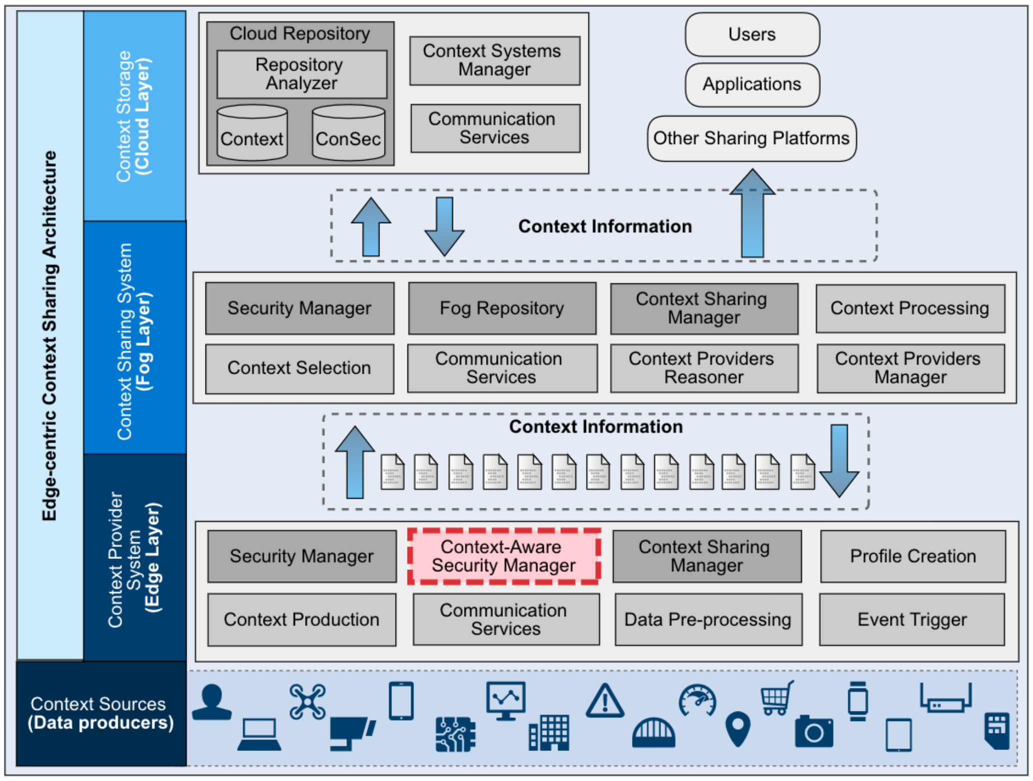


Figure 5.8 – Context-aware security module of the Context Sharing Architecture.

Fog Repository, as it already passed by the Classification phase (see Section 5.2) and have a domain related to it, improving the detail level of such context information.

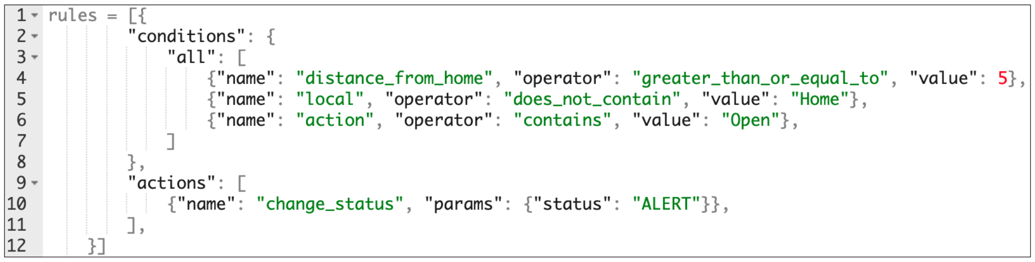
The Context-Aware Security functionality is implemented by the Context-Aware Security Manager module, detailed in Section 4.2. This module has the main responsibility of reason over context information to provide security decisions. It is a module deployed at the Edge layer of the Context Sharing Architecture. Thus, it should have a lightweight processing effort, one time that in most cases, the IoT entities at the Edge layer had less computational power processing than the ones of the Fog layer.

This work uses business rules for the context-aware security provision. It is considered a lightweight technique, which is ideal for such kind of IoT environment [109]. Many business rules libraries in the literature can be used for such type of processing. In this work, the *venmo/business-rules*[7] library for Pyhton was chosen to provide the reasoning functions for a proper context-aware security provision. The *venmo/business-rules* is an open source library with recently used in Internet of Things scenarios [73][44]. Moreover, the rules can be defined using the JSON format, which is already used by the entire Context Sharing Architecture for defining context information, making it entirely interoperable.

[7]https://github.com/venmo/business-rules

The implementation of context-aware security by rules allows the provision of security decisions in different areas: (i) authentication, (ii) authorization, (iii) access control, and (iv) privacy-preserving. The rules are IF-THEN-ELSE structures and make the provision of security decisions flexible. It depends on the defined rules set the kind of security provision that will be offered. For example, on the access control area, different methods can be provided, such as Role-based Access Control (RBAC), Attribute-based Access Control (ABAC), and Rule-based Access Control (RAC). It depends on which field (i.e., data, category) will be used for the rule analysis. For authentication and authorization, a set of rules can establish the level of access and/or operation that each unique entity will have [62]. Also, the privacy-preserving rules will infer decisions by the relationship between context information data, thus providing different levels of abstraction for the data. There are specific techniques for each one of the four security provision areas aforementioned. However, rules can provide a generic framework that can cover the four areas. Mainly, the action performed by the rule (i.e., THEN) that infers which security service will be provided.

Figure 5.9 shows an example of a rule used by the *venmo/business-rules* library. It is possible to define conditions and actions for the rules. The *actions* only are fired if a context information meets all the *conditions*.



```
rules = [{
        "conditions": {
            "all": [
                {"name": "distance_from_home", "operator": "greater_than_or_equal_to", "value": 5},
                {"name": "local", "operator": "does_not_contain", "value": "Home"},
                {"name": "action", "operator": "contains", "value": "Open"},
            ]
        },
        "actions": [
            {"name": "change_status", "params": {"status": "ALERT"}},
        ],
    }]
```

Figure 5.9 – Context-aware security rule.

The *conditions* uses compares a field from the context information with a data predefined at the rule. The field appears on the rule as *“name”*. It is associated with a *“value”* from the rule. Different *“operators”* can be used for this comparison. The *venmo/business-rules* library offers the following possibilities for the operator:

- **numeric** - an integer, float, or python Decimal:
  - equal_to
  - greater_than
  - less_than
  - greater_than_or_equal_to
  - less_than_or_equal_to
- **string** - a python bytestring or unicode string:

  - equal_to
  - starts_with
  - ends_with
  - contains
  - matches_regex
  - non_empty

- **boolean** - a True or False value:
  - is_true
  - is_false

- **select** - a set of values, where the threshold will be a single item:
  - contains
  - does_not_contain

- **select_multiple** - a set of values, where the threshold will be a set of items:
  - contains_all
  - is_contained_by
  - shares_at_least_one_element_with
  - shares_exactly_one_element_with
  - shares_no_elements_with

The *"actions"* of a rule can trigger a pre-defined function in the source code or simply modify a parameter in the context information, as shown in Figure 5.9.

The *venmo/business-rules* library uses the concept of *variables* to represent the data that appears in the *"name"* field of a rule (see Figure 5.9). Variables represent values the systems, usually the value of some particular object. The rules are created by setting threshold conditions such that when a variable is computed that triggers the condition some action is taken. It is possible to define all the available variables for a certain kind of object in the source code, and then later dynamically set the conditions and thresholds for those. Algorithm 5.6 shows an example of the variables representation related to the rules presented in Figure 5.9.

The *venmo/business-rules* library uses the concept of *actions* to determine the process available to be taken when a condition of a rule is triggered. Algorithm 5.7 shows some *actions* examples related to the rule presented at Figure 5.9. The decorator of each function will vary depending on the variable that an action modifies/access:

```
from business_rules.variables import BaseVariables, numeric_rule_variable,
string_rule_variable, select_rule_variable

class ContextVariables(BaseVariables):
    def __init__(self, context):
        self.context ← context

    @numeric_rule_variable
    def distance_from_home(self):
        return self.context.distance_from_home

    @select_rule_variable()
    def local(self):
        return self.context.local

    @string_rule_variable()
    def action(self):
        return self.context.action
```

Algorithm 5.6 – Variables implementation in context-aware security provision.

- **numeric**: @numeric_rule_variable
- **string**: @string_rule_variable
- **boolean**: @boolean_rule_variable
- **select**: @select_rule_variable
- **select_multiple**: @select_multiple_rule_variable

```
from business_rules.actions import BaseActions, rule_action
from business_rules.fields import FIELD_TEXT

class ContextActions(BaseActions):
    def __init__(self, context):
        self.context ← context

    @rule_action(params ← "status": FIELD_TEXT)
    def change_status(self, status):
        self.context.status ← status
        self.context.save()
```

Algorithm 5.7 – Actions implementation in context-aware security provision.

Every time that a new context is received (i.e., from a sharing event), the whole process of providing context-aware security is executed. This situation happens because the Context Sharing Architecture does not know exactly which context information is coming and if it has some security rules linked to it.

The Algorithm 5.8 shows the directives responsible for running the rules defined for the environment. In the example presented in the Algorithm 5.8, three context information were simulated with the properties defined by the *variables* aforementioned in Algorithm 5.6. The rules of this environment (see Figure 5.9) determine that if the *distance_from_home* is greater or equal to *5*, the *local* is different from *home*, and the *action* contains the work *open*, an *alert* is set to such context information. In this case, the *Context2* receive an *alert*. This example relates to the traditional context-aware security scenario presented in Section 2.5.

```
from business_rules import run_all

contexts ← [
        Context(name ← "Context1", status ← "ACTIVE", distance_from_home ← 4,
                local ← "Home", action ← "Open"),
        Context(name ← "Context2", status ← "ACTIVE", distance_from_home ← 5,
                local ← "Office", action ← "Open"),
        Context(name ← "Context3", status ← "ACTIVE", distance_from_home ← 10,
                local ← "Work", action ← "None"),
        ]

for each context in contexts do
  run_all(rule_list ← rules,
          defined_variables ← ContextVariables(context),
          defined_actions ← ContextActions(context),
          stop_on_first_trigger ← True)
end for
```

Algorithm 5.8 – Function responsible to run the rules.

