SEMU - SErvice Management for UAVs

Mariana Rodrigues

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

Mariana Rodrigues

SEMU - SErvice Management for UAVs

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dra. Kalinka Regina Lucas Jaquie Castelo Branco

USP – São Carlos
June 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

```
        Rodrigues, Mariana
R333s      SEMU - SErvice Management for UAVs / Mariana
        Rodrigues; orientadora Kalinka R. L. J. C. Branco. -
        - São Carlos, 2022.
           189 p.


           Tese (Doutorado - Programa de Pós-Graduação em
        Ciências de Computação e Matemática Computacional) --
         Instituto de Ciências Matemáticas e de Computação,
        Universidade de São Paulo, 2022.


           1. Veículos Autônomos. 2. Internet das Coisas. 3.
        Segurança de Redes. I. Branco, Kalinka R. L. J. C.,
        orient. II. Título.
```

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Mariana Rodrigues

SEMU - SErvice Management for UAVs

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutora em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Prof. Dra. Kalinka Regina Lucas Jaquie Castelo Branco

USP – São Carlos
Junho de 2022

ACKNOWLEDGEMENTS

To my family, for their unconditional love and support.

To Professor Kalinka R. L. J. C. Branco, for all her mentoring and patience in this journey.

To all my friends, colleagues and collaborating students for all the moments of joy, learning and frustration together.

To the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), for the financial support.

RESUMO

RODRIGUES, M. **SEMU - SErvice Management for UAVs**. 2022. 189 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Veículos Aéreos Não Tripulados (VANTs) tem sido utilizados em várias áreas como agricultura de precisão, monitoramento ambiental, suporte a desastres naturais, entre outros. Com o avanço tecnológico atual, esses veículos estão sendo conectados a outros dispositivos e sistemas como a Internet das Coisas (*Internet of Things* — IoT), permitindo que VANTs sejam capazes de fornecer e consumir serviços sendo umas das coisas inseridas no ambiente. Acredita-se que a Internet das Coisas trará um desenvolvimento sem precedentes à nossa sociedade e criando e impactando aplicações novas e mais poderosas em diversas áreas. Esta integração entre VANTs e a Internet das Coisas também possui desafios. A heterogeneidade intrínseca da IoT em relação a dispositivos, plataformas e protocolos de comunicação faz com que o fornecimento e gerenciamento de serviços sejam mais difíceis de serem feitos na camada de aplicação. Segurança é um dos desafios mais críticos em aplicações conectadas à IoT, sendo algumas vezes inserida após o desenvolvimento “principal”, se for inserida.

Estre trabalho demonstra como fazer VANTs se conectarem na IoT usando a SEMU — SErvice Management for UAVs. A SEMU inclui (a) segurança de comunicação, incluindo segurança sensível a contexto; (b) gerenciamento de serviços (registro, descoberta, requisição e consumo) para tipos e requerimentos de dados diferentes e (c) gerenciamento autônomo de tarefas que permite que o VANT cumpra sua missão sem intervenção humana. A funcionalidade e viabilidade da SEMU é demonstrada utilizando-a em aplicações simulando cenários possíveis no mundo real, como um serviço de entregas utilizando VANT e uma aplicação de coleta de dados de sensores utilizando VANTs. Pesquisas futuras poderão utilizar a SEMU e seu protótipo para investigar e avançar o estado da arte em aplicações de VANTs conectadas à IoT e seus mecanismos de segurança.

Palavras-chave: Internet das Coisas, VANTs, Serviços, SEMU.

ABSTRACT

RODRIGUES, M. **SEMU - SErvice Management for UAVs**. 2022. 189 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Unmanned Aerial Vehicles (UAVs) are being employed in many fields such as precision agriculture, environmental monitoring, disaster response support, and others. Following the technological development, these vehicles are now being connected to other devices and systems such as the Internet of Things (IoT), allowing them to provide and consume services as one of the things in the environment. The IoT is believed to bring new unprecedented development to our society and will impact and enable new, more powerful applications in many areas. This integration does not come without challenges. The intrinsic IoT heterogeneity regarding devices, platforms, and communication protocols makes the task of service provision and management much harder to be done in the application layer. Security is one of the most critical challenges in IoT-enabled applications, being sometimes put in after the "main" development is done, if at all.

This work demonstrates how to leverage UAVs to be connected in the IoT using SEMU — SErvice Management for UAVs. SEMU comprises (a) communication security, including context-aware security; (b) service management (registration, discovery, request and consumption) for different types and data requirements and (c) autonomous task management that allows the UAVs to perform their mission without human intervention. Its functionality and feasibility is demonstrated by its use in different applications simulating real-life scenarios such as a UAV-enabled delivery service and a UAV-enabled data collection from IoT sensors. Future research will be able to leverage on SEMU and its prototype to further investigate and advance the state-of-art regarding IoT-enabled UAV applications and their security mechanisms.

Keywords: Internet of Things, IoT, UAVs, Services, SEMU.

LIST OF FIGURES

LIST OF SOURCE CODES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
Cloud–SPHERE	**S**ecurity and safety **P**latform for **HE**te**R**ogeneous syst**E**ms connected to the **Cloud**
CSU	Central Security Unit
HAMSTER	**H**e**A**lthy, **M**obility and **S**ecurity based data communication archi**TE**ctu**R**e
NCI	Node Criticality Index
NP	Navigation Phases
PSI	Perceived Security Index
PoC	Proof-of-Concept
SEMU	Service Exchange Management Unit
SEO	Service Exchange Operator
SMU	Safety Management Unit
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
ATM	Air Traffic Management
BLE	Bluetooth Low Energy
BLoS	Beyond Line of Sight
C2	Command and Control
CA	Certificate Authority
CC	Cloud Computing
CoAP	Constrained Application Protocol
DAA	Detection And Avoid
DaaS	Drone-as-a-Service
DDS	Data Distribution Service Protocol
DoS	Denial of Service
EASA	European Aviation Safety Agency
EC	Edge Computing
FAA	Federal Aviation Administration
FANET	Flying Ad-hoc Network
FC	Fog Computing
GCS	Ground Control Station

GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IoT	Internet of Things
IPv6	Internet Protocol version 6
JARUS	Joint Authorities for Rulemaking of Unmanned Systems
LoS	Line of Sight
LWPA	Low Power Wide Area
MANET	Mobile Ad-hoc Network
MC	Mobile Computing
MQTT	Message Queuing Telemetry Transport Protocol
NASA	National Aeronautics and Space Administration
NB-IoT	Narrow-Band IoT
NFC	Near Field Communication
OS	Operating System
QoS	Quality of Service
QR code	Quick Response code
REST	REpresentational State Transfer
RFID	Radio Frequency Identification
RPAS	Remotely Piloted Aerial Systems
RPL	Routing Protocol for Low Power and Lossy Networks
RSSI	Received signal strength indication
SEMU	**SE**rvice **M**anagement for **U**AVs
SOA	Service-Oriented Architecture
SORA	Specific Operations Risk Assessment
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
UAVaaS	UAV-as-a-Service
UTM	Unmanned Traffic Management
VANET	Vehicular Ad-hoc Network
VLoS	Visual Line-of-Sight
WSN	Wireless Sensor Network
XMPP	Extensible Messaging and Presence Protocol
UUID	Universally Unique Identifier

CONTENTS

CHAPTER

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs), popularly known as *drones*, are vehicles without a crew that operate autonomously or remotely. Due to technological advances, UAVs have gained more computational power and can now be employed in many applications such as precision agriculture, infrastructure and environmental monitoring, surveillance, disaster control and response, among others. Usually, UAVs operate together with a Ground Control Station (GCS) and the communication links, composing an Unmanned Aircraft System (UAS) (VALAVANIS; VACHTSEVANOS, 2015).

Our world is now witnessing an ascending interest in connecting different cyber-physical systems in order to achieve better results through collaboration and information sharing. A very representative example of this trend is the Internet of Things (IoT) paradigm. The IoT has been gaining attention from the academia, industry and the media due to its potential to improve our productivity, efficiency and quality of life. Despite the increasing interest, the IoT is yet to gain a formal definition (WHITMORE; AGARWAL; Da Xu, 2015; MEDDEB, 2016). Most likely, the IoT will be formed by ubiquitous everyday objects or *things* able to communicate with other devices to share information or coordinate actions in order to achieve a common task or goal or offer services for different types of applications (ATZORI; IERA; MORABITO, 2010; ITU-T Y.2060, 2012; AL-FUQAHA *et al.*, 2015; WHITMORE; AGARWAL; Da Xu, 2015). It is believed that the IoT will have a significant impact in our world, contributing with the improvement of the quality of life and the economy's growth (AL-FUQAHA *et al.*, 2015).

As with many technologies, the potential of new or improved applications by integrating UAVs into the IoT to provide UAV-enabled services is being studied. For this to happen, UAVs must be seamlessly integrated into the IoT and be able to provide and consume services so intelligent decisions are made autonomously or according to a client request. Many UAV-enabled services are envisioned in many areas (PAKROOH; BOHLOOLI, 2021), making this integration a great opportunity for better technology development.

In the IoT, a UAV and all of its modules will be a *thing* able to connect, exchange data and collaborate with other things in the system, generating a set of always-available *services* which can be discovered and consumed by clients (MIORANDI *et al.*, 2012). The presence of Cloud/Edge/Fog computing in the IoT also allows the UAVs to promptly distribute their telemetry and captured data, as well as to use the remote storage and processing capabilities to aggregate, analyze and distribute data, as well as to control and/or manage systems (AL-FUQAHA *et al.*, 2015; BOTTA *et al.*, 2016; CAVALCANTE *et al.*, 2016).

The IoT is based on communication and collaboration, so a lot of information will travel through its numerous connections, making communication security and privacy major challenges to be tackled. However, given the device heterogeneity intrinsic to the IoT, it is highly unlikely one single security solution can be broadly utilized. In this case, a flexible and configurable solution that allows UAVs to provide services to other “things” in the IoT is highly desirable.

1.1 Motivation

As previously discussed, UAVs become a *thing* when they are integrated in the IoT infrastructure. UAV resources are available to clients/users through *services*, whose availability will depend on contextual information such as location, battery levels, available payload, and others (MAHMOUD; MOHAMED; AL-JAROODI, 2015). Since service clients can be either human or other machines, it is crucial to correctly identify all resources and services available (MAHMOUD; MOHAMED; AL-JAROODI, 2015). Even though some works such as (MAHMOUD; MOHAMED, 2015) and (ALWATEER; LOKE; RAHAYU, 2018) bring some service classification, it would be highly desirable that all services were properly described by a standardized set of definitions. Also, UAVs need a way to allow operations such as service advertisement, autonomous discovery, request and provision so the system is self-managed.

Security is a challenge in any communication system, and it is considered crucial for IoT acceptance from the general public and the widespread adoption of the technology. Regular network solutions are not suitable for IoT systems, impeding applying a single security solution on the entire system. Most likely, different security mechanisms will be available for the devices, and a negotiation on security measures will be necessary before IoT devices exchange data.

Previous studies have proposed connecting UAVs into the IoT or another similar system in order to provide services. However, very few papers address communication security, even though it is recognized as an important challenge.

One of the proposed solutions was the **He**~~~~**A**lthy, **M**obility and **S**ecurity based data communication archi**TE**ctu**R**e (HAMSTER). HAMSTER was envisioned to deal with different aspects in Unmanned Vehicles such as security, safety and mobility and was made available as a reference model in (PIGATTO, 2017).

This thesis presents the **SE**rvice **M**anagement for **U**AVs (SEMU), an evolution of HAMSTER. The architecture was improved, expanded and gained new functionality in order to enable IoT-connected UAVs to securely provide services in the network they are in. Besides that, the first software prototype was developed, making HAMSTER leave the platonic plane and come to life to enable further research and development in the area.

1.2 Research question and hypothesis

The research question of this theses is: can a UAV be integrated into the Internet of Things ecosystem and be able to autonomously and exchange services with other devices though a secure communication?

In order to answer this question, the following hypothesis was defined: a UAV can be integrated into a communicating system like the Internet of Things and exchange services with other devices through a communication architecture that provides support for communication security and service operations.

Based on the hypothesis, there are some other challenges to be addressed: (i) how to deal with device heterogeneity that translate to different network protocols and hardware capabilities; (ii) how to exchange services that can be very different in nature; (iii) how to provide security seamlessly to highly-diverse hardware and applications.