6. EXPERIMENT RESULTS

This chapter presents the evaluation tests performed in proposed Context Sharing Architecture implementation. The main goal of the tests is to evaluate the Context Sharing Architecture in terms of performance and usability. The tests were divided in sections. Each section represents a test scenario with different objectives and methodology.

6.1 EXPERIMENT 1 - CONTEXT SHARING ARCHITECTURE PERFORMANCE

A smart city is a complex IoT environment for encompasses many different application verticals (e.g., healthcare, home-care, urban traffic, EMS) interacting with each other to provide efficient services to the population. Let's consider the scenario previously discussed in Section 4.3 and seen in Figure 4.8 in which the focus is on sharing the context of a home-care patient when some important events related to the health condition occurs. In this scenario, there are four deployed instances of the Context Sharing System: (i) home-care and home-automation, (ii) EMS, (iii) hospital infrastructure, and (iv) urban traffic infrastructure. For every instance of the Context Sharing System it will be various instances of the Context Provider System. For example, the Context Sharing System instance of the home-care and home-automation will have one instance of the Context Provider system connected on it for each device of the house (i.e., automatic door, monitoring camera, etc.). In the EMS Context Sharing System, every ambulance has a Context Provider System.

The context information acquired by home-care sensors triggers the event of a patient having a heart attack. The Fog layers (i.e., Context Sharing System) instances of the architecture are responsible for sharing context information with other fogs. This sharing occurs three times in this application scenario: (i) the patient context is shared with EMS, and hospital infrastructures, (ii) hospital context is shared with the EMS, and (iii) EMS context is shared with home-automation and urban traffic infrastructure.

The received context information can be used in new processing. In this scenario, the ambulance context is shared with the urban traffic infrastructure, thus the city can adapts itself by creating routes to drain the traffic with a "green wave" in traffic lights. Finally, the context information of the ambulance arriving at the patient's home is used to open the door to facilitate paramedics access.

6.1.1 Environment Setup

The main goal is to demonstrate the Context Sharing Architecture suitability in different networks, which is very common in IoT environments. In order to evaluate that, it was measured the time taken for the context information from the source (i.e., Context Provider) to the destiny (i.e., another Context Provider from other domain) passing through two Context Sharing Systems (i.e., one for each Context Provider). In this sense, it has four entities and three communications (i.e., Context Provider A $\rightarrow$ Context Sharing A $\rightarrow$ Context Sharing B $\rightarrow$ Context Provider B). For the tests, it was considered three subsets of the previous IoT scenario and each one has its peculiarities. Figure 6.1 shows the three subsets defined for this evaluation and its peculiarities.

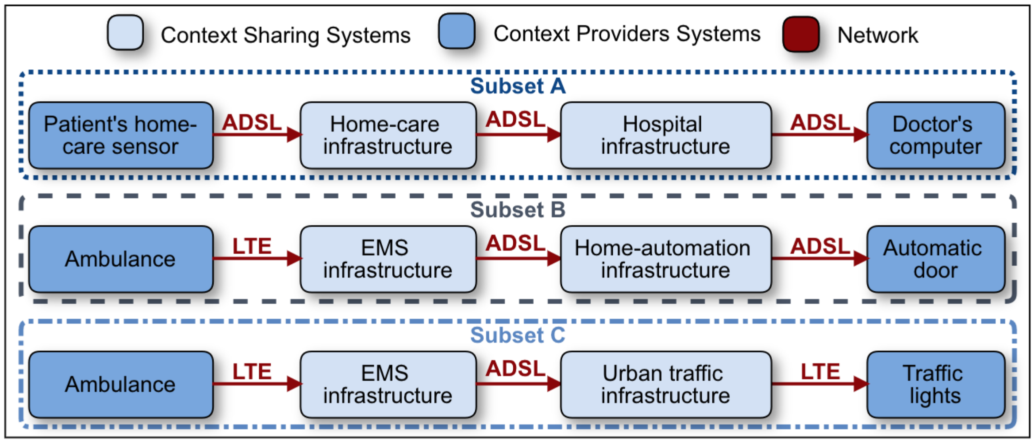


Figure 6.1 – Three possible context sharing conditions.

Subset A: The noise sensor of the patient house detects a suspect event and shares its context with the home-care system (i.e., Context Provider to Context Sharing). The context information is shared with the hospital infrastructure (i.e., Context Sharing to Context Sharing). Finally, the context is shared with doctor computer (i.e., Context Sharing to Context Provider). In this subset, all the communications are doing by ADSL infrastructure.

Subset B: The ambulance context (e.g., location) is shared with the EMS infrastructure. This context is shared with the home-automation infrastructure. Finally, the context information is shared with the automatic door. In this subset, the first communication is doing by LTE while the other two by ADSL.

Subset C: Ambulance context information is shared with the EMS infrastructure. The context is shared with the urban traffic infrastructure of the city. Finally, the context is shared with the traffic lights to produce a “green wave”. This subset has two communications by LTE and only the EMS-to-urban traffic communication doing by ADSL.

A heterogeneous infrastructure was used to perform the tests. The Context Sharing Systems were hosted by Dell All-in-one computers. Both were configured with Ubuntu 16.04 LTS (64-bit), Quad-Core 2.8 GHz and 8GB of RAM. When using LTE, the Context Providers Systems were hosted by a cell phone configured with Android 7.0, Octa-Core: 2.1 GHz Quad-Core and 1.5 GHz Quad-Core, 3GB of RAM. When using ADSL, the Context Providers Systems were hosted by a Raspberry Pi 3 Model B configured with Raspbian, Quad Core 1.2 GHz and 1GB of RAM.

6.1.2 Experiment Results - Context Information Path

With the previously defined scenario in mind, we performed two tests. The first one is related to the *Subset A*, in which the size of context information is variable. The second one presents the comparison of the execution time of *Subset A*, *Subset B*, and *Subset C* for different context information sizes.

In the first test, the time taken by the context information to go from one Context Provider to another in a different domain was measured. Each context information is approximately the size of 250 bytes (see Figure 2.4 for a context information example). Taking into account the *Subset A*, the architecture shares different amounts of context information simultaneously. The simulation starts from 250 bytes (one context information) to 12500 bytes (fifty context information), running the test 30 times for each context information size. Figure 6.2 shows the results (average of 30 executions) for the simulation of *Subset A*.

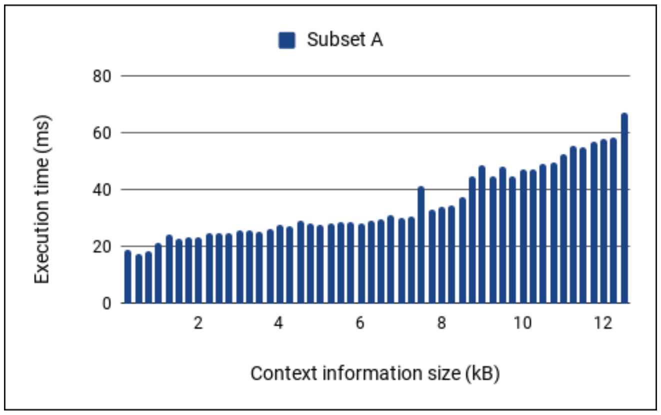


Figure 6.2 – Execution time (ms) of *Subset A* varying context information size.

In the second test, it is presented the comparison of how different networks deal with different context information sizes. The communications of *Subset A*, *Subset B*, and *Subset C* were simulated considering different context information size. The simulation begins from 250 bytes (one context information) to 1250 bytes (five context information).

Figure 6.3 presents the average execution time (i.e., time taken of the whole communication) considering 30 executions for each subset tested with different networks (in milliseconds) and different context information sizes. It was observed that the time taken by the architecture for share context information grows exponentially when LTE network is used. This is expected since mobile networks, as LTE, usually has bigger latency and packet loss.

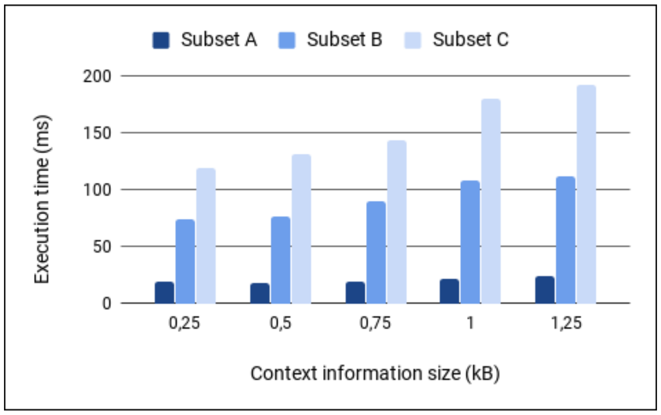


Figure 6.3 – Execution time in different networks (ms).

The results obtained for execution time were acceptable, mainly because the time taken for sharing context information between different domains was 192 ms (*Subset C*) for the worst case, and the best results are smaller than 20 ms (observed in the full-ADSL network - *Subset A*). Figure 6.2 shows that the *Subset A* has a high scalability once it vary only a few milliseconds even in a non-realist scenario in which a context information 50-times bigger than the exemplified one is shared (see Figure 2.3). In most cases, the context information size is up to five times 250 bytes. Figure 6.3 shows that even for the worst analyzed scenario (*Subset C*), the communication time for sharing context is 153 ms on average. The LTE network has presented an average of 95 ms on standard deviation, which was 28 times bigger than the ADSL.

The edge-centric approach helps in the positive result, once the context information is produced in the edge of the network, avoiding the transportation of a large amount of data. Even in the worst case, the communication time for the proposed architecture to share context information over the Internet in less than 0,5 s for each of the 30 times execution.

6.1.3 Experiment Results - Semantic Processing Phase

The *Semantic Processing* is one of the most important process in the Context Sharing Architecture. It represents one of the reasoning phases (see Section 5.2). The *Subset*

A was considered for this experiment in which a Context Provider can share a context information to the Context Sharing by ADSL network (e.g., home care device), that also uses ADSL network to share the context information with the destination. Two different scenarios were analyzed: (i) considering the network delay, and (ii) considering only the *Semantic Processing*.

This experiment focuses on evaluating the suitability of the Context Sharing Architecture *Semantic Processing* phase (see Section 5.2) to IoT resource-constrained environments. This processing is related to the search for synonyms of a specific word through the WordNet base. The number of synonyms may vary depending on the word.

The last available version of WordNet tool[1] was used for the tests. The implementation was made with Java by the JWI WordNet library [53]. It was measured the time taken for the extraction of the *domain* from a context information, the search for its domain synonyms in the WordNet base, and the attribution of a destination to a context information. The ontology classification time was excluded from the test because it may vary depending on the deployed domain.

For the first scenario (i), we considered the *Semantic Processing* time plus the network delay of a contex information going from one Context Provider to another in a different domain. The size of each context information is 250 bytes (see Figure 2.4 for a context information example). The simulation consists of the architecture sharing different amounts of context information simultaneously. Thus, the *Semantic Processing* phase must perform the same process simultaneously in an acceptable time frame being scalable to deal with complex IoT environments. The simulation starts from 250 bytes (one context information) to 12500 bytes (fifty context information). It was used context information from different domains. Figure 6.4 shows the results (average of 30 executions for each context information size).

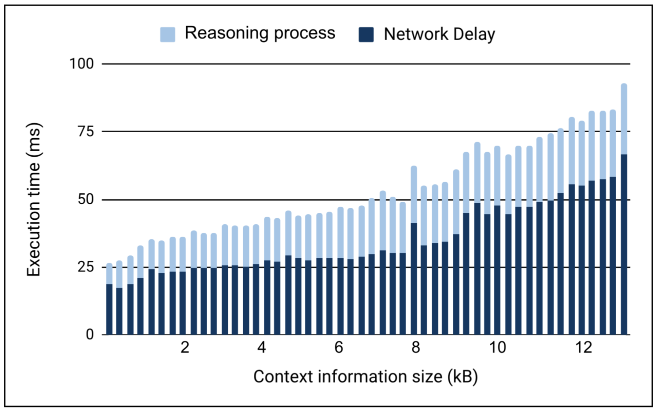


Figure 6.4 – Execution time (ms) of the Semantic Processing varying context information size.

[1]https://wordnet.princeton.edu/download/current-version

Figure 6.4 shows that the network delay takes longer time than the *Semantic Processing*. Considering the results, network delay takes an average of 64.5% of the execution time and the *Semantic Processing* only 35.5%. This result is considered satisfactory for ADSL network communications, as it has a lower latency when compared to cellular networks. In other words, the *Semantic Processing* execution time will express less than 35.5% of the total time in communications using cellular networks. The total time may increase when using LTE networks, that are common in IoT environments, however the *Semantic Processing* execution time will still be the same.

For the second scenario (ii), we considered the Context Sharing system effort in simultaneous *Semantic Processing*, without including the network delay. The simulation starts from 12500 bytes (fifty context information) to 2500000 (ten thousand context information). The context information used in this test is similar to the one used for the previous first scenario, but on a larger scale (i.e., more quantity). Table 6.1 shows the results (average of 30 executions for each context information size).

Table 6.1 – Context Sharing Architecture Semantic Processing process results.

Feature/Context amount	50	500	1000	5000	10000
Context info. size (kb)	12.5	125	250	1250	2500
Execution time (ms)	26.2	73.1	115.9	416.9	809.4

Table 6.1 shows that *Semantic Processing* is suitable for Internet of Things environments. In the worst tested scenario, even with ten thousand context information being processed simultaneously, the entire processing takes less than one second (809.4 ms). The results show that the *Semantic Processing* can be applied even at the Edge layer (Context Provider). Moreover, the entire WordNet base has the size of 55.2 MB, which fits in most traditional IoT devices like smartphones and Raspberry Pi 3 boards.

6.2 EXPERIMENT 2 - CONTEXT SHARING ARCHITECTURE ACCURACY

This Section evaluates the *Classification* phase of the Context Sharing Manager (see Section 5.2). The goal is to test the Context Sharing Architecture regarding its accuracy in classifying the context information according to the Internet of Things domains. Moreover, the overall performance (i.e., time taken for execution) of the *Classification* phase also is analyzed.

6.2.1 Environment Setup

The *Classification* phase uses the upper-layer ontology and the domain-specific ontologies to classify the context information with a specific domain. The upper-layer ontology is represented by the Context ontology developed by this work (see Figure 5.7). The domain-specific ontology may vary depending on how constrained is the application scenario. On the one hand, the upper-layer ontology sets the possible domains that the context information can be classified. On the other hand, the domain-specific ontologies give more options for the classification. For example, on the *Transportation* domain, the upper-layer ontology presents the possible categories for classification: Bus, Ferry, Roadway, Station, Train. In the same domain, the Houda et al. [67] ontology for transportation Context provides another specific categories for classification: Transport line, Stop point, Journey path, Railway. Figure 6.5 shows the aforementioned *Transportation* domain example.

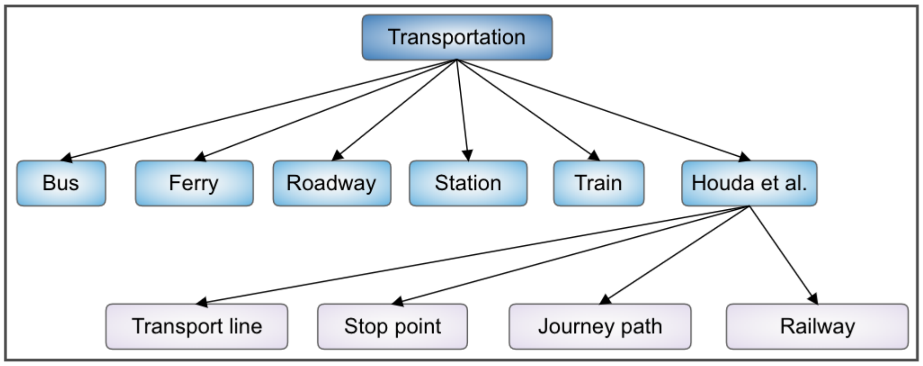


Figure 6.5 – Transportation domain upper-layer ontology and domain-specific categories.

For this experiment, we classified 10 (ten) context information from different domains with the 8 (eight) possible domains presented in the developed Context ontology (see Figure 5.7). The Leacock-Chodorow Similarity method was used to compare each context information and its synonyms and hypernyms with each domain (i.e., categories from the upper-layer ontology) and its variations (i.e., categories from the specific-domain ontology). The Leacock-Chodorow Similarity method returns a score denoting how similar two word senses are, based on the shortest path that connects the senses. This method uses the WordNet library to correlate the compared words. In this experiment, 10 (ten) scores are produced for each context information. The greater score represents the bigger similarity with the context and the domain, inferring that such context should be of interest to such domain. In some cases, more than one domain may receive the greater score, in this case all the tied domains will be notified with the context information.

Table 6.2 – Main data extracted from context information, its synonyms and hypernyms.

Main data	Synonyms	Hypernyms
Ambulance	—	wagon, minivan
Pacemaker	cardiac pacemaker	cardiac muscle, heart muscle
Heart attack	—	heart failure, coronary failure
Rain	rainfall	precipitation, downfall, rain shower
Car	auto, automobile, motorcar	motor vehicle, automotive vehicle
Car accident	—	stroke, fortuity, chance event
Door	room access	entrance, entryway, entry
Fleet	—	collection, aggregation, accumulation
Route	path, itinerary	line, way
Traffic	—	aggregation, pedestrians, vehicles

As the *Classification* phase occurs at the Fog layer, only the Fog layer is used in this experiment. It was hosted by a MacBook Pro configured with macOS 10.15.2, Six-Core 2.2 GHz and 16GB of RAM.

6.2.2 Experiment Results

For this experiment, we calculated the *score* of the *Classification* phase between some examples of context information and the different domains present in IoT environments. It is considered as context information for this experiment only the main data from each regular context information (i.e., JSON format). The main data refers to the semantic information extracted from the original context information at the *Semantic Processing* method, detailed in Section 5.2. All the synonyms and hypernyms of the main data were used for the *Classification* process. Those information are gathered using the WordNet library. Table 6.2 shows the main data its synonyms and hypernyms used in this experiment, a dash (—) symbol is used across all columns to denote that the WordNet consult returned no terms.

Table 6.3 shows the expected classification results for each main data related with the following domains: healthcare, agriculture, smart home, weather, logistics, aerospace, smart city, and transportation. The (✓) symbol represents the expected domains for each main data. Taking into account that a context information can be of interest of more than one domain. For each domain, the aforementioned domain-specific ontologies (see Section 5.2) were used to extract the categories to be compared with the terms shown in Table 6.2. Taking into account the example shown in Figure 6.5, the terms were compared with the following categories: transportation, bus, ferry, roadway, station, train, transport line, stop point, journey path, and railway. For example, for the main data *Ambulance* and its synonyms and hypernyms, the *Classification* process runs the Leacock-Chodorow Similarity

Table 6.3 – Expected classification domain for each main data extracted from context information.

Main data	Domains							
	Healthcare	Agriculture	Smart Home	Weather	Logistics	Aerospace	Smart City	Transportation
Ambulance	✓	—	—	—	—	—	—	—
Pacemaker	✓	—	—	—	—	—	—	—
Heart attack	✓	—	—	—	—	—	—	—
Rain	—	✓	—	✓	—	—	—	—
Car	—	—	—	—	—	—	—	✓
Car accident	—	—	—	—	—	—	✓	✓
Door	—	—	✓	—	—	—	—	—
Fleet	—	—	—	—	✓	—	—	—
Route	—	—	—	—	—	—	✓	✓
Traffic	—	—	—	—	—	—	✓	✓

Table 6.4 – Context Sharing Architecture Classification score results.