1.3 Thesis Organization

This thesis consists of a research papers collection written during this PhD, as follows:

1. RODRIGUES, M.; PIGATTO, D. F.; FONTES, J. V. C.; PINTO, A. S. R.; DIGUET, J.-P.; BRANCO, K. R. L. J. C. UAV Integration Into IoIT : Opportunities and Challenges. **International Conference on Autonomic and Autonomous Systems (ICAS)**, IARIA, p. 6, 2017.
2. RODRIGUES, M.; PIGATTO, D. F.; BRANCO, K. R. L. J. C. Cloud-SPHERE: A Security Approach for Connected Unmanned Aerial Vehicles. In: **International Conference on Unmanned Aircraft Systems (ICUAS)**. Dallas, EUA: IEEE, 2018. p. 769–778. ISBN 978-1-5386-1354-2. Available: <https://doi.org/10.1109/ICUAS.2018.8453302>.
3. ____. Navigation phases platform: Towards green computing for uavs. In: **2018 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2018. p. 01171–01176. Available: <https://doi.org/10.1109/ISCC.2018.8538713>.
4. RODRIGUES, M.; BRANCO, K. R. L. J. C. Uavs at your service: Towards iot integration with hamster. In: **2019 International Conference on Unmanned Aircraft Systems (ICUAS)**. [s.n.], 2019. p. 856–865. Available: <https://doi.org/10.1109/ICUAS.2019.8797790>.

5. ____. Cloud–SPHERE: Towards Secure UAV Service Provision. **Journal of Intelligent & Robotic Systems**, v. 97, p. 249–268, 2020. ISSN 1573-0409. Available: <https://doi.org/10.1007/s10846-019-01046-6>.
6. ____. Context-aware operation for unmanned systems with hamster. In: **2020 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2020. p. 1–6. Available: <https://doi.org/10.1109/ISCC50000.2020.9219657>.
7. ____. Enabling uav services in the iot with hamster. In: **2021 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2021. p. 1–6. Available: <https://doi.org/10.1109/ISCC53001.2021.9631461>.
8. ____. Providing UAV Secure Services in the Internet of Things with HAMSTER: a Tutorial. [Manuscript submitted for publication]. 2022.

Since the idea evolved and models changed during the course of this work, it would not make sense to attach them verbatim. Therefore, their text and references were adapted and combined to better fit a thesis format, as follows:

- Chapter 2 presents background information on Unmanned Aerial Systems (UASs) and the Internet of Things based on paper #8.
- Chapter 3 discusses the challenges of integrating UAVs into the Internet of Things and present some opportunities studied in the field, based on papers #1, #3, #4, #5, #6 and #8;
- Chapter 4 presents the concept of SEMU and its components, including their model in a high-level approach, based on papers #2, #3, #4, #5, #6, #7 and #8;
- Chapter 5 presents the details of the latest prototype version, including its modeling and operation, based on paper #8;
- Chapter 6 presents the obtained results of this work, including securing a MQTT application and an entire SEMU-enabled application with services, based on papers #2, #3, #5, #6, #7 and #8;
- The thesis finished in Chapter 7 by presenting the conclusions and research opportunities derived from this thesis.

CHAPTER 2

LITERATURE REVIEW

Our world is now witnessing an ascending interest in connecting different cyber-physical systems in order to achieve better results through collaboration and information sharing. A very representative example of this trend is the Internet of Things (IoT) paradigm. The IoT has been gaining attention from the academia, industry and the media due to its potential to improve our productivity, efficiency and quality of life.

Popularly known as drones, Unmanned Aerial Vehicles (UAVs) are gaining popularity in both military and civilian segments, with the technology being successfully used in applications such as surveillance, disaster control and response, infrastructure and environmental monitoring, among others. UAVs are natural candidates to integrate the IoT and with it be able to deploy more complex and impacting applications.

Both domains have intrinsic characteristics that influence aspects such as autonomy, connectivity and performance — those in turn determine how applications can be deployed. In this Chapter, we present background information on both Unmanned Aerial Systems (UASs) and the IoT, including new paradigms that are commonly associated with boosting IoT performance: Cloud / Fog / Edge Computing.

2.1 Unmanned Aerial Systems

Unmanned Aerial Vehicles (UAVs) are powered vehicles autonomously or remotely operated. The classification of those vehicles will depend on the used metrics, which can be based on weight, size, collision risk, operation altitude, payload, and others (KIM; JO; SHRESTHA, 2014). Authors from (DALAMAGKIDIS, 2015) provide an initial glimpse on UAV classification.

UAVs are usually a part of an Unmanned Aerial System (UAS), a more complex system composed by other elements such as (VALAVANIS; VACHTSEVANOS, 2015):

- GROUND CONTROL STATION (GCS): The center of man-machine interface, it can be airborne or on the ground. It uses the communication links so that the UAV operator can direct the vehicle operation and supervise status or payload data. Usually, it is also the interface to other systems.

- UNMANNED VEHICLE: The type of vehicle used is determined by the application — each one has different demands in terms of payload, response time, mobility, level of autonomy and coverage area (HAYAT; YANMAZ; MUZAFFAR, 2016).

- PAYLOAD: What the UAV is transporting is also dependent on the mission and/or application the UAV is deployed for. It can be for example cameras for surveillance applications, goods or medicines in delivery applications or weapons in military applications.

- NAVIGATION SYSTEMS: Essential for UAVs to flight autonomously, navigation information can be acquired by combining Global Positioning System (GPS) sensor and inertial information acquire by an Inertial Measurement Unit (IMU). For non-autonomous operation, other methods such as radar tracking, radio tracking or direct reckoning can be deployed.

- LAUNCH, RECOVERY AND RETRIEVAL EQUIPMENT: Will be required for UAVs not able to flight vertically or without access to a suitable runaway for launching/landing. Retrieval equipment is necessary if a UAV cannot be carried.

- COMMUNICATION LINKS: Because they do not have an on board operator, UAVs require a robust and effective communication for them to communicate with other manned and/or unmanned aircraft or vehicles, with the GCS and with traffic control authorities (VALAVANIS; VACHTSEVANOS, 2015).

Historically, UAVs were mainly adopted in military operations such as transportation, intelligence gathering, target attacks and combat support (MOTLAGH; TALEB; AROUK, 2016). However, in recent years UAVs have also been used in civilian applications such as aerial photography, environmental, traffic and infrastructure monitoring (KANISTRAS *et al.*, 2015; ZHOU *et al.*, 2016; YU *et al.*, 2017; BOLLA *et al.*, 2018), agriculture (CHEBROLU; LABE; STACHNISS, 2018; PERZ; WRONOWSKI, 2018), search and rescue (AL-KAFF *et al.*, 2017; PEREZ-GRAU *et al.*, 2017) and surveillance. Also, they can be used in hazardous environments and therefore be helpful in disaster-response scenarios (GOMEZ; PURDIE, 2016; TURNER; HARLEY; DRUMMOND, 2016).

Due to their numerous applications, UAVs will most likely be a part of our daily lives, but not without challenges.Among them, we highlight and discuss (a) the UAS insertion into the shared airspace ensuring the safety and privacy of the general population and (b) the establishment of reliable communication between mobile, resource-constrained vehicles and (c) securing the vehicles and their communication.

2.1.1 Airspace Integration, Safety and Privacy

Integrating UAS into the Airspace must be done in an organized and controlled manner without affecting the current air traffic (MOTLAGH; TALEB; AROUK, 2016). The lack of proper regulation can be considered one of the biggest obstacles to UAV development in civilian areas (SAHINGOZ, 2014) and the increasing number of UAVs will impact how air traffic control is conducted (ALTAWY; YOUSSEF, 2016), and a reliable Unmanned Traffic Management (UTM) is necessary for low level airspace (DAVIES *et al.*, 2021). Among others challenges, air control agencies will need to address:

- RESTRICTED DETECTION AND AVOID (DAA): UAV operators have a restricted awareness of the vehicle surroundings, more so if operating Beyond Line of Sight (BLoS). (ALTAWY; YOUSSEF, 2016). DAA operating requirements need to be developed so UAVs maintain a safe distance from other manned or unmanned aircraft and prevent collisions (FAA, 2018a). Particularly in cases of smaller UAVs not radar-detectable, air controllers will not be able to provide aircraft separation to manned aircraft. (MARSHALL, 2016).
- C2 LINK LATENCY AND MAINTENANCE: Except for fully autonomous UAVs, the Command and Control (C2) link is critical. It allows the operator to control the UAV and provides telemetry (such as altitude, airspeed and position) information. The connection made by C2 link should be active for as long as possible, and research is required for emergency actions for when the link is lost (FAA, 2018a). Also, C2 link is vulnerable to radio or electromagnetic interference, radio or satellite system failure (MARSHALL, 2016), and its latency may retard the UAV response to traffic control requests, which can compromise systems with real-time requirements (ALTAWY; YOUSSEF, 2016).
- STANDARDIZATION: Standards on efficiency, safety and reliability are required for UAVs to operate within the Airspace. According to the Federal Aviation Administration (FAA) in the United States, those standards will vary depending on the operation nature and environment, the aircraft itself and the pilot qualification (FAA, 2018a).
- SPECTRUM MANAGEMENT: Spectrum regulation is an important issue to be considered (MOTLAGH; TALEB; AROUK, 2016) for the UAV not to disturb the communication of current air traffic. Frequency bands for UAS payload communication must be allocated (FAA, 2018a).
- AIRSPACE MANAGEMENT: The Airspace Management needs adaptation to operate with UAVs (MARSHALL, 2016). When managing traffic, air traffic controllers must consider the vehicle characteristics (a smaller UAV will be more affected by turbulence than a bigger one), a challenging task when the high heterogeneity is taken into account. At the same time, UAV operators have to be trained to answer to air traffic controllers in a separate interface while piloting (ALTAWY; YOUSSEF, 2016). In this regard, a traffic management system could provide services such as congestion management, path planning

and rerouting or geofencing, accelerating the use of UAVs in civilian applications (KIM; JO; SHAW, 2015).

Authorities like the FAA in the United States and the European Aviation Safety Agency (EASA) are already taking actions regarding UAS regulation. In the USA, 14 CFR part 107 (FAA 14 CFR PART 107, 2016) contains the final regulation for small drones, which flight in low altitude airspace, i.e., below 400 feet. The vehicle is required to operate within Visual Line-of-Sight (VLoS) during daylight twilight hours if equipped with anti-collision lights. Since the existing Air Traffic Management (ATM) system do not provide services in this airspace, both FAA and National Aeronautics and Space Administration (NASA) are collaborating with industry to develop an Unmanned Traffic Management (UTM) (FAA, 2018b), which is separate but complementary to the FAA ATM system (FAA, 2018a). UTM can offer services like flight planning, communications, command and control, and others. In this system, UAS must be registered before operating in the airspace (FAA, 2018b). In Europe, EASA has released a ‘Prototype’ for UAS regulation (EASA, 2016). For more details, the work in (STÖCKER *et al.*, 2017) presents an investigation of UAV regulations on global scale, and authors from (DAVIES *et al.*, 2021) review UTM framework proposals.

A crucial point for technology acceptance by the general public is to guarantee the population safety and privacy (MOTLAGH; TALEB; AROUK, 2016). Privacy invasion is a heavily debated topic yet to be resolved (KIM; JO; SHAW, 2015), aggravated by the low cost and small size of many available UAVs (STÖCKER *et al.*, 2017). One of the greatest challenges is the fact that the majority of acquisitions are made by hobbyists or non-certified operators with no knowledge of the rules, who can create unnecessary risks for other people or manned aircraft. UAV owners may also have their own privacy violated — in (GABRIELSSON; BUGEJA; VOGEL, 2021), authors demonstrate a deauthentication attack targeting a commercial drone that was able to recover personal information (audio, video, and location) stored in the UAV.

However, it would be virtually impossible for any regulating agency to identify and pursue all malefactors without any market regulation. For this, registration and collaboration with law enforcement is necessary. The FAA has a registration process since 2015 and it allows law enforcers and regulators to identify an operator more quickly. Also, the registration process is viewed by the agency as an opportunity to instruct operator on safety guidelines, emphasizing the operator accountability (FAA, 2018a).