Main data	Domains							
	Healthcare	Agriculture	Smart Home	Weather	Logistics	Aerospace	Smart City	Transportation
Ambulance	0.998	0.641	0.864	0.804	0.693	0.693	0.864	**1.335**
Pacemaker	1.072	0.864	**1.168**	0.864	0.641	0.864	0.693	1.152
Heart attack	**1.521**	1.080	**1.521**	0.864	1.152	0.864	1.080	1.080
Rain	0.929	1.080	1.001	**2.538**	0.804	0.929	0.864	0.998
Car	0.748	0.748	1.268	0.864	0.747	0.748	0.929	**1.440**
Car accident	**1.268**	1.080	1.001	0.864	0.998	0.864	1.239	**1.268**
Door	0.864	0.864	**1.688**	0.929	0.804	1.001	1.001	1.558
Fleet	1.168	1.335	1.268	1.152	**1.384**	1.335	1.268	1.239
Route	0.929	0.998	1.558	1.239	1.072	1.072	1.558	**1.688**
Traffic	1.168	1.268	1.268	1.152	1.384	1.168	1.384	**1.688**

method 30 (thirty) times, as 3 (three) terms to be compared with 10 (ten) categories for the *Transportation* domain. This process occurs for the other domains as well.

Table 6.4 shows the *Classification* scores obtained with this experiment. The expected domains of classification are highlighted in color in Table 6.4. The greater score for each main data is presented in **bold** at Table 6.4, denoting that the developed *Classification* method implies that such domain is the more related with the main data terms.

The *Classification* scores rely on a correct domain inferencing for most cases. The accuracy of the method is of 80%. It classifies the domain wrongly in the *Ambulance* and *Pacemaker* contexts. The correct classification would be the *Healthcare* domain for both *Ambulance* and *Pacemaker*. It is important to highlight the difference between *Car* and *Car accident* context classification. For the first one, only the *Transportation* domain was notified, for the second, both *Transportation* and *Healthcare* domains received the context, what highlights the accuracy of the proposed model.

6.3 EXPERIMENT 3 - CONTEXT-AWARE SECURITY

This Section evaluates the context-aware security provision of the Context Sharing Architecture (see Section 5.3). The goal is to test the context-aware security implementation regarding its performance in running pre-established rules and providing security decisions.

6.3.1 Environment Setup

The context-aware security provision uses the context information to infer security decisions. It can get context information both directly from the device and from the Fog Repository module, which contains the already *classified* context (see Section 5.2). It has a set o rules that infer different security decisions (e.g., authentication, authorization, access control, and privacy-preserving). The rules are structured in an IF-THEN-ELSE manner, in which the *THEN* can provide different actions, related to the different security decisions aforementioned.

Every context information received by the Edge layer will pass by this module to verify if it has some security rule linked to it. In light of this, it is essential to the context-aware security provision implementation to have a good performance, specially in Edge layer devices, which have less computational processing power when compared with the Fog and Cloud layers devices/systems.

This experiment evaluates the performance of the Edge layer device on performing the context-aware security provision by running all the rules on its domain. The time taken by the Edge layer device was measured for different situations, varying the number of contexts to be analyzed and the number of rules conditions.

As the context-aware security provision occurs at the Edge layer, only the Edge layer is used in this experiment. It was hosted by a Raspberry Pi 3 Model B configured with Raspbian, Quad Core 1.2GHz CPU and 1GB RAM. The *venmo/business-rules* library for Pyhton was used to provide the context-aware security provision.

6.3.2 Experiment Results

For this experiment, the number of contexts varies from only 1 (one) information to 10000 (ten thousand). For each number of context variations, it varied also the number of rules conditions. Each rule condition represents one test to be made with the contexts to verify if it has a security decision related to it. For example, the Figure 5.9 shows an example with 3 (three) rules conditions. In this experiment, the rules conditions vary from 3 (three) to

300 (three hundred). Figure 6.6 shows the results (average of 30 executions for each one of the variations).

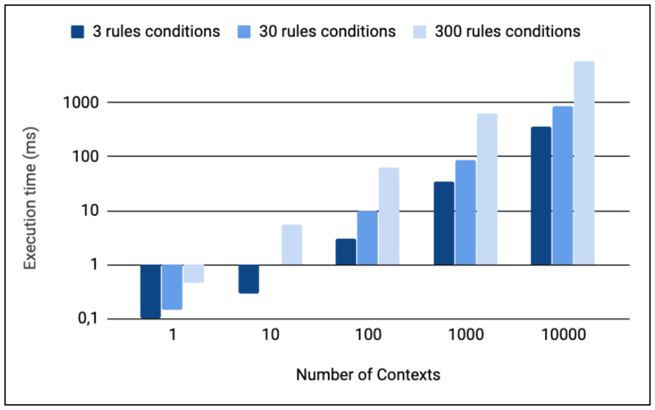


Figure 6.6 – Execution time (ms) of the context-aware security provision varying number of contexts and rules conditions.

Table 6.5 shows the detailed context-aware security provision execution time results. It shows that such processing is suitable for Internet of Things environments. In most scenarios, even with ten thousand context information being processed simultaneously, the entire processing takes less than one second (884 ms). However, in the most extreme tested scenario, with ten thousand context information and three hundred rules conditions, the execution time reaches the maximum number of almost 6 seconds (5835.9 ms). The results show that the context-aware security provision can be applied at the Edge layer (Context Provider) for most scenarios. However, it will perform better when applied to a specific domain scenario rather than to a generic scenario that could contain thousands of context information.

Table 6.5 – Execution time (ms) of context-aware security provision.

Number of	Number of rules conditions		
Contexts	3	30	300
1	0.1	0.15	0.46
10	0.3	1.01	5.6
100	3	10.01	64
1000	35	87.2	620.1
10000	365.1	884	5835.9

6.4 RESULTS DISCUSSION

The results presented at this Chapter were taken aiming to investigate the behavior of the proposed platform. We used different hardware pieces to perform the tests. Manly, the Fog layer was hosted by a regular computer, and the Edge layer by development boards, with less computational capabilities when compared with the Fog layer devices.

Our platform allows the deployment of its architectural modules in different IoT entities, many times using different network technologies. For that, the tests evaluated its performance regarding execution time for both the context sharing and context-aware security provision. The network effort appeared as the most time-demanding task regarding the whole process, which shows that the processing time for performing context sharing provision is suitable for such a scenario. The context-aware security processing execution time scales up when a massive set of rules were fired simultaneously. However, we believe that security improvement and its unique characteristic of using shared context information for its provision surpasses the liens. Moreover, the accuracy of the context sharing feature was also measured. The tests showed that for the majority of the cases, the context information was classified correctly. Depending on the scenario, the percentage can grow, as it is possible to add more complex ontologies to the classification.

An extensive discussion about the characteristics and architecture view of approaches similar to the one proposed in this work were presented in Chapter 3. That comparison was made both regarding Context Sharing and Context-Aware Security. On the other hand, it is difficult to compare the results of the experiments of the proposed architecture with already developed platforms because of its singularity. As none of the analyzed similar work provides the Context-Aware Security feature using shared information, every direct execution time comparison would become unfair.

7. FINAL CONSIDERATIONS

This Chapters presents the final considerations related to this work. First, the contributions and publications related to the developed work are presented. Finally, the conclusions and future work are presented.

7.1 CONTRIBUTIONS

Among the contributions presented in this work, it is possible to identify the main contributions in the following items:

- The extensive review of the background technologies used for providing context sharing feature in Internet of Things (Chapter 2). Published in [43] and [39].
- The definition of a taxonomy for the context-aware security provision encompassing its possible application areas (Chapter 2). Published partially in [42].
- The extensive review of the state-of-the art in both context sharing and context-aware security area. Showing both academic and industry works for each area (Chapter 3). Published partially in [43] and [42].
- The definition and implementation of a context sharing architecture able to provide context information interoperability for Internet of Things environments (Chapter 4 and Chapter 5). Published in [41].
- The definition and implementation of a context-aware security provision method for Internet of Things environments that uses shared context information. (Chapter 4 and Chapter 5). Published partially in [42].
- The creation and usage of a reference platform for devices management and context-aware services provisions (Chapter 5). Published in [8], [7], [90], and [40].
- The execution of tests to validate the developed contex sharing architecture and context-aware security provision method (Chapter 6). Published partially in [41].
- This work was awarded with a Fulbright Doctoral Dissertation Award scholarship to be developed partially in the United States. It was developed in collaboration with the University of Southern California Viterbi School of Engineering's Center for Cyber-Physical Systems and the Internet of Things. This collaboration has strengthened the relation between PUCRS and University of Southern California, Los Angeles. This work contributed to research in context management systems for the I3 USC project.

- The publication of 12 (twelve) scientific papers and 1 (one) accepted for publication during the Ph.D. as shows the Section 7.2. The publications are: 3 (three) international journals, 3 (three) book chapters, and 6 (six) international conferences.

7.2 PUBLICATIONS

Table 7.1 summarizes the author's previous works in chronological order (from the newest to the oldest). The studies in [7] and in [121] give definitions and implementation of middleware systems in IoT environments, showing how they manage devices, and provide interoperability. The studies in [135], [136], [139], [138], and [96] explain how security can be reached in the IoT. The studies [39] and in [40] explain details of the context-aware feature of IoT systems and how it can be used in a system/architecture. The studies [41] and in [43] relate to the context sharing provision. The study [42] relates to the context-aware security provision.

Table 7.1 – Papers published during the PhD degree.

Ref.	Work Title	Venue and Publisher	Year	Impact Factor
Accepted	Edge Decentralized Security Architecture for Industrial IoT Applications	IEEE World Forum on Internet of Things (WF-IoT)	2020	—
[43]	Context information sharing for the Internet of Things: A survey	Elsevier Computer Networks	2020	3.03
[96]	Privacy and security of Internet of Things devices	Real-Time Data Analytics for Large Scale Sensor Data (Academic Press)	2020	—
[138]	Evaluating the DTLS Protocol from CoAP in Fog-to-Fog Communications	IEEE International Conference on & Service-Oriented System Engineering (SOSE)	2019	—
[139]	Lightweight Security Architecture Based on Embedded Virtualization and Trust Mechanisms for IoT Edge Devices	IEEE Communications Magazine	2019	10.35
[42]	Providing Context-Aware Security for Environments Through Context Sharing Feature	IEEE International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom)	2018	—
[41]	Context Interoperability for IoT through an Edge-centric Context Sharing Architecture	IEEE Symposium on Computers and Communications (ISCC)	2018	—
[40]	A Sensing-as-a-Service Context-Aware System for Internet of Things Environments	IEEE Consumer Communications & Networking Conference (CCNC)	2017	—
[136]	Evaluating the Use of TLS and DTLS Protocols in IoT Middleware Systems Applied to E-health	Consumer Communications & Networking Conference (CCNC)	2017	—
[39]	Context-Aware Systems: Technologies and Challenges in Internet of Everything Environments	Beyond the Internet of Things: Everything Interconnected (Springer International Publishing)	2017	—
[135]	The Role of Lightweight Approaches Towards the Standardization of a Security Architecture for IoT Middleware Systems	IEEE Communications Magazine	2016	10.35
[7]	Middleware Technology for IoT Systems: Challenges and Perspectives Toward 5G	Internet of Things (IoT) in 5G Mobile Technologies (Springer International Publishing)	2016	—
[121]	Arquitetura para Fog Computing em Sistemas de Middleware para Internet das Coisas	Congresso da Sociedade Brasileira de Computação (CSBC)	2016	—

7.3 REVISITING THE HYPOTHESES AND RESEARCH QUESTIONS

This Dissertation investigated two hypotheses: (i) context sharing is well suited to provide context interoperability among different systems of IoT environments; and (ii) it is possible to provide security based on context to IoT entities through context sharing feature.

For the first hypothesis (i), the definition of an architecture presented in Chapters 4 and 5 shown that the Semantic Processing and Context Classification by ontologies methods are a suitable approach for providing context sharing feature. The evaluation presented in Sections 6.1 and 6.2 shown that it is suitable for IoT environments both in performance and accuracy.

For the second hypothesis (ii), by the developed architecture (see Chapters 4 and 5), it is possible to provide context information interoperability by the implementation of the FIWARE Orion Context Broker. The context-aware security method can access this context information pool and provide security actions with the shared context information.

In the Chapter 2, the **Research Question** *"Which requirements a system that produces context information must have to be prepared to share its context information with other entities?"* was answered by the definition of the context sharing building blocks that a platform must have for sharing context information.

In the Chapters 4 and 5, the **Research Question** *"How can the context information interoperability between heterogeneous IoT platforms that produce different kinds of context be provided?"* was answered by the definition and implementation of a Context Sharing Architecture that provides reasoning methods able to analyze context information generically, making it possible to be classified with a specific domain.

In the Chapter 5, the **Research Question** *"What criteria will be taken into account to define which entities will receive the shared context information and which entity will perform this control?"* was answered by the presentation of implementation details in the FIWARE Orion Context Broker developed instance. The Fog and Cloud Repositories classifies the context information by domains, that is the criteria to define which entities receive the context information, also those modules perform this control.

In the Chapters 2 and 5, the **Research Question** *"In which ways can shared context information be used for context-based security provisioning and how it can be implemented?"* was answered by the definition of a taxonomy demonstrating the ways that context-aware security can be provided (Chapter 2), and the implementation of a context-aware security method that uses context information from the FIWARE Orion Context Broker pool for providing security decisions (Chapter 5).

7.4 CONCLUSION

Context sharing applied to IoT environments has become mandatory. Although the development of such approach is noteworthy, it is crucial to be careful with the way in which it is applied. There are two critical challenges to overcome that are missing in the existing systems. The first challenge is to deal with the large heterogeneity of IoT environments. The use and optimization of ontologies and web services can be a first step towards the mitigation of this issue. The second challenge is related to scalability and real-time sharing. There is a need to optimize mechanisms in order to minimize the data/context traffic between entities. The use of Edge Computing concept can be a way of reducing the extra information exchanged.

Even that there are various sharing platforms deployed with different characteristics, there are challenges to be overcome. In this work, it was presented essential building blocks towards the development of a context sharing platform. It was also introduced various existing context information sharing platforms and discussed their features in detail. Moreover, it was reviewed the challenges and open issues for such platforms, the potential enhancements for them alongside with the definition of a Context Sharing Architecture that encompasses all the building blocks and the discussed challenges.

It was presented recent trends and advancements in the context-aware security area applied to IoT environments. It was defined a taxonomy of context-aware security, and outlined the key requirements for the deployment of context-aware security solutions in IoT environments. Also, existing solutions in the area were presented. Based on the conducted study, it is possible to conclude that although the deployment of context-aware security in IoT environments provides many benefits, it is essential to care about the IoT characteristics, as the vast heterogeneity, for the consolidation of the context-aware security area. A next step to the context-aware security solutions found in the literature would be to use context information from different domains when providing security services. Most of the solutions are developed thinking about only on a specific domain, whereas in IoT, it is very common solutions integrating more than one domain (e.g., smart cities).

This work defined, implemented, and validated an Edge-based Context Sharing Architecture for the Internet of Things Environments. The proposed architecture performs hybrid reasoning for sharing context information. It uses two different techniques for such processing: lexical analysis and ontologies. This hybrid approach appears as a viable solution for heterogeneous IoT environments, which may have heterogeneous devices with different processing power capabilities. Moreover, a context-aware security provision method that uses shared context information from different domains was presented, standing out for the traditional context-aware security solutions that mainly focus on only one application domain.

When compared with the related work approaches (see Table 3.2 and Table 3.4), the Context Sharing Architecture developed in this work stands out by the possibility of using shared context information to provide Context-Aware Security decisions. The proposed architecture provides an interoperable way of sharing context information, regardless of the domain that the IoT entity is inserted in. Also, it provides context-aware security services with shared context information from different domains, which is considered a gap in the area.

7.5 FUTURE WORK

As future work, it is possible to identify the following possibilities:

- The context-aware security is an incipient area with great possibility of development. New ways of providing such security services should be researched and implemented. The use of Machine Learning appears also as a trend in Computer Science area. Studies of context-aware security services provided by Machine Learning techniques were not developed yet and should be investigated.
- Different ways of protecting the context information should be investigated. The use of blockchain technologies is already a trend in Computer Science solutions. Moreover, lightweight blockchain approaches are starting to fit with the Internet of Things scenarios. It is important to research and develop solutions that combine both blockchain technology and context information management.
- IoT data marketplaces have been developed for the Internet of Things environments. In such marketplaces, the IoT data have a monetary value and can be commercialized by peers. However, most of the IoT data marketplaces deals only with raw IoT device data. The commercialization of context-aware data, which is a more enriched data than raw device one, should be considered for such scenarios.

REFERENCES

[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] Agarwal, R.; Fernandez, D. G.; Elsaleh, T.; Gyrard, A.; Lanza, J.; Sanchez, L.; Georgantas, N.; Issarny, V. "Unified IoT ontology to enable interoperability and federation of testbeds". In: Proceedings of the 3rd IEEE World Forum on Internet of Things, 2016, pp. 70–75.

[3] Al-Muhtadi, J.; Ranganathan, A.; Campbell, R.; Mickunas, M. D. "Cerberus: a context-aware security scheme for smart spaces". In: Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications, 2003, pp. 489–496.

[4] Al-Turjman, F.; Alturjman, S. "Confidential smart-sensing framework in the IoT era", *The Journal of Supercomputing*, vol. 74–10, Oct 2018, pp. 5187–5198.

[5] Al-Turjman, F.; Alturjman, S. "Context-Sensitive Access in Industrial Internet of Things (IIoT) Healthcare Applications", *IEEE Transactions on Industrial Informatics*, vol. 14–6, Jun 2018, pp. 2736–2744.

[6] Alagar, V.; Alsaig, A.; Ormandjiva, O.; Wan, K. "Context-Based Security and Privacy for Healthcare IoT". In: Proceedings of the 2nd IEEE International Conference on Smart Internet of Things, 2018, pp. 122–128.

[7] 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.

[8] 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.

[9] Anand, N.; Yang, M.; van Duin, J.; Tavasszy, L. "Genclon: An ontology for city logistics", *Expert Systems with Applications*, vol. 39–15, Nov 2012, pp. 11944 – 11960.

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

[11] Azeez, A.; Perera, S.; Gamage, D.; Linton, R.; Siriwardana, P.; Leelaratne, D.; Weerawarana, S.; Fremantle, P. “Multi-tenant SOA Middleware for Cloud Computing”. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing, 2010, pp. 458–465.

[12] Baker, T.; Whitehead, B.; Musker, R.; Keizer, J. “Global agricultural concept space: lightweight semantics for pragmatic interoperability”, *NPJ Science of Food*, vol. 3–1, Sep 2019, pp. 1–8.

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

[14] Bellini, P.; Benigni, M.; Billero, R.; Nesi, P.; Rauch, N. “Km4city ontology building vs data harvesting and cleaning for smart-city services”, *Journal of Visual Languages & Computing*, vol. 25–6, Dec 2014, pp. 827 – 839.

[15] Bettini, C.; Brdiczka, O.; Henricksen, K.; Indulska, J.; Nicklas, D.; Ranganathan, A.; Riboni, D. “A survey of context modelling and reasoning techniques”, *Pervasive and Mobile Computing*, vol. 6–2, Apr 2010, pp. 161–180.

[16] Bikakis, A.; Patkos, T.; Antoniou, G.; Plexousakis, D. “A survey of semantics-based approaches for context reasoning in ambient intelligence”. In: Proceedings of the 1st European Conference on Ambient Intelligence - Constructing Ambient Intelligence, 2008, pp. 14–23.

[17] Bird, S.; Klein, E.; Loper, E. “Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit”. Beijing: O’Reilly, 2009, 504p.

[18] Boavida, F.; Kliem, A.; Renner, T.; Riekki, J.; Jouvray, C.; Jacovi, M.; Ivanov, S.; Guadagni, F.; Gil, P.; Triviño, A. “People-Centric Internet of Things—Challenges, Approach, and Enabling Technologies”. In: Intelligent Distributed Computing IX: Proceedings of the 9th International Symposium on Intelligent Distributed Computing, Novais, P.; Camacho, D.; Analide, C.; El Fallah Seghrouchni, A.; Badica, C. (Editors), 2016, pp. 463–474.

[19] Bonomi, F.; Milito, R.; Natarajan, P.; Zhu, J. “Fog Computing: A Platform for Internet of Things and Analytics”. Cham: Springer International Publishing, 2014, chap. 7, pp. 169–186.

[20] Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. “Fog computing and its role in the internet of things”. In: Proceedings of the Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

[21] Brezillon, P.; Mostefaoui, G. K. “Context-based security policies: a new modeling approach”. In: Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004, pp. 154–158.