2.1.2 Reliable Communication

Communication is a cardinal aspect of UAV applications (BEKMEZCI; SAHINGOZ; TEMEL, 2013), with different types of links — such as command and control, telemetry and payload — being employed between the UAVs and the infrastructure (HE; CHAN; GUIZANI, 2017). Communication is even more critical when multiple UAVs are used in a mission — a growing occurrence in civilian applications (GUPTA; JAIN; VASZKUN, 2016) — which can be homogeneous or heterogeneous regarding characteristics like storage, processing and sensing capabilities. Homogeneous-UAV applications are easier to develop, but the use of heterogeneous UAVs can show itself as a cost-effective way to perform complex missions that require different capabilities of each vehicle (MOHAMED; AL-JAROODI, 2013).

The use of multiple UAVs in a mission presents some other advantages. First of all, acquisition and maintenance costs are reduced for small UAVs compared with bigger ones, and the missions can be completed faster due to task parallelization (SINGH; VERMA, 2018; ROVIRA-SUGRANES *et al.*, 2021). Multi-UAV systems also scale more easily to wider coverage and have higher survivability, since there are other vehicles to complete the mission if one of them fail. Finally, there is a very small radar cross-section, crucial for military operations (BEKMEZCI; SAHINGOZ; TEMEL, 2013).

In a single-UAV system, the communication is made between the vehicle and the control station by a direct link or a Line of Sight (LoS) communication. In a multi-UAV system, the communication can be established in two different forms: centralized or decentralized. In centralized or infrastructure-based communication, the GCS or a satellite acts as a central node to which all UAVs are connected and through which all inter-UAV communication is routed. In this setup, the vehicle must be equipped with an expensive hardware to communicate, and the operation area is limited to the communication coverage of the infrastructure — a UAV cannot operate if it is not connected. In decentralized communication, UAVs communicate directly or using other UAVs as relays, in an ad-hoc network. In this setup, the mission target area is increased, since the communication among vehicles and the GCS can be performed through other vehicles in communication hops (BEKMEZCI; SAHINGOZ; TEMEL, 2013; MOTLAGH; TALEB; AROUK, 2016; CHRIKI *et al.*, 2019).

An ad-hoc network is formed by a collection of nodes also functioning as routers, allowing the creation of a temporary network with no fixed topology or centralized administration that can operate stand-alone or connected to the Internet. When those nodes are mobile, the network is called a Mobile Ad-hoc Network (MANET). For vehicular applications, the nodes are embedded in vehicles, forming a Vehicular Ad-hoc Network (VANET). If the nodes are UAVs, they compose a Flying Ad-hoc Network (FANET), a special form of MANET or VANET (BEKMEZCI; SAHINGOZ; TEMEL, 2013). The work in (CHRIKI *et al.*, 2019) presents the FANET state of the art at the time of its publication, discussing FANET surveys, communication issues and requirements, mobility models and security schemes, as well as open challenges.

FANETs allow UAVs to operate in a larger area — if a UAV can no longer connect to the infrastructure, it can use other UAVs to relay the message to its destination, enhancing the reliability of multi-UAV systems. Also, FANETs in many cases require lighter communication hardware, increasing the system endurance (BEKMEZCI; SAHINGOZ; TEMEL, 2013). However, FANETs have some intrinsic characteristics that make them different from other ad hoc networks (BEKMEZCI; SAHINGOZ; TEMEL, 2013; GUPTA; JAIN; VASZKUN, 2016;

SINGH; VERMA, 2018; CHRIKI *et al.*, 2019):

- NODE MOBILITY: Node mobility in FANETs is much higher compared to MANETs and VANETs, which could influence the stability of communication links and cause nodes to leave and re-enter the network. Because of that, the network routes between the nodes change frequently.
- MOBILITY MODEL: While nodes within a VANET follow a determined path following streets and highways, FANET nodes have diverse mobility models, since they can be either static in a hovering mission or moving at a very high speed. Mobility impacts FANET performance, so the application mobility model should be chosen carefully so the task is completed efficiently.
- NODE DENSITY: Since FANET nodes are usually spread out, FANETs present lower node density (average number of nodes in a given unit area) then MANETs or VANETs, which could lead to network partitioning.
- DYNAMIC TOPOLOGY: Due to high node mobility and the not always predictable mobility model, FANET topology changes more frequently than in other ad-hoc networks. Any UAVs entering or leaving the network, as well as disruption and reestablishment of communication links, trigger a topology update.
- PLATFORM CONSTRAINS: UAVs have many resource constrains that impact on their design and applications. First of all, UAVs have limited payload capability regarding size and weight. Heavier payloads mean higher battery consumption and shorter flight time. Limited battery also influence UAV applications, whose algorithms for communication, security and control must be energy efficient and fit to meet UAV computational constrains.

These characteristics have a great impact on a FANET operation. A multi-UAV network need to (a) be robust to tolerate delays, (b) have a high adaptability to deal with frequent topology changes, which requires changes at the MAC and network layers, (c) provide efficient data delivery even with link disruption or outages and (d) employ energy saving mechanisms at both platforms and network layers (GUPTA; JAIN; VASZKUN, 2016; SINGH; VERMA, 2018). In FANETs, there are many challenges to be studied, such as:

- UAV MOBILITY AND PLACEMENT: Where the UAVs are placed is a major research concern in FANET (SINGH; VERMA, 2018). Since UAVs have limited payload, different sensors are likely to be carried by different UAVs. If the positioning of UAVs is not optimized, repeated data from the same area can be acquired.The use of mobility models designed for UAVs may be helpful to study this problem and can affect FANET performance significantly. However, currently mobility models do not reproduce UAV behavior realistically, and a more precise simulation of UAV motion in a practical situations is needed (CHRIKI *et al.*, 2019). The work in (BUJARI; PALAZZI; RONZANI, 2017) analyze possible mobility models for typical UAV applications such as search and rescue, traffic and urban monitoring, reconnaissance and patrolling, agricultural management, environmental sensing and relaying network in order to provide guidelines to FANET research and simulation.

- ROUTING: The frequent topology changes that happen in a FANET present a difficult challenge for data routing. Routing protocols have to be able to work with high mobile nodes and to update routing tables dynamically as the network topology changes (SINGH; VERMA, 2018). There are many FANET routing protocols, classified into static, proactive, reactive, hybrid or position-based. Papers (SAHINGOZ, 2014; FAN; CAI; LIN, 2017; KHAN *et al.*, 2017; OUBBATI *et al.*, 2017; CHRIKI *et al.*, 2019; ROVIRA-SUGRANES *et al.*, 2021) survey FANET routing protocols and authors from (BILAL; KHAN, 2017) evaluated combinations of four mobility models with three different routing protocols for FANETs, analyzing packet delivery ratio, average end to end delay and routing overhead.

- LINK MAINTENANCE: Given the node mobility and speed in a FANET, it is not always possible to maintain a stable link, making delays and disconnections a common occurrence in a FANET and the communication link intermittent (GUPTA; JAIN; VASZKUN, 2016). Link quality variations influence the MAC layer robustness in a FANET. Using directed antennas is believed to be a promising technology, although estimating UAV location is a challenge (BEKMEZCI; SAHINGOZ; TEMEL, 2013). How to keep acceptable send-and-receive signal strength (Received Signal Strength Indication – RSSI) and how to deal with congestion when multiple UAVs try to connect with the same GCS are other challenges (MOTLAGH; TALEB; AROUK, 2016).

- QUALITY OF SERVICE (QOS): reliable communication is necessary to maintain QoS. FANETs transmit diverse types of data with different real-time requirements. The network need to satisfy QoS requirements such as delay, packet loss and jitter compromising the already constrained UAV resources as little as possible (SAHINGOZ, 2014; CHRIKI *et al.*, 2019).

2.1.3 Secure Communication

Communication security is one of the most critical issues related to UAVs. Multi-UAV systems extends the attack surface, and node mobility also impacts security with frequent network topology changes and link inconsistency.

UAS design and development must take security issues into account so the system can answer autonomously and dynamically to attacks or failures, whether accidental or intentional. Conventional security mechanisms are not efficient for real-time transmission required by UAVs, making distributed solutions that do not depend on the GCS and do not compromise UAV resources or network performance necessary (HE; CHAN; GUIZANI, 2017; BEKMEZCI; ŞENTÜRK; TÜRKER, 2016).

Regarding a UAS, the following security requirements apply (ALTAWY; YOUSSEF, 2016; HE; CHAN; GUIZANI, 2017):

- AVAILABILITY: all elements of a UAV must perform their tasks without any disruption during its operation, even if the aircraft is under attack.

- CONFIDENTIALITY: Communication between UAVs or between the UAV and the infrastructure can not leak information to unauthorized parties.
- INTEGRITY: In UASs, integrity can refer to *information integrity* or *system integrity*. Information integrity ensures that exchanged data in communication links, such as mission data, telemetry and GPS signals have not been altered. System integrity ensures the authenticity of software and hardware components.
- ACCOUNTABILITY: UASs need accountability mechanisms to ensure non-repudiation. A digital signature algorithm can bind the action to an entity, and logging procedures allow action tracking.
- AUTHENTICATION AND ACCESS CONTROL: Identification and authentication of nodes are necessary for secure communication. When a node is inserted in the network, it must be identified and authenticated before communicating with other nodes within the network. Access control policies are necessary to prevent unauthorized personnel from accessing sensitive data and/or control operations.

According to (YAACOUB *et al.*, 2020), these essential requirements can be met through the AAA security: *Authorization* to the personnel controlling the UAV, strong *Authentication* to ensure the operator identity and *Auditing/Accounting* to track and ensure UAV owners are held accountable in case of malicious activities.

All UAS components present vulnerabilities that can be exploited and result in a security breach or a system control loss. UAVs rely on different systems (navigation, collision avoidance, sensing) to operate. If any of them is compromised or tampered with, the UAV can be lost and accept malicious commands. Also, UAV systems that depend on external plain signals such as GPS navigation or collision avoidance can receive falsified data and alter the way the aircraft operates (HARTMANN; STEUP, 2013). The GCS must be protected against software threats such as malware and hardware tampering. Communication links in a UAS can be employed in many ways, and each one has its own set of vulnerabilities.

Next, the main threats to a UAS are discussed (ALTAWY; YOUSSEF, 2016; TAN *et al.*, 2020). For a more complete read on attack taxonomy, the reader can refer to (YAHUZA *et al.*, 2021).

2.1.3.1 GPS Jamming or Spoofing

UAV navigation depends heavily on Global Navigation Satellite System (GNSS), being the Global Positioning System (GPS) the most common option. Without this signal, a UAV may be susceptible to Denial of Service (DoS) and/or errors in measurements that can lead the vehicle to crash or diverted from its course. GPS signals can be either *jammed* — when a stronger signal jams the channel and make the receiver unable to distinguish the original GPS signal — or easily *spoofed* — when a faked signal injects false positioning information to the UAV. The lack of

authentication in civil GPS signals makes it easy for its falsification with low-cost equipment and the GPS receiver unable to discern if the received signal is authentic or not (KHAN; MOHSIN; IQBAL, 2021).
COUNTERMEASURES: Authenticated GPS signals could prevent GPS spoofing. Such strategy, however, would require changes in the entire satellite system. Surveillance of GPS signal for sudden changes in signal strength and traveling time can give indications of signal spoofing. In order to counter GPS jamming, alternative navigation methods such as vision and/or inertial navigation systems can be employed in the absence of GPS signals.

2.1.3.2 Signal Jamming or Spoofing

It is expected UAVs will emit broadcasts with its own position and velocity to be used by other aircraft in their collision avoidance system. Like GPS, this broadcast would be open and therefore easily jammed or spoofed, which could lead to UAV collisions or insertion into unfriendly territory. Spoofing UAV telemetry and video feeds can influence operator commands and compromise the system operation. Jamming GCS control signals can induce the UAV into an emergency protocol and impede mission realization. With GCS control signal spoofing, the adversary can take control of the aircraft.
COUNTERMEASURES: UAV signal authenticity verification is necessary to ensure reliable information. In the case of jammed GCS control signals, UAV response needs to be carefully planned so it will not act unpredictably if others signals, such as GPS, are also jammed. In the case of a fixed GCS location-based authentication can be used to mitigate spoofing.

2.1.3.3 Information Injection

Without proper identification and authentication schemes, a malicious user can pose as a trusted UAV or UAV module and inject the system with falsified information, destabilizing UAVs and compromising UAS operation.
COUNTERMEASURES: Authentication of all UAVs modules and of the UAVs within a FANET is necessary to ensure only reliable information is being received. Also, comparing the expected behavior/data from actual provided information from modules can identify discrepancies and identify possible falsified data.