[22] Bruneo, D.; Distefano, S.; Longo, F.; Merlino, G.; Puliafito, A.; D’Amico, V.; Sapienza, M.; Torrisi, G. “Stack4Things as a fog computing platform for Smart City applications”. In: Proceedings of the 35th IEEE Conference on Computer Communications Workshops, 2016, pp. 848–853.

[23] Budanitsky, A.; Hirst, G. “Evaluating wordnet-based measures of lexical semantic relatedness”, *Computational Linguistics*, vol. 32–1, May 2006, pp. 13–47.

[24] Bugiel, S.; Heuser, S.; Sadeghi, A.-R. “Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies”. In: Proceedings of the 22nd USENIX Security Symposium, 2013, pp. 131–146.

[25] Byers, C. C. “Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks”, *IEEE Communications Magazine*, vol. 55–8, Aug 2017, pp. 14–20.

[26] Cabitza, F.; Dal Seno, B. “DJess-A Knowledge-Sharing Middleware to Deploy Distributed Inference Systems”, *International Journal of Computer and Information Engineering*, vol. 1–4, Jan 2007, pp. 1023–1026.

[27] Casadei, R.; Fortino, G.; Pianini, D.; Russo, W.; Savaglio, C.; Viroli, M. “A development approach for collective opportunistic Edge-of-Things services”, *Information Sciences*, vol. 498, Sep 2019, pp. 154 – 169.

[28] Chandola, V.; Banerjee, A.; Kumar, V. “Anomaly detection: A survey”, *ACM Computing Surveys*, vol. 41–3, Jul 2009.

[29] Chazelle, B.; Kilian, J.; Rubinfeld, R.; Tal, A. “The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables”. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 30–39.

[30] Chen, G.; Kotz, D. “A Survey of Context-Aware Mobile Computing Research”, Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, 2000, 16p.

[31] Cho, S.; Julien, C. “The Grapevine Context Processor: Application Support for Efficient Context Sharing”. In: Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems, 2015, pp. 68–71.

[32] Cho, S.; Julien, C. “Chitchat: Navigating tradeoffs in device-to-device context sharing”. In: Proceedings of the 14th International Conference on Pervasive Computing and Communications, 2016, pp. 1–10.

[33] Compton, M.; Barnaghi, P.; Bermudez, L.; GarcíA-Castro, R.; Corcho, O.; Cox, S.; Graybeal, J.; Hauswirth, M.; Henson, C.; Herzog, A.; et al.. "The SSN ontology of the W3C semantic sensor network incubator group", *Web semantics: science, services and agents on the World Wide Web*, vol. 17, Dec 2012, pp. 25–32.

[34] Covington, M. J.; Fogla, P.; Zhan, Z.; Ahamad, M. "A context-aware security architecture for emerging applications". In: Proceedings of the 18th Annual Computer Security Applications Conference, 2002, pp. 249–258.

[35] Das, P. K.; Narayanan, S.; Sharma, N. K.; Joshi, A.; Joshi, K.; Finin, T. "Context-Sensitive Policy Based Security in Internet of Things". In: Proceedings of the 2nd IEEE International Conference on Smart Computing, 2016, pp. 1–6.

[36] Dastjerdi, A. V.; Buyya, R. "Fog Computing: Helping the Internet of Things Realize Its Potential", *Computer*, vol. 49–8, Aug 2016, pp. 112–116.

[37] De, S.; Cassar, G.; Christophe, B.; Fredj, S. B.; Bauer, M.; Santos, N.; Jacobs, T.; Zeybek, E.; de las Heras, R.; Martín, G.; et al.. "Concepts and solutions for entity-based discovery of IoT resources and managing their dynamic associations", Technical Report IoT-A D4 3, EC FP7, 2012, 202p.

[38] de Freitas, A. A.; Nebeling, M.; Ranithangam, A. S. K. K.; Yang, J.; Dey, A. K. "Bluewave: Enabling Opportunistic Context Sharing via Bluetooth Device Names". In: Proceedings of the 8th Symposium on Engineering Interactive Computing Systems, 2016, pp. 38–49.

[39] de Matos, E.; Amaral, L. A.; Hessel, F. "Context-Aware Systems: Technologies and Challenges in Internet of Everything Environments". Cham: Springer International Publishing, 2017, chap. 1, pp. 1–25.

[40] de Matos, E.; Amaral, L. A.; Tiburski, R. T.; Schenfeld, M.; Hessel, F.; de Azevedo, D. "A Sensing-as-a-Service Context-Aware system for internet of things environments". In: Proceedings of the 14th IEEE Annual Consumer Communications & Networking Conference, 2017, pp. 725–728.

[41] de Matos, E.; Tiburski, R. T.; Amaral, L. A.; Hessel, F. "Context Interoperability for IoT Through an Edge-Centric Context Sharing Architecture". In: Proceedings of the 23th IEEE Symposium on Computers and Communications, 2018, pp. 00667–00670.

[42] de Matos, E.; Tiburski, R. T.; Amaral, L. A.; Hessel, F. "Providing Context-Aware Security for IoT Environments Through Context Sharing Feature". In: Proceedings of the 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications, 2018, pp. 1711–1715.

[43] de Matos, E.; Tiburski, R. T.; Moratelli, C. R.; Filho, S. J.; Amaral, L. A.; Ramachandran, G.; Krishnamachari, B.; Hessel, F. “Context information sharing for the Internet of Things: A survey”, *Computer Networks*, vol. 166, Jan 2020, pp. 1–19.

[44] de Souza, R. S.; Lopes, J. L. B.; Geyer, C. F. R.; da Rosa Silveira João, L.; Cardozo, A. A.; Yamin, A. C.; Gadotti, G. I.; Barbosa, J. L. V. “Continuous monitoring seed testing equipaments using internet of things”, *Computers and Electronics in Agriculture*, vol. 158, Mar 2019, pp. 122 – 132.

[45] Dhallenne, J.; Jayaraman, P. P.; Zaslavsky, A. “RCOS: Real Time Context Sharing Across a Fleet of Smart Mobile Devices”. Cham: Springer International Publishing, 2016, chap. 8, pp. 87–100.

[46] Dolui, K.; Datta, S. K. “Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing”. In: Proceedings of the Global Internet of Things Summit, 2017, pp. 1–6.

[47] Doukas, C.; Capra, L.; Antonelli, F.; Jaupaj, E.; Tamilin, A.; Carreras, I. “Providing generic support for IoT and M2M for mobile devices”. In: Proceedings of the 12th IEEE RIVF International Conference on Computing Communication Technologies - Research, Innovation, and Vision for Future, 2015, pp. 192–197.

[48] ETSI. “ETSI launches new group on Context Information Management for smart city interoperability”. Source: http://goo.gl/PLAoHb, Sep 2019.

[49] ETSI. “ETSI ISG CIM group releases first specification for context exchange in smart cities”. Source: http://goo.gl/QgTRdE, Sep 2019.

[50] European Commission - Horizon 2020. “AutoMat - Automotive Big Data Marketplace for Innovative Cross-sectorial Vehicle Data Services”. Source: https://cordis.europa.eu/project/rcn/194228_en.html, Sep 2018.

[51] Facca, F. M.; Komazec, S.; Guglielmina, C.; Gusmeroli, S. “COIN: Platform and Services for SaaS in Enterprise Interoperability and Enterprise Collaboration”. In: Proceedings of the 3rd IEEE International Conference on Semantic Computing, 2009, pp. 543–550.

[52] Faieq, S.; Saidi, R.; Elghazi, H.; Rahmani, M. D. “C2iot: A framework for cloud-based context-aware internet of things services for smart cities”, *Procedia Computer Science*, vol. 110, Jul 2017, pp. 151 – 158.

[53] Finlayson, M. “Java libraries for accessing the princeton wordnet: Comparison and evaluation”. In: Proceedings of the 7th Global Wordnet Conference, 2014, pp. 78–85.

[54] Fortino, G.; Savaglio, C.; Palau, C. E.; de Puga, J. S.; Ganzha, M.; Paprzycki, M.; Montesinos, M.; Liotta, A.; Llop, M. “Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach”. Cham: Springer International Publishing, 2018, chap. 10, pp. 199–232.

[55] Friedman-Hill, E. “Jess, the Java expert system shell”. Source: https://www.osti.gov/biblio/565603, Sep 2018.

[56] Gansel, S.; Schnitzer, S.; Gilbeau-Hammoud, A.; Friesen, V.; Dürr, F.; Rothermel, K.; Maihöfer, C.; Krämer, U. “Context-Aware Access Control in Novel Automotive HMI Systems”. Cham: Springer International Publishing, 2015, chap. 8, pp. 118–138.

[57] Gheisari, M.; Wang, G.; Khan, W. Z.; Fernández-Campusano, C. “A context-aware privacy-preserving method for IoT-based smart city using Software Defined Networking”, *Computers & Security*, vol. 87, Nov 2019, pp. 101470.

[58] Gupta *et al.*, H. “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments”, *Software: Practice and Experience*, vol. 47–9, Jun 2017, pp. 1275–1296.

[59] Gyrard, A. “A Machine-to-machine Architecture to Merge Semantic Sensor Measurements”. In: Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 371–376.

[60] Gyrard, A.; Bonnet, C.; Boudaoud, K. “Enrich machine-to-machine data with semantic web technologies for cross-domain applications”. In: Proceedings of the 1st IEEE World Forum on Internet of Things, 2014, pp. 559–564.

[61] Habib, K.; Leister, W. “Context-Aware Authentication for the Internet of Things”. In: Proceedings of the 11th International Conference on Autonomic and Autonomous Systems, 2015, pp. 6.

[62] Harif, S. “Rule-based operation and service provider authentication for a keyed system”. US Patent App. 09/751,829, Source: https://patents.google.com/patent/US20020133716A1/en, Sep 2018.

[63] Hassani, A.; Medvedev, A.; Haghighi, P. D.; Ling, S.; Indrawan-Santiago, M.; Zaslavsky, A.; Jayaraman, P. P. “Context-as-a-Service Platform: Exchange and Share Context in an IoT Ecosystem”. In: Proceedings of the 16th IEEE International Conference on Pervasive Computing and Communications Workshops, 2018, pp. 385–390.

[64] Hayashi, E.; Das, S.; Amini, S.; Hong, J.; Oakley, I. “CASA: Context-aware Scalable Authentication”. In: Proceedings of the 9th Symposium on Usable Privacy and Security, 2013, pp. 3:1–3:10.