2.1.3.4 Malicious Hardware or Software

Both GCS and UAV systems are exposed to hardware and software trojan. GCS can be infected by the actions of an unsuspecting operator or through a connection to an external system or the Internet itself. Hardware trojan can be inserted in chips during fabrication and disable security mechanisms or provide a back door for adversaries.
COUNTERMEASURES: Intrusion detection systems, antivirus software, firewalls, and other

policies can mitigate the threat of software trojan and malware. The use of reliable hardware suppliers as well as side-channel analysis minimize the risk of hardware trojan.

2.1.3.5 Unauthorized Disclosure of Communication

Exchanged information between UAVs or between the UAV and the infrastructure needs to be protected against unauthorized disclosure when intercepted.
COUNTERMEASURES: Employing cryptography can ensure data confidentiality and integrity. Elliptic curve cryptography solutions can be used to provide a good security level with less resource consumption than other public-key solutions. However, implementing a Certification Authority functionality in an ad-hoc network is challenging and may fail depending on the employed approach (BEKMEZCI; ŞENTÜRK; TÜRKER, 2016).

2.1.3.6 Denial of Service (DoS)

Both the infrastructure and UAVs can be at risk of DoS. In a centralized communication UAS, a DoS in GCS will leave all UAVs in the system without receiving any control commands. In a decentralized UAS, compromising the GCS availability can impede UAVs to send pertinent mission information to a remote operator or system. Flooding UAVs with random commands can force them into an unexpected state and consequently compromise UAS operation.
COUNTERMEASURES: UAVs and GCS need to implement mechanisms that made them resistant to DoS. Implementing contingency policies for DoS situations can prevent the system from achieving an unexpected state.

2.1.4 Collaborating UAVs

UAS communication has many future directions that are being tackled by the academic community. Artificial Intelligence (AI) use is already being made in some UAS problems such as positioning and detection, autonomous path control, scheduling and resource allocation, security and sensing, and can be used to other problems such as collective intelligence (ZHOU; RAO; WANG, 2020) and communication protocols (ROVIRA-SUGRANES *et al.*, 2021).

Multi-UAV systems make communication a very critical challenge. Newer technologies like 5G and beyond promise to increase connectivity, allowing UAVs to provide information promptly and make fast collaboration with other connected vehicles or devices. Communication security is still a key requirement, and many security mechanisms for UAVs are purposed — from lightweight cryptography to blockchain-enabled confidentiality and integrity solutions.

However, new technologies such AI and blockchain add computation complexity and increase energy consumption. When considering smaller UAVs with computational and power constrains, developed strategies and protocols must be energy-efficient in order to prolong the network lifetime and UAV autonomy (GUPTA; JAIN; VASZKUN, 2016; ROVIRA-SUGRANES *et al.*, 2021).

UAVs can also collaborate in many ways if connected to larger communication infrastructure. Some examples are collecting data from other devices and taking them to a central station, gathering the data itself, and then taking it to a central station and enabling connectivity to arrive in remote areas. All of this is envisioned to happen with the UAVs connected to the Internet of Things (IoT), discussed next.

2.2 The Internet of Things

The Internet of Things (IoT) has been vastly explored over the last few years, becoming a matter of interest for both industry and academia. A formal definition of IoT is yet to be reached (WHITMORE; AGARWAL; Da Xu, 2015), but there is a consensus that IoT will be formed by ubiquitous everyday objects or 'things' that can communicate with other devices to share information, coordinate actions to achieve a common task or goal, or offer services for different applications (ITU-T Y.2060, 2012; AL-FUQAHA *et al.*, 2015; WHITMORE; AGARWAL; Da Xu, 2015). The IoT is expected to have a notable impact in our everyday lives, with significant applications in many areas such as healthcare, smart environments (smart homes, factories, or cities), commercial, industrial, environmental, and infrastructural monitoring (ASGHARI; RAHMANI; JAVADI, 2019; HASSAN *et al.*, 2020).

Since this is a very complex system, it has many different aspects to be considered. In this Section, we provide an overall view of the IoT briefly discussing its elements (Section 2.2.1), requirements (Section 2.2.2), architecture models (Section 2.2.3), communication technologies and protocols (Section 2.2.4), the use of Cloud/Fog/Edge technologies (Section 2.2.5) and security (Section 2.2.6). The reader can refer to IoT surveys such as (AL-FUQAHA *et al.*, 2015; LIN *et al.*, 2017; ČOLAKOVIĆ; HADŽIALIĆ, 2018) and (BANSAL; KUMAR, 2020) for more information and references for particular IoT aspects.

2.2.1 IoT Elements

In this Section, the six main elements required to provide IoT functionality according to (AL-FUQAHA *et al.*, 2015) are discussed: Identification and Addressing, Sensing, Communication, Computation, Services, and Semantics.

2.2.1.1 Identification and Addressing

IoT devices represent the threshold between the digital and the physical world. In the system, each physical entity has a digital representation. This digital-physical operation is only possible with proper device identification and addressing. A physical entity is mapped to a digital entity through an identity, whereas the digital entities reach their physical counterparts through device addressing (NING *et al.*, 2020). Moreover, many security mechanisms on the Internet, such as authentication and authorization, are ID-based, making identity modeling and addressing crucial elements within the IoT system (ZHU; BADR, 2018).

Some identification methods are considered feasible to the IoT system. Radio Frequency Identification (RFID), Quick Response code (QR code) and Near Field Communication (NFC) are examples commonly found in the literature of identity carriers. In RFID technology, Electronic Product Codes (EPC) and Ubiquitous Codes (uCode) are often cited as encoding techniques. The Internet Protocol version 6 (IPv6) is presumed the addressing protocol to ensure unique device addressing, since its address space is believed large enough to comprise the entire system. There are also other tools to aid device identification in the IoT. The survey brought by (NING *et al.*, 2020) compiles different techniques of identity modeling and addressing proposed to the IoT, discuss some of its challenges and presents an identity framework for IoT systems. The work in (ZHU; BADR, 2018) brings a survey on Blockchain based identity modeling and management solution for IoT.

2.2.1.2 Sensing

Sensors are hardware devices that perceive environmental data and transform it into a digital representation, acting as a bridge between the real and the digital world. Sensors in IoT can be smart sensors, actuators, or wearable devices (AL-FUQAHA *et al.*, 2015), which gather data to be analyzed, adding value to IoT services (ČOLAKOVIĆ; HADŽIALIĆ, 2018).

When provided with communication capabilities, those sensors become nodes that can compose a network and transmit their data. Not by chance, Wireless Sensor Networks (WSNs) are considered by many in the literature as an important IoT element. In many cases, the IoT network infrastructure is based on WSNs, which have some benefits such as scalability, dynamic reconfiguration, reliability, small size, low cost, and low energy consumption (LIN *et al.*, 2017).

Those IoT devices are likely to be considered resource-constrained in their majority due to CPU, memory, and power resource limitations. These constraints have a direct impact on code complexity (limited ROM/Flash), size of buffers (limited RAM), computational operations (limited CPU), and operation time (limited power), which in turn interfere with device support for functionality such as identity management, communication, and security (ČOLAKOVIĆ; HADŽIALIĆ, 2018).

2.2.1.3 Communication

The IoT paradigm, constrained devices form a network, becoming constrained nodes in a *Constrained-Node Network*, whose characteristics are impacted by its resource-constrained nodes (BORMANN; ERSUE; KERANEN, 2014). Constrained-node networks require low-power communication protocols and security, as well as memory-efficient routing protocols (BANSAL;

KUMAR, 2020). The IoT network also connects heterogeneous objects, which demands the employ of cross-technology communication devices at some point.

The IoT is envisioned to have different applications. Some of them can be in a peer-to-peer manner, or involve a few devices in a personal network; others may involve various heterogeneous devices in larger networks communicating through different protocols. A communication technology deemed suitable for a particular IoT application might not necessarily be suited to many others. The ability to connect and coexist various devices deploying several communication technologies is the vision behind the IoT. Having an ecosystem of coexisted technologies and devices is what enables the IoT vision of extending communications to anything and anywhere (ELKHODR; SHAHRESTANI; CHEUNG, 2016).

2.2.1.4 Computation

Nowadays, there are in the market two types of IoT hardware: wearable devices to whom only applications can be developed, and embedded systems boards, that allow both hardware and software development. There are various hardware platforms developed for IoT such as Arduino, RaspberyPi, and ESP8266 module. Authors from (SINGH; KAPOOR, 2017) review the characteristics of several open-source hardware IoT platforms and discuss a method for choosing the best possible options for DIYers IoT projects and/or prototyping.

IoT functionality is provided by software platforms and/or middlewares. According to (BUTTERFIELD; NGONDI; KERR, 2016), a middleware is a product that occupies an intermediate position in a computer application, such as between hardware and software (as in an Operating System) or between the Operating System and the application program, particularly in a distributed system. Authors from (NGU *et al.*, 2017) classify IoT middlewares into three types: (1) *Service-based*, which adopts a Service-Oriented Architecture (SOA) and users can integrate diverse devices as services; (2) *Cloud-based*, which enable the users to connect devices and interpret their data and (3) *Actor-based*, which works in a plug-and-play manner and make devices reusable in the network. The paper also performs a comparative analysis of various emerging IoT middleware, as well as points out the research challenges IoT middleware faces. Authors from (RAZZAQUE *et al.*, 2016) make a more complex classification for IoT middlewares: (1) event-based, (2) service-oriented, (3) agent-based, (4) tuple-space, (5) VM-based, (6) database-oriented, and (7) application-specific. The paper relates each category against discussed middleware requirements such as interoperability, context-awareness, resource management, and discovery.

When considering low-end IoT devices, i.e., those closer to the physical world and usually very resource-restricted, there is a need for tailored IoT Operating Systems (OSs), since traditional ones such as Linux are not able to run in such a constrained device. An OS operating in low-end IoT devices requires a small memory footprint, support for heterogeneous hardware, network connectivity, energy efficiency, real-time capabilities, and security (HAHM *et al.*, 2016). There are already some IoT-suited OSs in the market, such as Contiki (Contiki, 2021), RIOT

(RIOT, 2021), FreeRTOS (FreeRTOS, 2021) and TinyOS (TinyOS, 2022), among others, which are compared by some papers. Authors from (SABRI; KRIAA; AZZOUZ, 2017) compares FreeRTOS, Mbed, Contiki, TinyOS, and RIOT regarding their architecture, scheduling and real-time capabilities, programming model, memory footprint, hardware support, network connectivity, protocol support, and energy efficiency. Contiki, TinyOS, and FreeRTOS are surveyed in respect of process, memory, energy, communication, and file management in (MUSADDIQ *et al.*, 2018), which also provides future research directions and challenges in resource management of IoT-suited OSs. Finally, the work from (HAHM *et al.*, 2016) discuss IoT OSs requirements and overviews many OS choices, both open-source and proprietary, and perform a more in-depth study about Contiki, RIOT, and FreeRTOS.

Cloud/Fog Platforms are other important IoT enablers. Cloud Computing (CC) can offer IoT devices virtually unlimited processing and data storing capacity and provide fault tolerance and scalability at low cost, being able not only to aggregate, analyze, distribute and extract meaningful information from collected IoT data but also to remotely manage and control systems. Besides, CC is globally accessible and available, allowing IoT devices to be located and accessed anywhere, anytime. Moreover, CC can concentrate services from different providers and bring them to users in an easy, intuitive way, acting as an intermediate level between IoT devices and the application customer and hiding IoT heterogeneity and the complexities of the system (BOTTA *et al.*, 2016; CAVALCANTE *et al.*, 2016; SINGH *et al.*, 2016).

The Fog Computing (FC) paradigm is an extension of CC to the edge of the network. Its integration with the IoT paradigm is mainly motivated by lower latency and network throughput reduction. FC has a very large number of nodes that can provide localized storage, data filtering, and analysis, support device mobility, and facilitate the Cloud interplay with low-end IoT devices (BOTTA *et al.*, 2016; UHER *et al.*, 2016).

2.2.1.5 Services

The entire IoT concept is based on the provision of *pervasive* or *ubiquitous services*, i.e., services available anytime, anywhere, to anyone (AL-FUQAHA *et al.*, 2015). Generally, IoT services are understood as a way to access the functionality of IoT devices through their virtual representation. IoT services are different from their traditional counterparts due to IoT characteristics such as resource-constrained devices, dynamic network formation, and heterogeneity.

An IoT service can be *atomic* or *composite*. An atomic service is a self-contained, fundamental service that cannot be divided, while a composite service is a combination of atomic or other composite services to provide a more complex functionality (ARELLANES; LAU, 2020). For example, there could be atomic services provided by IoT sensors like location, temperature, and humidity that can be combined in a weather-report composite service. Usually, IoT services can be categorized under four classes (AHMED *et al.*, 2019):

- IDENTITY-RELATED SERVICES: provide identification for IoT devices. They can be active (services that broadcast information, which would require more available power) or passive (services with no power source dependent on external devices or mechanisms).
- DATA AGGREGATION SERVICES: acquire, process, and transmit data from various sensors. If done by an IoT gateway, the service is able to aggregate data from different types of sensors and transmit it in a more friendly manner such as JSON or XML.
- COLLABORATION-AWARE SERVICES: use aggregated data to perform analysis and make decisions. In many scenarios, these services involve both human and machine interactions with the help of an IoT Cloud. This type of service requires a reliable and fast IoT infrastructure, not being provided by low-end devices.
- PERVASIVE SERVICES: These services are the IoT main goal, enabling collaborative-aware services to be provided anytime, anywhere, for everything and everyone. To enable them, the IoT would have to overcome its heterogeneity and follow many steps of service management, such as service modeling, identification, discovery, selection, composition, and orchestration.

IoT *Service Management* process is very complex and faces challenges related to interoperability, scalability, security, and Big Data usage (AHMED *et al.*, 2019). Interoperability challenges require cross-domain solutions and include limited connectivity, semantic incompatibility between data and models, and heterogeneous representation of device data. Scalability challenges include both vertical (IoT device enhancement and updates) and horizontal (addition of devices and/or services) scalability. Security challenges include identity management, authentication and authorization, and vulnerability detection, while Big Data challenges relate to data accuracy, analysis, and visualization.

IoT Services are far from being standardized at this time, with many propositions for service modeling, management, discovery, and Quality of Service (QoS). Some reviews can help the reader to dive deeper into this topic. The paper (AHMED *et al.*, 2019) provides a comprehensive taxonomy on IoT Service Management regarding aspects such as architectural organization, middleware type, run-time service management, security, and applications. It also reviews available commercial service management platforms for IoT and discusses open research challenges. (AZIEZ; BENHARZALLAH; BENNOUI, 2017) makes a study comparing different service discovery techniques in IoT analyzing the service description model, discovery mechanism, adopted architecture, and context-awareness. The work in (ASGHARI; RAHMANI; JAVADI, 2018) provides a Systematic Literature Review on service composition approaches in IoT, providing a taxonomy to classify and discuss them, as well as pointing to future research challenges. Authors from (HAMZEI; Jafari Navimipour, 2018) also present a systematic mapping of IoT service composition techniques, comparing them and discussing future challenges. The work in (ARELLANES; LAU, 2020) investigates service composition mechanisms for IoT regarding their scalability, reviewing and analyzing the fundamental semantics of the composition mechanisms. Authors from (ACHIR; ABDELLI; MOKDAD, 2020) and (POURGHEBLEH;

HAYYOLALAM; ANVIGH, 2020) classify and survey service discovery strategies in IoT.Finally, the study presented in (WHITE; NALLUR; CLARKE, 2017) provides a systematic mapping to investigate QoS strategies in IoT, investigating wherein the network stack are the solutions being implemented, which factors are being taken into account in IoT QoS, and what types of research are being conducted.

2.2.1.6 Semantics

Semantics is considered the brain that can make IoT *smart*, dealing with data heterogeneity, integration, and interpretation. Providing accurate semantic description so data from different sources are understood without ambiguity allows automated communications and interactions, as well as seamless data integration, being a major requirement for overcoming IoT data interoperability. Semantic data also improve IoT services, supporting service/resource search and discovery (SHI *et al.*, 2018). It is common to find the term “Semantic Web of Things” referring to a semantic-enabled IoT application, in which web data become more meaningful and readable by humans, computers, and things (SZILAGYI; WIRA, 2016).

Authors from (SHI *et al.*, 2018) present a three-stage general architecture for *data semantization*, that is, describing IoT data uniformly with fixed mark-ups that can be processed and understood by other entities. The first stage, *Data Collection*, comprises the gathering of sensor identification and measurement data. The second stage, *Data Preprocessing*, takes care of anomalies to provide more reliable data. Finally, the third and last stage is the *Semantic Annotation*, which is the actual insertion of semantics in IoT data based on a chosen *Semantic Model* used to describe the domain. This survey also describes key techniques for those stages used in IoT, presenting possible applications and challenges.

One of the main challenges regarding semantics in IoT is semantic *interoperability*, i.e., enforcing that a piece of data is understood and interpreted unambiguously by different parties. Semantic descriptions, however, are not enough to provide semantic interoperability. There are other obstacles such as data modeling and exchange (provide a standardized way to represent data), data management, service discovery and knowledge representation (NAGOWAH; STA; GOBIN-RAHIMBUX, 2018).

When considering semantic interoperability, there are many studies about the use of *ontologies* for semantic description. An ontology is a model that describes concepts and their relationship concerning a domain (BUTTERFIELD; NGONDI; KERR, 2016), and can be used to describe IoT devices and services so different players understand and process data in the same manner. There are some references regarding the use of ontologies in IoT the reader can refer to for a deeper look. The work in (SZILAGYI; WIRA, 2016) reviews different ontologies proposed to the IoT and discusses the necessary protocol stack for semantics. Authors from (NAGOWAH; STA; GOBIN-RAHIMBUX, 2018) briefly describe more than a dozen ontologies and frameworks for IoT, discussing their limitations and challenges. The paper (SOUZA; FILHO,

2019) performed a systematic mapping regarding the use of ontologies for IoT semantics, analyzing aspects such as representation language, design methodology, and characteristics. Finally, the work in (GYRARD; DATTA; BONNET, 2018) surveys ontology-based software tools for IoT.

2.2.2 IoT Requirements

According to the ITU (ITU-T Y.2060, 2012), the IoT is fundamentally characterized by (a) high interconnectivity (any device can access the Internet); (b) things provisioning services with high constraints; (c) high heterogeneity due to the variety of hardware platforms and network technologies; (d) variable number and state of devices; (e) generation of a large amount of data (that needs to be managed and interpreted). These characteristics translate into high-level requirements that IoT systems need to support regardless of the application (MIORANDI *et al.*, 2012; ITU-T Y.2060, 2012):

2.2.2.1 Device identification

Every device should be identifiable, either by a physical tag or self-identification means. Device connectivity should be based on its identification, and different identification schemes need to be handled seamlessly.

2.2.2.2 Device interoperability

IEEE defines interoperability as "*The ability of two or more systems or components to exchange information and to use the information that has been exchanged*" (IEEE STD 610.12-1990, 1990). IoT systems need to manage and enable the communication of different types of devices, making interoperability a crucial requirement that has many challenges to be faced, as exposed by work from (KONDURU; BHARAMAGOUDRA, 2017). In the IoT context, we can generally refer to two main types of interoperability:

- TECHNICAL INTEROPERABILITY: encompass hardware, network connections/protocols and shared data formatting. This is most likely solved by *adapters*, either converters for hardware and lower-layer communication protocols or frameworks and APIs for upper-layer communication protocols. The work in (AMJAD *et al.*, 2021) surveys different interoperability tools for application layer IoT protocols.
- SEMANTIC INTEROPERABILITY: refers to how the devices reason the shared information. For that, there must be an agreement on how data is structured and tracked (i.e., its origin, destination, quality of the source) so they can be efficiently discovered and correctly reasoned by its recipient. The representation of knowledge has been tackled by the use of Ontologies and, since stakeholders use different representations, there is the need for tools and techniques to reason between different representations — some of them are discussed in (NAGOWAH; STA; GOBIN-RAHIMBUX, 2018).

2.2.2.3 Scalability

The IoT needs to support device identification, network addressing and communication, and information and service management at a very large scale so IoT deployment may transpire as envisioned.

2.2.2.4 Energy efficiency

It is believed that most IoT devices will be very resource-constrained. Even though energy harvesting techniques are being developed, battery replacement is an obstacle for IoT dissemination (GIRI *et al.*, 2017), requiring processing and communication strategies to be energy-aware.

2.2.2.5 Device localization

Being able to locate and track devices automatically enables location-based communications and services need to be compliant with different laws and regulations. Localization technologies will also enable customers to search nearby services and interact with them (AL-FUQAHA *et al.*, 2015). Even though GPS signals are available in outdoor scenarios, their energy consumption makes way for low-power localization options such as Low Power Wide Area Networks (LPWANs) discussed in (MORADBEIKIE *et al.*, 2021). Indoor localization must be done without Global Navigation Satellite Systems (GNSS) and need to be done using wireless technologies — the work in (KHAN *et al.*, 2021) compares different wireless IoT standards for indoor localization.

2.2.2.6 Autonomous operation

This is a very important IoT requirement since it would be unfeasible to deploy any system that demanded regular human intervention. The autonomy relates to (a) networking, which requires self-management, self-configuration, and self-protecting capabilities; (b) service discovery and provisioning, which should be done without an external trigger and therefore enabling device integration and cooperation and plug-and-play capability without affecting the system; and (c) self-diagnosis, so devices can provide better services by monitoring themselves and their physical condition (AL-FUQAHA *et al.*, 2015).

2.2.2.7 Security and Privacy

The core IoT functionality is sensing data, which can contain private information and need to be protected during transmission, aggregation, storage, mining, and processing. There is also the need to integrate different security policies and mechanisms, which are likely to be diverse within the IoT ecosystem.

2.2.3 IoT Architecture Models

The IoT is expected to bring together billions of devices with different hardware capabilities, providing several services with reasonable quality in different communication protocols respecting security and privacy. How to manage such heterogeneity is an important challenge in IoT research that includes how things are going to be organized communication-wise.

When talking about network architecture, it seems possible to connect all IoT devices to a Cloud server that would manage the services and user interaction. For some IoT applications, however, full network support is cost-prohibitive, making it more reasonable to have bridging components to which IoT devices can connect in low-performance communication protocols such as ZigBee, BLE, and others (WANT; SCHILIT; JENSON, 2015).

Therefore, the IoT requires an architecture capable of supporting a huge number of both constrained and resource-rich devices, with heterogeneous applications and service models being delivered in different communication protocols intermittently (MIORANDI *et al.*, 2012; AL-FUQAHA *et al.*, 2015). IoT architecture should be able to provide device management, availability, scalability, Quality of Service (QoS), privacy and security of both users and data (GIRI *et al.*, 2017).

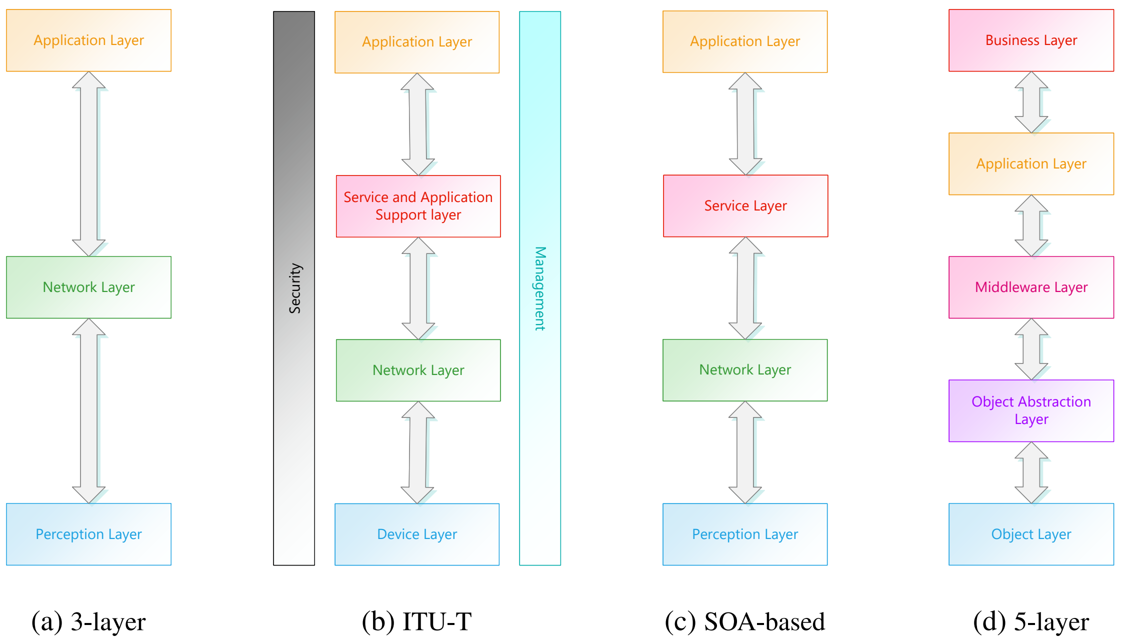
There are many proposed architectures for IoT systems with different layers that do not always overlap, so there is no standardized reference model. The simplest and more basic model is a 3-layer architecture (Figure 1a), which includes 1. a Perception Layer responsible for the interaction with the physical world collecting data and triggering actuators, 2. a Network Layer responsible for transmitting the data over the integrated network and 3. an Application Layer responsible to receive the data and provide services to users(AL-FUQAHA *et al.*, 2015). This model oversimplifies the system and makes it difficult to deal with technology heterogeneity and communication architectures.