[65] Herley, C. “So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users”. In: Proceedings of the 12th New Security Paradigms Workshop, 2009, pp. 133–144.

[66] Hosseinzadeh, S.; Virtanen, S.; Díaz-Rodríguez, N.; Lilius, J. “A Semantic Security Framework and Context-aware Role-based Access Control Ontology for Smart Spaces”. In: Proceedings of the 1st International Workshop on Semantic Big Data, 2016, pp. 8:1–8:6.

[67] Houda, M.; Khemaja, M.; Oliveira, K.; Abed, M. “A public transportation ontology to support user travel planning”. In: Proceedings of the 4th International Conference on Research Challenges in Information Science, 2010, pp. 127–136.

[68] Hu *et al.*, J. “A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications”. In: Proceedings of the 1st Workshop on Pervasive Privacy Security, Privacy, and Trust, 2004, pp. 1–8.

[69] Hulsebosch, R. J.; Salden, A. H.; Bargh, M. S.; Ebben, P. W. G.; Reitsma, J. “Context Sensitive Access Control”. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, 2005, pp. 111–119.

[70] Huru, D.; Leordeanu, C.; Apostol, E.; Cristea, V. “BigClue: Towards a generic IoT cross-domain data processing platform”. In: Proceedings of the 14th IEEE International Conference on Intelligent Computer Communication and Processing, 2018, pp. 427–434.

[71] Iwamura, M. “NGMN View on 5G Architecture”. In: Proceedings ot the 81st IEEE Vehicular Technology Conference, 2015, pp. 1–5.

[72] Jia, Y. J.; Chen, Q. A.; Wang, S.; Rahmati, A.; Fernandes, E.; Mao, Z. M.; Prakash, A.; Unviersity, S. J. “ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms”. In: Proceedings of the 21st Network and Distributed System Security Symposium, 2017, pp. 1–15.

[73] João, L.; Machado, R.; Tabim, V.; Cardoso, A.; Lopes, J. L.; Pernas, A.; Yamin, A. “Applying the internet of things in precision viticulture: An approach exploring the exehda middleware”. In: Proceedings of the 44th Latin American Computer Conference, 2018, pp. 680–687.

[74] Kansal, A.; Nath, S.; Liu, J.; Zhao, F. “SenseWeb: An Infrastructure for Shared Sensing”, *IEEE MultiMedia*, vol. 14–4, Oct 2007, pp. 8–13.

[75] Kim, J.; Chung, K.-Y. “Ontology-based healthcare context information model to implement ubiquitous environment”, *Multimedia Tools and Applications*, vol. 71–2, Nov 2014, pp. 873–888.

[76] Krishnamachari, B.; Power, J.; Kim, S. H.; Shahabi, C. “I3: An IoT marketplace for smart communities”. In: Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, 2018, pp. 498–499.

[77] Kroner, A.; Schneider, M.; Mori, J. “A Framework for Ubiquitous Content Sharing”, *IEEE Pervasive Computing*, vol. 8–4, Oct 2009, pp. 58–65.

[78] Lai, T.; Li, W.; Liang, H.; Zhou, X. “FRASCS: A Framework Supporting Context Sharing”. In: Proceedings of the 9th International Conference for Young Computer Scientists, 2008, pp. 919–924.

[79] Langheinrich, M. “A Privacy Awareness System for Ubiquitous Computing Environments”. Berlin: Springer Berlin Heidelberg, 2002, chap. 19, pp. 237–245.

[80] Leacock, C.; Miller, G. A.; Chodorow, M. “Using corpus statistics and wordnet relations for sense identification”, *Computational Linguistics*, vol. 24–1, Mar 1998, pp. 147–165.

[81] Li, W.; Joshi, A.; Finin, T. “SVM-CASE: An SVM-Based Context Aware Security Framework for Vehicular Ad-Hoc Networks”. In: Proceedings of the 82nd IEEE Vehicular Technology Conference, 2015, pp. 1–5.

[82] Liu, C.; Hua, J.; Julien, C. “SCENTS: Collaborative Sensing in Proximity IoT Networks”. In: Proceedings of the 17th IEEE International Conference on Pervasive Computing and Communications Workshops, 2019, pp. 189–195.

[83] Liu, C.; Julien, C. “Pervasive context sharing in magpie: Adaptive trust-based privacy protection”. In: Mobile Computing, Applications, and Services, 2015, pp. 122–139.

[84] Lu, L.; Xu, L.; Xu, B.; Li, G.; Cai, H. “Fog Computing Approach for Music Cognition System Based on Machine Learning Algorithm”, *IEEE Transactions on Computational Social Systems*, vol. 5–4, Dec 2018, pp. 1142–1151.

[85] Lunardi, W.; Matos, E.; Tiburski, R.; Amaral, L.; Marczak, S.; Hessel, F. “Context-based Search Engine for Industrial IoT: Discovery, Search, Selection, and Usage of Devices”. In: Proceedings of the 20th IEEE Conference on Emerging Technology & Factory Automation, 2015, pp. 1–8.

[86] Maarala, A. I.; Su, X.; Riekki, J. “Semantic Reasoning for Context-Aware Internet of Things Applications”, *IEEE Internet of Things Journal*, vol. 4–2, Apr 2017, pp. 461–473.

[87] Madhukalya, M.; Kumar, M. “ConCon: Context-Aware Middleware for Content Sharing in Dynamic Participating Environments”. In: Proceedings of the 15th IEEE International Conference on Mobile Data Management, 2014, pp. 156–161.

[88] Manashty, A.; Light, J.; Yadav, U. “Healthcare event aggregation lab (HEAL), a knowledge sharing platform for anomaly detection and prediction”. In: Proceedings of the 17th International Conference on E-health Networking, Application Services, 2015, pp. 648–652.

[89] Manashty, A.; Thompson, J. L. “Cloud Platforms for IoE Healthcare Context Awareness and Knowledge Sharing”. Cham: Springer International Publishing, 2017, chap. 12, pp. 303–322.

[90] Matos, E.; Amaral, L.; Tiburski, R.; Lunardi, W.; Hessel, F.; Marczak, S. “Context-aware system for information services provision in the Internet of Things”. In: Proceedings of the 20th IEEE Conference on Emerging Technologies Factory Automation, 2015, pp. 1–4.

[91] Miettinen, M.; Heuser, S.; Kronz, W.; Sadeghi, A.-R.; Asokan, N. “ConXsense: Automated Context Classification for Context-aware Access Control”. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, 2014, pp. 293–304.

[92] Miller, G. A. “Wordnet: A lexical database for english”, *Communications of the ACM*, vol. 38–11, Nov 1995, pp. 39–41.

[93] Mingozzi, E.; Tanganelli, G.; Vallati, C.; Martínez, B.; Mendia, I.; González-Rodríguez, M. “Semantic-based context modeling for quality of service support in IoT platforms”. In: Proceedings of the 17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks, 2016, pp. 1–6.

[94] Morabito, R.; Petrolo, R.; Loscrí, V.; Mitton, N. “Enabling a lightweight Edge Gateway-as-a-Service for the Internet of Things”. In: Proceedings of the 7th International Conference on the Network of the Future, 2016, pp. 1–5.

[95] Moratelli, C.; Johann, S.; Neves, M.; Hessel, F. “Embedded Virtualization for the Design of Secure IoT Applications”. In: Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, 2016, pp. 2–6.

[96] Moratelli, C. R.; Tiburski, R. T.; de Matos, E.; Portal, G.; Johann, S. F.; Hessel, F. “Privacy and security of Internet of Things devices”. In: *Real-Time Data Analytics for Large Scale Sensor Data*, Das, H.; Dey, N.; Balas, V. E. (Editors), Academic Press,

2020, *Advances in Ubiquitous Sensing Applications for Healthcare*, vol. 6, chap. 9, pp. 183 – 214.

[97] Mostefaoui, G. K.; Brezillon, P. “Modeling context-based security policies with contextual graphs”. In: Proceedings of the IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004, pp. 28–32.

[98] Mowafi, Y.; Abou-Tair, D.; Aqarbeh, T.; Abilov, M.; Dmitriyev, V.; Gomez, J. M. “A Context-aware Adaptive Security Framework for Mobile Applications”. In: Proceedings of the 3rd International Conference on Context-Aware Systems and Applications, 2014, pp. 147–153.

[99] Munir, A.; Kansakar, P.; Khan, S. U. “IFCIoT: Integrated Fog Cloud IoT: A novel architectural paradigm for the future Internet of Things”, *IEEE Consumer Electronics Magazine*, vol. 6–3, Jul 2017, pp. 74–82.

[100] Nagorny, K.; Scholze, S.; Ruhl, M.; Colombo, A. W. “Semantical support for a CPS data marketplace to prepare Big Data analytics in smart manufacturing environments”. In: Proceedings of the 1st IEEE Industrial Cyber-Physical Systems, 2018, pp. 206–211.

[101] Nezhad, H. R. M.; Benatallah, B.; Casati, F.; Toumani, F. “Web services interoperability specifications”, *Computer*, vol. 39–5, May 2006, pp. 24–32.

[102] Ngu, A. H.; Gutierrez, M.; Metsis, V.; Nepal, S.; Sheng, Q. Z. “IoT Middleware: A Survey on Issues and Enabling Technologies”, *IEEE Internet of Things Journal*, vol. 4–1, Feb 2017, pp. 1–20.

[103] Nihei, K. “Context sharing platform”, *NEC Journal of Advanced Technology*, vol. 1–3, Jul 2004, pp. 200–204.

[104] Nitti, M.; Pilloni, V.; Colistra, G.; Atzori, L. “The Virtual Object as a Major Element of the Internet of Things: A Survey”, *IEEE Communications Surveys Tutorials*, vol. 18–2, Secondquarter 2016, pp. 1228–1240.

[105] Niyato, D.; Lu, X.; Wang, P.; Kim, D. I.; Han, Z. “Economics of Internet of Things: an information market approach”, *IEEE Wireless Communications*, vol. 23–4, Aug 2016, pp. 136–145.

[106] Nuevo, D. A. L.; Valles, D. R.; Medina, E. M.; Pallares, R. M. “OIoT: A Platform to Manage Opportunistic IoT Communities”. In: Proceedings of the 11th International Conference on Intelligent Environments, 2015, pp. 104–111.