The Standardization Sector of the International Telecommunication Union (ITU-T) has proposed a 4-layer model (Figure 1b): the Device layer includes both devices and gateway capabilities; the Network layer includes both networking and transport capabilities; the Service and Application Support layer includes data processing and storage technologies and whatever functionality necessary to support different support to different applications. Finally, the Application layer includes the IoT application. There are also two complementary capabilities sets (Management and Security) that permeate all layers (ITU-T Y.2060, 2012).

SOA-based architectures (Figure 1c) have 4 layers: the Perception layer is responsible for collecting data from physical devices; the Network layer is used for data routing and transmission; the Service layer is responsible for service management and can sometimes be divided into two sub-layers, Service Composition and Service Management; finally, the Application layer is used to support service requests (LIN *et al.*, 2017).

The 5-layer architecture (Figure 1d) is composed of the Objects (or Perception) layer

Figure 1 – IoT architecture proposals.



Source: Elaborated by the author.

representing the physical devices; the Object Abstraction layer responsible for transmitting and/or processing acquires data; the Middleware or Service Management layer connects service with their clients, managing service request and delivery; the Application layer provides services requested by costumers and the Business layer manages the entire system.

2.2.4 IoT Communication Technologies and Protocols

Since its inception, the IoT has been envisioned to be used in different applications with diverse requirements. It is understandable then that many protocols and technologies are used to deploy it, each of them more suitable for a specific scenario and/or environment. In fact, the IoT protocol stack is a point that still has many standardization efforts (MEDDEB, 2016). Table 1 classifies some protocols commonly associated with the IoT into Application, Service Discovery, Network, and Connectivity. It is beyond the scope of this thesis to enter into protocol details, but the reader can refer to general surveys such as (TRIANTAFYLLOU; SARIGIANNIDIS; LAGKAS, 2018; DING *et al.*, 2020) and (SALMAN; JAIN, 2016).

2.2.4.1 Application Protocols

Application protocols are sometimes defined as *messaging protocols*, since their main function is to provide message and/or data exchange between entities. The protocols follow different messaging schemes and each one has different strengths and weaknesses, some of them discussed here. For a better comparison between application protocols, the reader can refer to

Table 1 – IoT communication protocols and technologies.

Application		COAP (IETF RFC 7252, 2014; CoAP, 2022) MQTT (MQTT 3.1.1, 2014; MQTT, 2022) XMPP (IETF RFC 6120, 2011) AMQP (AMQP 1.0, 2012) DDS (DDS 1.4, 2015)
Service Discovery		mDNS (IETF RFC 6762, 2013) DNS-SD (IETF RFC 6763, 2013)
Network		6LowPan (MONTENEGRO *et al.*, 2007) RPL (IETF RFC 6550, 2012),
Connectivity	short-range	NFC (COSKUN; OZDENIZCI; OK, 2013) BLE (BLUETOOTH SIG, 2019) Zigbee (IEEE STD 802.15.4-2015, 2016)
	long-range	LoRaWAN (LoRa Alliance, 2022) NB-IoT (POPLI; JHA; JAIN, 2019; RASTOGI *et al.*, 2020)

Source: Elaborated by the author.

the work in (NAIK, 2017), which makes a relative analysis between MQTT, CoAP, AMQP, and HTTP protocols.

2.2.4.1.1 Constrained Application Protocol (CoAP)

The CoAP is defined by IETF RFC 7252 and provides built-in discovery, multicast support, and asynchronous message exchange (IETF RFC 7252, 2014). The protocol is based on REpresentational State Transfer (REST), but is bound to UDP instead of TCP. It has two main sublayers: the *message sublayer*, which handles reliable communication in UDP, and the *request-response sublayer*, which handles REST communication (AL-FUQAHA *et al.*, 2015). To provide the aforementioned functionality, CoAP has four different messaging types: reliable (needs an acknowledgment and are resent after an exponential back-off timeout), non-reliable (does not need an acknowledgment), piggyback (the response goes together with the acknowledgment), and separate (an acknowledgment is sent to prevent retransmission, but the response is sent separately from the acknowledgment). The CoAP website (`https://coap.technology`) provides an overview, specifications, and implementations for those intended on using the protocol.

2.2.4.1.2 Message Queuing Telemetry Transport (MQTT) Protocol

The MQTT is a simple, lightweight binary protocol built over TCP for constrained devices and networks defined by (MQTT 3.1.1, 2014). MQTT implements the publish/subscribe pattern with three components: (1) Publishers, (2) Subscribers and the (3) Broker. Publishers provide information on specific topics of interest to which subscribers can register themselves. The broker then acts as a middleman receiving the updates from the publishers and forwarding them to registered subscribers through three levels of QoS (AL-FUQAHA *et al.*, 2015; LIN *et al.*, 2017; NAIK, 2017). MQTT-SN (MQTT-SN 1.2, 2014) is an MQTT extension aimed at non-TCP/IP networks envisioned for constrained applications such as Wireless Sensor Networks (WSNs). It introduces two new components: a Gateway located between the nodes and the broker that acts as a protocol converter from MQTT to MQTT-SN, and a Forwarder located between the Gateway and the clients (publishers/subscribers) that simply forwards the messages unchanged. The MQTT website `http://mqtt.org` provides MQTT documentation and software implementations for users.

2.2.4.1.3 Extensible Messaging and Presence Protocol (XMPP)

the XMPP is defined by (IETF RFC 6120, 2011), enables the near-real-time exchange of XML-based messages between network entities. It is usually implemented in a distributed client-server architecture over TCP, is platform-independent and provides security mechanisms such as authentication, access control, and encryption. However, it is not widely implemented due to lack of QoS and high communication overhead (AL-FUQAHA *et al.*, 2015).

2.2.4.1.4 Advanced Message Queuing Protocol (AMQP)

The AMQP is defined by (AMQP 1.0, 2012), runs over TCP and supports reliable communication for message-oriented environments, implementing different ways of message distribution such as store and forward, publish and subscribe, message distribution, message queuing, context-based routing, and point-to-point routing (LIN *et al.*, 2017). The communication is handled by two types of components: exchanges used to route messages and queues, which receive routes messages and send them to receivers (AL-FUQAHA *et al.*, 2015).

2.2.4.1.5 Data Distribution Service (DDS) Protocol

DDS (DDS 1.4, 2015) is a publish-subscribe broke-less protocol for real-time M2M communications suitable to constrained devices. DDS has data writers (which interacts with the Publishers regarding the shared data) and data readers (which read the published data and deliver it to Subscribers) — the topics are the types of data being published (SALMAN; JAIN, 2016). The protocol offers 23 QoS policies and achieves high reliability and scalability (AL-FUQAHA *et al.*, 2015; LIN *et al.*, 2017).

2.2.4.2 Service Discovery Protocols

Service discovery protocols allow devices to be aware of available services and functionality in the network without human intervention. To be used in the IoT environment, the protocols must be able to work with legacy sensor networks and provide secure access to resources energy-efficiently. Besides specific protocols for service discovery such as Multicast DNS and DNS Service Discovery, there are other service discovery approaches based on Application protocols such as MQTT-based and CoAP-based (ACHIR; ABDELLI; MOKDAD, 2020). Papers (VILLAVERDE *et al.*, 2014) and (CABRERA; PALADE; CLARKE, 2017) evaluate different service discovery protocols for the IoT.

2.2.4.2.1 Multicast DNS (mDNS) Protocol

The Multicast DNS (mDNS) Protocol (IETF RFC 6762, 2013) is a name-discovery protocol that requires no configuration and can be run without infrastructure. Node names are inquired through IP multicast — when a node receives its name, it multicasts a response message with its IP address, allowing every node in the local network to link name with IP addresses (AL-FUQAHA *et al.*, 2015).

2.2.4.2.2 DNS Service Discovery (DNS-SD) Protocol

The DNS Service Discovery (DNS-SD) Protocol (IETF RFC 6763, 2013) allows clients to discover a set of desired services using DNS messages multicasted to specific addresses through UDP. It can be used together with mDNS, and there are efforts in developing lightweight versions for the IoT ecosystem (AL-FUQAHA *et al.*, 2015).

2.2.4.3 Network Protocols

Network protocols are responsible for transmitting the data over the data link layer by providing connections between two different entities and routing the information from source to destination. Some IoT intrinsic characteristics that influence how those protocols should operate are: the connection between devices can be intermittent, the network topology may change frequently, the devices are likely to be heavily constrained regarding power and computation, and at times the devices operate in a harsh environment (DHUMANE; PRASAD; PRASAD, 2016). Therefore, network protocols need to be adaptable to different possible situations of disconnection and re-connection, power-efficient so resources are saved and autonomous so they require minimum human intervention.

2.2.4.3.1 IPv6 over Low power Wireless Personal Area Networks (6LoWPAN)

6LowPan (MONTENEGRO *et al.*, 2007) is one of the most remembered network encapsulation protocol for IoT. It acts as an intermediary layer between IPv6 and 15.4 MAC and PHY, making the IPv6 suitable for low-power wireless networks. Low resource consumption, good connectivity, and ad-hoc self-organization make 6LoWPAN a suitable protocol for IoT applications (AL-FUQAHA *et al.*, 2015; LIN *et al.*, 2017). The standard is not dependent on the PHY layer

— it can be used over other PHY protocols such as Ethernet and WiFi (TRIANTAFYLLOU; SARIGIANNIDIS; LAGKAS, 2018).

2.2.4.3.2 Routing Protocol for Low Power and Lossy Networks (RPL)

The RPL (IETF RFC 6550, 2012) was designed to take into account constrained devices and provides bidirectional connectivity, reliability, robustness, flexibility, and scalability. The protocol has a multi-hop hierarchical topology in which the leaf traffic is redirected to a root node by a single path (TRIANTAFYLLOU; SARIGIANNIDIS; LAGKAS, 2018; KHARRUFA; AL-KASHOASH; KEMP, 2019). RPL has gained much attention in recent years. Some characteristics, challenges, applications in IoT, and proposed enhancements can be found in (KHARRUFA; AL-KASHOASH; KEMP, 2019).

2.2.4.4 Connectivity Protocols

Connectivity protocols are the interface between the IoT components and the real world. There are many connectivity protocols for the IoT and enough diverse applications so a protocol is more suited to one application than others. They are usually classified according to their range: short-range or long-range protocols. Regardless of their classification, the protocols focus on low power consumption given the characteristic resource constraint in IoT devices. The Low Power Wide Area (LWPA) technologies were developed to attend applications that required longer-range keeping the costs low — this is obtained with lower data rates and higher latency (RAZA; KULKARNI; SOORIYABANDARA, 2017). Papers (RAZA; KULKARNI; SOORIYABANDARA, 2017) and (DING *et al.*, 2020) bring a general view of several IoT connectivity protocols.

2.2.4.4.1 Near-field communication (NFC)

NFC is a short-range half-duplex communication protocol deployed over an inductive coupling between devices, wich is mostly used in industrial applications, mobile phones, and payment systems. The work in (COSKUN; OZDENIZCI; OK, 2013) presents a survey on NFC technology, discussing its working principle, ecosystem, applications, security issues, and open research areas.

2.2.4.4.2 Bluetooth

Bluetooth is a communication protocol that uses data packets and a master-slave network structure. Version 4.0 of Bluetooth has introduced the *Bluetooth Low Energy (BLE)*, a protocol with lower-power consumption, lower setup time, and capable of operating in a star network topology with an unlimited number of nodes, making BLE one of the key enabling technologies for short-range IoT applications. New versions 5.0 and later (BLUETOOTH SIG, 2019) of

Bluetooth brought improvements in speed, range, security, energy efficiency, and interoperability (ČOLAKOVIĆ; HADŽIALIĆ, 2018). The work in (YIN *et al.*, 2019) discusses the Bluetooth 5.0 new features and the use of Mesh topology for it, as well as presenting the impact of these improvements in IoT applications and future research directions. Authors from (DARROUDI; GOMEZ, 2017) survey the state of the art in BLE mesh networking specifically, reviewing different propositions.

2.2.4.4.3 Zigbee

Zigbee is a low-power, low-cost technology based on the IEEE 802.15.4 standard (IEEE STD 802.15.4-2015, 2016) that is being used in IoT for home automation, industrial monitoring, and health and aging population care applications. It provides multi-hop routing, maintenance of routes, identification of add/remove of IoT devices (BANSAL; KUMAR, 2020), the lack of QoS is a challenge to be overcome (ČOLAKOVIĆ; HADŽIALIĆ, 2018).

2.2.4.4.4 LoRaWAN

LoRaWAN is an open-source technology designed and maintained by the LoRa Alliance (LoRa Alliance, 2022). It provides end-to-end protection, mobility, localization and two-way communication. The technology architecture is mainly composed by (a) *End Devices* that communicate with the (b) *Gateways*, which in turn act as a mediator between the devices and the Internet — device data is then aggregated and processed by (c) *Network Servers*. LoRaWAN allows sensing nodes to transmit data through large distances in a single hop, resulting in lower energy consumption (ALMUHAYA *et al.*, 2022).

2.2.4.4.5 NB-IoT

Narrow-Band IoT (NB-IoT) is a narrowband LPWAN technology based on cellular infrastructure. Its objective is to support a wider coverage with higher connection density with lower device cost and longer battery life thanks to available power-efficiency mechanisms. Its protocol is based on LTE, making the NB-IoT technology able to coexist with existing LTE networks and therefore reducing its deployment cost (DING *et al.*, 2020).

2.2.5 IoT Capability Boost

One of the main characteristics of IoT devices is their resource constraint, i.e., most devices are not able to perform high-complex tasks which require a lot of computational power. To address these challenges, other computing paradigms such as Cloud Computing (CC), Mobile Computing (MC), Edge Computing (EC), Fog Computing (FG), and Cloudlets have been considered to work with IoT and provide greater processing and storage capabilities, as well as easy access to data from other sensors. In this section, we review three of them: Cloud Computing, Edge Computing, and Fog Computing.

2.2.5.1 Cloud Computing

Cloud Computing is a computing paradigm envisioned for enabling ubiquitous and on-demand network access to a shared pool of configurable physical or virtual resources with quick provisioning and minimal management effort (ISO/IEC 17788:2014, 2014). It can be viewed as a distributed system whose resources are aggregated and reassigned among its clients, who can acquire and release computing capabilities according to the demand without human interaction and be billed only for which resources have been used (MELL; GRANCE, 2011).

Cloud Computing can provide fault-tolerance and scalability at low cost, being able not only to aggregate, analyze, distribute and extract meaningful information from collected IoT data but also to remotely manage and control systems. Besides, CC is globally accessible and available, allowing IoT devices to be located and accessed anywhere, anytime. Moreover, CC can concentrate services from different providers and bring them to users in an easy, intuitive way, acting as an intermediate level between IoT devices and the application customer and hiding IoT heterogeneity and the complexities of the system (BOTTA *et al.*, 2016; CAVALCANTE *et al.*, 2016; SINGH *et al.*, 2016).

There are also disadvantages to using CC with the IoT. First, the devices need to be connected to the Cloud infrastructure to be available, which increases the energy consumption on already resource-constrained devices — there is no guarantee of a stable network performance since broadband Internet access has not evolved as storage and computation technologies (CAVALCANTE *et al.*, 2016; TSCHOFENIG, 2016). Also, contextual information in IoT can be as important as the data itself and need to be provided to the Cloud, which in turn needs to be able to store and process these data properly (CAVALCANTE *et al.*, 2016). Moreover, the centralized nature of the Cloud and the time required to access it may not be practical for mission-critical scenarios of high data transmission and processing or low-latency requirements (YOUSEFPOUR *et al.*, 2019). To provide IoT devices with high-bandwidth and low latency computing capabilities, researchers and industries proposed to bring those capabilities closer to the edge of the network, in paradigms such as Fog Computing and Edge Computing.

2.2.5.2 Fog Computing

According to the IEEE Standard 1934 (IEEE STD 1934-2018, 2018), Fog Computing is "*A horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum*". It is considered a Cloud Computing extension that brings computational resources (computing, storage, communication) closer to the network edge. Fog Computing is associated with moderate computational resources than Cloud Computing, and lower power consumption (PULIAFITO *et al.*, 2019). Despite being closer to the devices being data, Fog nodes can be located in anywhere in the cloud-to-thing continuum, i.e., Fog Computing is not restricted to a specific layer of a network's topology (PULIAFITO *et al.*, 2019).

The use of Fog computing is very beneficial to IoT systems. Because it is closer to smart objects, Fog can provide better latency performance for real-time systems. The proximity also decreases bandwidth consumption (since the amount of data sent to a Cloud datacenter is reduced) and improves the system's context awareness. Also, sensitive data can be stored and processed locally, improving data security (DUSTDAR; AVASALCAI; MURTURI, 2019; PULIAFITO *et al.*, 2019). Fog Computing also helps on the system scalability and distribution — as the number of devices increase, it is possible to create more Fog nodes at a lower cost when comparing to a new Cloud data-center. At the same time, a single Fog node can interact with more than one Cloud provider, making it easier to provide one service in multiple platforms (BOTTA *et al.*, 2016; CAVALCANTE *et al.*, 2016; UHER *et al.*, 2016).

It is likely that with a decentralized structure, the Fog nodes interact directly with one another, making service and resource orchestrate a big challenge. Also, Fog benefits may be limited with mobile IoT devices, since the topological distance between the device and its associated Fog Node may increase and with that impact latency. Finally, the distributed nature of Fog and its resource constrain makes security more challenging, since it is likely those nodes less robustness in security mechanisms (SINGH *et al.*, 2016; LIN *et al.*, 2017; PULIAFITO *et al.*, 2019) — the work in (PATWARY *et al.*, 2021) discuss security challenges of Fog Computing in general, while the work in (NI *et al.*, 2018) discuss security challenges of Fog Computing in the IoT context.

2.2.5.3 Edge Computing

Edge computing is another paradigm that brings computational resources from the Cloud to the edge of the Network. Even though Fog Computing and Edge computing are sometimes seen as the same paradigm and have its terms used interchangeably, some authors and even the Fog Standard see key differences between them: the Fog is more disperse in the network topology and are more resourceful, while the Edge is more resource constrained and is located in the local network of IoT devices, i.e., Edge nodes would be the immediate first hop from IoT devices (IEEE STD 1934-2018, 2018; YOUSEFPOUR *et al.*, 2019).

Edge Computing presents the same advantages as Fog due to its proximity to the device: better latency, reduced bandwidth usage and power consumption and increased reliability, security, and privacy protection (LAROUI *et al.*, 2021). Likewise, resource estimation, allocation and optimization, less security robustness due to resource constrains and network management able to support mobile IoT devices are challenges for Edge Computing (DUSTDAR; AVASALCAI; MURTURI, 2019).

2.2.5.4 Edge vs. Fog vs. Cloud

The work in (VARSHNEY; SIMMHAN, 2017) brings a small comparison between all three paradigms, which most likely will work together in a IoT-Edge-Fog-Cloud continuum to support different types of IoT service requirements. In general, as the network become more centralized (i.e., from Edge to Fog to Cloud) aspects such as computational resources, bandwidth, reliability and latency increases while mobility, the number of devices, energy-awareness and cost decreases, as shown by Figure 2.

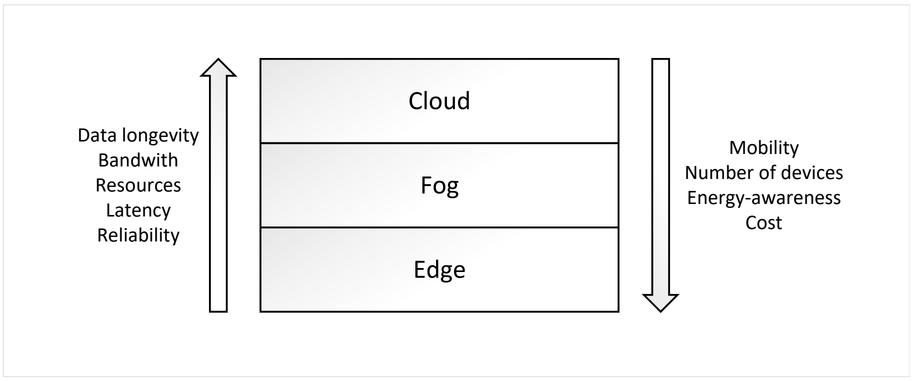


Figure 2 – Comparison between Edge, Fog and Cloud Computing paradigms from (VARSHNEY; SIMMHAN, 2017).

In another words, the Cloud is most likely to be used in more complex and not critical situations; the Fog can be used in more latency-sensitive tasks that still requires some computational power, or in tasks that need better physical presence; finally, the Edge can be used in very critical and lightweight tasks that need high physical availability.

2.2.6 IoT Security

Assuring Security, Privacy and Trust is a crucial requirement for widespread adoption of the IoT, specially for critical applications. Security solutions for IoT systems must permeate all of its components (network & communication, devices, services, platforms, etc.), and be addressed in all stages of development (EOM, 2015; LI; TRYFONAS; LI, 2016).

IoT intrinsic characteristics directly impact security. Firstly, the high heterogeneity of IoT devices impede a comprehensive security solution. Also, device resource constraints hinder the adoption of off-the-shelf security mechanisms, demanding lightweight security algorithms which are able to make a compromise between resource consumption and security. Finally, the scale and complexity of the IoT system increases the attack surface and the lack of software security updates and/or device protection such as anti-virus magnifies the threats of conventional Internet (SICARI *et al.*, 2015; LI; TRYFONAS; LI, 2016; MAKHDOOM *et al.*, 2019).

The following security requirements apply to an IoT security solution:

- AUTHENTICATION: In IoT, each thing needs to be properly authenticated. Given the large number of diverse objects and the fact that many will operate with little or no human intervention, a proper identification and authentication mechanism is needed (LIN *et al.*, 2017).
- AVAILABILITY: In IoT, there are both hardware and software availability. *Software availability* refers to providing services to users simultaneously, while *hardware availability* refers to the presence of devices at all times to provide the services. In cases of critical applications or with real-time requirements such as emergency response, availability is a crucial requirement of the system (AL-FUQAHA *et al.*, 2015; LIN *et al.*, 2017).
- INTEGRITY: It is necessary to ensure that received data was not forged, tempered with and properly stored so that the system do not take wrong decisions and actions which could cause damage to the system or even to users (LIN *et al.*, 2017).
- CONFIDENTIALITY: IoT has to ensure that collected data will not be accessed by unauthorized parties. Due to the enormous amount of data generated by IoT and the resource limitation of many of its components, customary security solutions may not be applied. Confidentiality must be simple so not to present high resource consumption and flexible enough to be modified at run-time (LIN *et al.*, 2017)
- PRIVACY: Sensed data can contain sensitive information regarding their owner or users. Privacy protection mechanisms are necessary to regulate which data referring to each user can be accessed, and by whom. This is considered a crucial requirement for technology acceptance by the public (LIN *et al.*, 2017). Moreover, privacy protection should not interfere with data source authentication (ITU-T Y.2060, 2012).
- BUILT-IN DEVICE SECURITY: Many IoT end-nodes will operate with little or no human intervention. Moreover, the sheer number of them will create a vast attack surface if built-in security mechanisms, such as automatic firmware upgrades, are not deployed. Therefore, it is highly desirable that devices are be able to detect, diagnose, isolate and even counterattack security threats (SICARI *et al.*, 2015; UHER *et al.*, 2016).
- TRUST: This is a fundamental security requirement in IoT environment due to its heterogeneity and privacy requirements. Trust need to be negotiated and achieved between each IoT layer, between devices, between devices and applications and between users and applications. When considering trust negotiation in IoT systems, the issue of accepted certification authorities and their requirements need to be addressed, as well as the definition of a trust negotiation language (SICARI *et al.*, 2015; LIN *et al.*, 2017).