[107] OpenFog Consortium. “OpenFog Reference Architecture for Fog Computing”, Technical Report OPFRA001.020817, OpenFog Consortium, 2017, 162p.

[108] Pahl, C.; Helmer, S.; Miori, L.; Sanin, J.; Lee, B. "A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters". In: Proceedings of the 4th IEEE International Conference on Future Internet of Things and Cloud Workshops, 2016, pp. 117–124.

[109] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. "Context Aware Computing for The Internet of Things: A Survey", *IEEE Communications Surveys Tutorials*, vol. 16–1, First Quarter 2014, pp. 414–454.

[110] Pingley, A.; Yu, W.; Zhang, N.; Fu, X.; Zhao, W. "A context-aware scheme for privacy-preserving location-based services", *Computer Networks*, vol. 56–11, Jul 2012, pp. 2551 – 2568.

[111] Princeton University. ""About WordNet." WordNet. Princeton University." Source: http://wordnet.princeton.edu, Nov 2019.

[112] Putera, C. A.; Lin, F. J. "Incorporating OMA Lightweight M2M protocol in IoT/M2M standard architecture". In: Proceedings of the 2nd IEEE World Forum on Internet of Things, 2015, pp. 559–564.

[113] Rachid, S.; Challal, Y.; Nadjia, B. "Internet of things context-aware privacy architecture". In: Proceedings of the 12th IEEE/ACS International Conference of Computer Systems and Applications, 2015, pp. 1–2.

[114] Rahman, H.; Hussain, M.; et al.. "LiO-IoT: A Light-weight Ontology to provide Semantic Interoperability in Internet of Things", *International Journal of Computational Intelligence & IoT*, vol. 2–4, Mar 2019, pp. 571–575.

[115] Ramos, J. L. H.; Bernabe, J. B.; Skarmeta, A. F. "Managing Context Information for Adaptive Security in IoT Environments". In: Proceedings of the 29th IEEE International Conference on Advanced Information Networking and Applications Workshops, 2015, pp. 676–681.

[116] Razzaque, M. A.; Milojevic-Jevric, M.; Palade, A.; Clarke, S. "Middleware for internet of things: A survey", *IEEE Internet of Things Journal*, vol. 3–1, Feb 2016, pp. 70–95.

[117] Ricci, A.; Viroli, M.; Omicini, A. "Agent Coordination Context: From Theory to Practice". In: Proceedings of the 17th European Meeting on Cybernetics and Systems Research, 2004, pp. 618–623.

[118] Ruta, M.; Scioscia, F.; Ieva, S.; Loseto, G.; Gramegna, F.; Pinto, A. "Knowledge Discovery and Sharing in the IoT: The Physical Semantic Web Vision". In: Proceedings of the 32nd Symposium on Applied Computing, 2017, pp. 492–498.

[119] Sandhu, R. “Access control: The neglected frontier”. Berlin: Springer Berlin Heidelberg, 1996, chap. 20, pp. 219–227.

[120] Sanya, I.; Shehab, E. “An ontology framework for developing platform-independent knowledge-based engineering systems in the aerospace industry”, *International Journal of Production Research*, vol. 52–20, May 2014, pp. 6192–6215.

[121] Schenfeld, M.; Amaral, L.; de Matos, E.; Hessel, F. “Arquitetura para fog computing em sistemas de middleware para internet das coisas”. In: Proceedings of the 43rd Seminário Integrado de Software e Hardware, 2016, pp. 199–209.

[122] Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. “Edge computing: Vision and challenges”, *IEEE Internet of Things Journal*, vol. 3–5, Oct 2016, pp. 637–646.

[123] Shi, W.; Dustdar, S. “The Promise of Edge Computing”, *Computer*, vol. 49–5, May 2016, pp. 78–81.

[124] Sikder, A. K.; Babun, L.; Aksu, H.; Uluagac, A. S. “Aegis: A Context-Aware Security Framework for Smart Home Systems”. In: Proceedings of the 35th Annual Computer Security Applications Conference, 2019, pp. 28–41.

[125] Sinha, R. S.; Wei, Y.; Hwang, S.-H. “A survey on LPWA technology: LoRa and NB-IoT”, *ICT Express*, vol. 3–1, Mar 2017, pp. 14 – 21.

[126] Sinha Roy, D.; Behera, R. K.; Reddy, K. H. K.; Buyya, R. “A Context-Aware Fog Enabled Scheme for Real-Time Cross-Vertical IoT Applications”, *IEEE Internet of Things Journal*, vol. 6–2, Apr 2019, pp. 2400–2412.

[127] Slimani, T. “Description and evaluation of semantic similarity measures approaches”, *International Journal of Computer Applications*, vol. 80–10, Oct 2013, pp. 25–33.

[128] Snidaro, L.; García, J.; Llinas, J. “Context-based Information Fusion: A survey and discussion”, *Information Fusion*, vol. 25, Sep 2015, pp. 16 – 31.

[129] Staroch, P. “A weather ontology for predictive control in smart homes”. Master Thesis, Vienna University of Technology, 2013, 183p.

[130] Strang, T.; Linnhoff-Popien, C. “A context modeling survey”. In: Proceedings of the 1st International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp, 2004, pp. 1–8.

[131] Su, X.; Li, P.; Li, Y.; Flores, H.; Riekki, J.; Prehofer, C. “Towards semantic reasoning on the edge of iot systems”. In: Proceedings of the 6th International Conference on the Internet of Things, 2016, pp. 171–172.

[132] Taneja, M. "LTE-LPWA networks for IoT applications". In: Proceedings of the 7th International Conference on Information and Communication Technology Convergence, 2016, pp. 396–399.

[133] Tang, B.; Chen, Z.; Hefferman, G.; Wei, T.; He, H.; Yang, Q. "A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities". In: Proceedings of the 1st ASE BigData & SocialInformatics, 2015, pp. 28:1–28:6.

[134] Tao, M.; Zuo, J.; Liu, Z.; Castiglione, A.; Palmieri, F. "Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes", *Future Generation Computer Systems*, vol. 78, Jan 2018, pp. 1040 – 1051.

[135] Tiburski, R. T.; Amaral, L. A.; de Matos, E.; de Azevedo, D. F. G.; Hessel, F. "The Role of Lightweight Approaches Towards the Standardization of a Security Architecture for IoT Middleware Systems", *IEEE Communications Magazine*, vol. 54–12, Dec 2016, pp. 56–62.

[136] Tiburski, R. T.; Amaral, L. A.; de Matos, E.; Hessel, F.; de Azevedo, D. "Evaluating the use of TLS and DTLS protocols in IoT middleware systems applied to e-health". In: Proceedings of the 14th IEEE Annual Consumer Communications & Networking Conference, 2017, pp. 480–485.

[137] Tiburski, R. T.; Amaral, L. A.; Matos, E. D.; Hessel, F. "The importance of a standard security architecture for SOA-based IoT middleware", *IEEE Communications Magazine*, vol. 53–12, Dec 2015, pp. 20–26.

[138] Tiburski, R. T.; de Matos, E.; Hessel, F. "Evaluating the DTLS Protocol from CoAP in Fog-to-Fog Communications". In: Proceedings of the 14th IEEE International Conference on Service-Oriented System Engineering, 2019, pp. 90–905.

[139] Tiburski, R. T.; Moratelli, C. R.; Johann, S. F.; Neves, M. V.; d. Matos, E.; Amaral, L. A.; Hessel, F. "Lightweight Security Architecture Based on Embedded Virtualization and Trust Mechanisms for IoT Edge Devices", *IEEE Communications Magazine*, vol. 57–2, Feb 2019, pp. 67–73.

[140] Tikkinen-Piri, C.; Rohunen, A.; Markkula, J. "EU General Data Protection Regulation: Changes and implications for personal data collecting companies", *Computer Law & Security Review*, vol. 34–1, Feb 2018, pp. 134 – 153.

[141] Travizano, M.; Minnoni, M.; Ajzenman, G.; Sarraute, C.; Della Penna, N. "Wibson: A decentralized marketplace empowering individuals to safely monetize their personal data". Source: https://wibson.org/wp-content/uploads/2019/04/Wibson-Technical-Paper-v1.1.pdf, Nov 2019.

[142] Trnka, M.; Cerny, T. “On Security Level Usage in Context-aware Role-based Access Control”. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 1192–1195.

[143] Wang, X. H.; Zhang, D. Q.; Gu, T.; Pung, H. K. “Ontology based context modeling and reasoning using OWL”. In: Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004, pp. 18–22.

[144] Xie, K.; Luo, W.; Wang, X.; Xie, D.; Cao, J.; Wen, J.; Xie, G. “Decentralized Context Sharing in Vehicular Delay Tolerant Networks with Compressive Sensing”. In: Proceedings of the 36th International Conference on Distributed Computing Systems, 2016, pp. 169–178.

[145] Xu, L. D.; He, W.; Li, S. “Internet of Things in Industries: A Survey”, *IEEE Transactions on Industrial Informatics*, vol. 10–4, Nov 2014, pp. 2233–2243.

[146] Yamamoto, J.; Nakagawa, H.; Nakayama, K.; Tahara, Y.; Ohsuga, A. “A Context Sharing Message Broker Architecture to Enhance Interoperability in Changeable Environments”. In: Proceedings of the 3rd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2009, pp. 31–39.

[147] Zeng, W.; Zhang, S.; Yen, I.; Bastani, F. “Invited Paper: Semantic IoT Data Description and Discovery in the IoT-Edge-Fog-Cloud Infrastructure”. In: Proceedings of the 14th IEEE International Conference on Service-Oriented System Engineering, 2019, pp. 106–115.

[148] Zhang, L.; Li, Y.; Wang, L.; Lu, J.; Li, P.; Wang, X. “An Efficient Context-Aware Privacy Preserving Approach for Smartphones”, *Security and Communication Networks*, vol. 2017, Apr 2017, pp. 1–11.

[149] Zoppi, T.; Ceccarelli, A.; Bondavalli, A. “Context-awareness to improve anomaly detection in dynamic service oriented architectures”. In: Proceedings of the 35th International Conference on Computer Safety, Reliability and Security, 2016, pp. 145–158.

MARISTA

PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br