Since IoT is a convergence of many types of technologies, it inherits security vulnerabilities from all of them. *Resource-constrained devices* are unable to deploy robust security mechanisms and can compromise availability if it goes out of service (due to malfunction, spent battery or communication instability) (WHITMORE; AGARWAL; Da Xu, 2015; LI; TRYFONAS; LI, 2016). *Insecure interfaces and protocols* between the many IoT components (e.g., devices and middlewares, middlewares and Fog/Cloud Computing, Fog/Cloud computing and users) may compromise confidentiality, integrity, availability and accountability due to insufficient validation of data or authorization checks and weak credentials (HASHIZUME *et al.*, 2013; EOM, 2015). The *wireless medium* used in many IoT applications is more susceptible to eavesdropping and replayed or faked messages (WHITMORE; AGARWAL; Da Xu, 2015). The *components* themselves are also a vulnerability — from small IoT devices to Cloud serves, any node may be compromised, tampered with or even hijacked, compromising the system (LI; TRYFONAS; LI, 2016). Also, compromised IoT nodes or the Cloud may be used to deploy attacks to other systems (STELLIOS *et al.*, 2018), such as DDoS or brute-force attacks and malware distribution (Cloud Security Alliance, 2016).

Next, some threats to the IoT environment are discussed. The text focus on broader threats that goes through all layers, based on (Beecham Research Limited, 2015) and (Cloud Security Alliance, 2016). There are, however, many surveys on IoT security and specifically on threats and attacks. Authors from (BUTUN; ÖSTERBERG; SONG, 2020) review and classify various WSNs/IoT attacks and review possible defense mechanisms against them; the work in (SAMAILA *et al.*, 2018) reviews security requirements, threat models and protocols in various IoT application domains; IoT vulnerabilities are surveyed in (NESHENKO *et al.*, 2019).

2.2.6.1 Illegal/non-authorized access

All entities in the IoT-Cloud-User continuum need to be able to confirm each other's identity so it is possible to establish a trusted communication. Also, there can be many sensitive data in the system that most likely needs to be better protected than others, such as private information in a IoT-based healthcare system. This threat can result in unauthorized disclosure of information, misuse due to incorrect privilege elevation, falsification and/or corruption of data. COUNTERMEASURES: proper user authentication is essential to ensure the identity of an entity and access control strategies are crucial to ensure reliable user authorization. Since IoT devices are not able to deploy standard security measures, lightweight authentication algorithms such as reviewed by (EL-HAJJ *et al.*, 2019) and (FERRAG *et al.*, 2017) are necessary. Access control policies (role-based, geo-based, device-based, etc) ensures that only authorized users/devices have access to sensitive data. In this context, multi-factor authentication for users and least privilege policies are highly recommended.

2.2.6.2 Data breaches/disclosure

Data breaches can happen due to many different reasons, such as eavesdropping an insecure transmission medium, weak security storage (i.e., storing data in plaintext or have a server vulnerable to attacks such as SQL injection) or wrong system configuration that allows an unauthorized party to access information that can be use to violate user privacy or even violate player's intellectual property to competitors, resulting in great legal and commercial impacts. COUNTERMEASURES: data security is achieved with both confidentiality and integrity. In this context, encryption techniques can protect the data, but at the same time impacts system performance. Even though Some IoT protocols have encryption methods, they establish the security at the lowest level supported by both sides in the communication, i.e., a device with no encryption capacity may gain access to the system without any protection (UHER *et al.*, 2016). Therefore, lightweight cryptography algorithms such as those reviewed by (SINGH *et al.*, 2017) and (KHAN; RAO; CAMTEPE, 2021)) are necessary. Not only that, key management strategies also need to be tailored for the IoT — the work in (NAOUI; ELHDHILI; SAIDANE, 2016) review some IoT key management protocols regarding their security capability.

2.2.6.3 Device/application/account hijacking

this is a threat which can have serious consequences depending on the security credentials of the hijacked object. If a highly privileged asset is hijacked or if the system do not employ necessary access control of its assets, the entire system — data, services, business logic — is at risk. However, many IoT devices are supposed to work without any human intervention, making them more susceptible to hijack or to be tampered with. Also, Cloud users may neglect security measures and have their account compromised.
COUNTERMEASURES: identity management control can mitigate account hijacking, and more secure accounts (i.e., accounts with two-factor authentication, few root accounts) are also highly advisable to protect the system. Strict identity and access controls are necessary to mitigate the consequences when a device is compromised.

2.2.6.4 Network intrusion

this is a threat present in any communicating system, and is the same principle of the regular Internet: a malicious actor is able to take control of a trusted entity of the system or circumvent the security measures in place to gain access to a system. The IoT has a huge attack surface, so network intrusion can happen in many different ways. Besides illegal access and hijacks already discussed, devices can be compromised by software injection or physical tampering. Also, any entity that runs a software can be compromised through malwares (the work in (VIGNAU *et al.*, 2021) reviews successful IoT malwares from the last decade) or have ‘original’ vulnerabilities in their software. Finally, communication protocols such as those discussed in Section 2.2.4 may have gaps in their security measures that allows a malicious entity to penetrate the network.
COUNTERMEASURES: in order to protect IoT entities, it is important to (1) have malware protection; (2) be able to provide security updates whenever it becomes vulnerable to a new exploit; (3) force the user to change default settings such as root/admin password; (4) create a chain of trust in the various software levels to provide secure booting; (5) implement security by design for both integrated circuits and software so they are robust against physical tampering and common software vulnerabilities that can be easily sanitized and (6) use secure protocols for communication that, together with other security mechanisms such as authentication and

encryption, hinders the action of malicious actors. Systems as a hole may also deploy Intrusion Detect Systems (IDSs) (some IDSs for IoT model are surveyed in (ELRAWY; AWAD; HAMED, 2018)) to impede and/or mitigate the impact of a security breach.

2.2.7 Final Remarks

The Internet of Things is a technology with potential to transform the way we live as a society, allowing real-time, data-based decisions to be made an enabling solutions in healthcare, environmental preserving and other important areas of our lives. New technologies and solutions are still being brought to the table, such as Mist Computing for computation (YOUSEFPOUR *et al.*, 2019), Blockchain and Artificial Intelligence to help security (TAHSIEN; KARIMIPOUR; SPACHOS, 2020; LONE; NAAZ, 2021) and 5G and others to communication (CHETTRI; BERA, 2020).

The future of IoT is still challenging and full of opportunities. One of them, which is the focus of this work, is the possibility to connect Unmanned Aerial Systems into the IoT, making it possible for them to provide and consume services to and from other IoT devices. This specific scenario of IoT-connected UAVs is discussed in the next Chapter.

CHAPTER 3 IOT-ENABLED UAVS

Many opportunities arise when Unmanned Aerial Vehicles (UAVs) are connected to other devices or systems like the Internet of Things (IoT). It is possible for UAVs to perform more complex and collaborate in new applications, but at the same time there are several challenges in different domains. In this Chapter, a brief discussion on how IoT-enabled UAVs can be used, and different challenges of this integration are discussed.

There are two main benefits for UAVs when they integrate the IoT ecosystem: *virtualization* and *computation offloading*. Virtualization makes UAVs available to clients through user-friendly, abstract interfaces, allowing its resources to be pooled among multiple users and applications depending on their availability. This is being referred to as Drone-as-a-Service (DaaS) or UAV-as-a-Service (UAVaaS) (KOUBAA *et al.*, 2017). As a result, clients do not need to own a UAV to make use of its services, which lowers the overall cost and facilitates the implementation of service systems such as these (MAHMOUD; MOHAMED; AL-JAROODI, 2015). Computation offloading allows the Cloud/Fog/Edge to be used to process and compute most of UAV acquired data, reducing UAV power consumption and improving its performance. Reliability is increased because of data back-up, and users are able to monitor the UAV throughout the mission (MAHMOUD; MOHAMED; AL-JAROODI, 2015; MOTLAGH; TALEB; AROUK, 2016).

Besides those main impacts regarding service exchange, there are other interesting situations in which UAVs can either benefit from IoT nodes to share data with other vehicles with no line-of-sight (LoS) using those nodes as relay or in the opposite situation, functioning as relay nodes in a region where the connection infrastructure is not present (remote areas) or damaged (disaster scenarios).

Authors from (PAKROOH; BOHLOOLI, 2021) surveys different types of services UAVs can assist and classify them into four types: data-related services (data collection, dissemination), battery-charging services, and communication services (in remote areas, disaster scenarios and others) and Mobile Edge Computing (MEC) services. Works from (MOHAMED *et al.*, 2017) and (ABRAR *et al.*, 2021) fit into the last category, with UAVs taking roles as Fog and Edge nodes to IoT sensors, respectively. The work on (ALWATEER; LOKE; RAHAYU, 2018) presents a simulation study on UAV service provision, investigating how the number of UAVs, their relative distance from control stations and the frequency of client request impact on service attendance and delivery.

A UAV hub with different vehicles able to provide services and managed by a central Cloud-base platform is a frequent example of IoT-connected UAVs. Among others, the Dronemap presented in (KOUBAA *et al.*, 2017) is a cloud-based UAV management system that provides real-time UAV control and monitoring and dynamic mission scheduling by users; the work in (YAPP; SEKER; BABICEANU, 2016) proposes a framework to enable a UAV-as-a-Service application in which UAVs in a hub are allocated by a Cloud coordinator to provide services to its clients; in (MAHMOUD; MOHAMED; AL-JAROODI, 2015), RESTful communication architecture (with UAVs as web servers) to provide services to clients and other UAVs; finally, authors from (ERMACORA *et al.*, 2014) propose a service platform for surveillance managed by the Cloud and requested through mobile app.

3.1 IoT-enabled UAV Challenges

As seen before, the integration of UAVs into the IoT brings challenges that need to be overcome — some of them well known by the unmanned vehicle community, others emerging due to the connection of vehicles to an infrastructure such as the IoT. This work classifies these challenges in three categories: public safety and privacy, standardization and technical.

3.1.1 Public Safety and Privacy Challenges

Confidentiality issues regarding the data acquired by UAVs is a major concern — particularly if there is critical information being collected — and will play a starring role in public acceptance of the technology. There is a tendency to store as much data as possible into the UAV main memory in order to ensure availability (HARTMANN; STEUP, 2013), which ends up being a critical security weakness (JING *et al.*, 2014). If the UAV is stolen or has its control taken, it can be used as a gateway to probe sensitive information from the secured network it is authenticated in. IoT could improve security by transferring sensitive data to the cloud. However, providing data distribution in a secure manner on UAVs or other resource-restricted devices is another challenge.

Another concern for the general population would be snooping — a UAV can take unauthorized videos or photos and share them online, making them nearly impossible to be removed and very difficult to identify the perpetrator. The UAV owner privacy can also be compromised if a stolen UAV contains private data and media. It is likely that governments will enact legislation to UAV registration and to prevent privacy invasion, as done in some states in the US (KIM; JO; SHRESTHA, 2014).

3.1.2 Regulation & Standardization Challenges

Certification is a must for insertion of UAVs into the airspace and prevent the technology of being used for nefarious purposes, such as physical assault, drug smuggling and others (MOTLAGH; TALEB; AROUK, 2016). UAV safety plays a major role in this regulations, since a UAV out of control can cause damage to property or harm people. Because of that, many regulation agencies such as the FAA and the EASA are regulating the UAV market.

For IoT, standardization is also a major challenge, and must involve both the industry and governments. From a technical point of view, standardization is necessary to ensure all devices are able to communicate, preventing a “Babel Tower Effect” in which devices become split into disjoint subsets (for instance, all devices from the same manufacturer) that can only talk to others from the same subset. From a social and economic point of view, standardization will favor the entrance of small and medium companies in the market, stimulating entrepreneurship and competition, benefiting the final customer and spreading the use of the technology.

3.1.3 Technical challenges

There are many technical issues in UAVs being integrated into IoT, with some important ones presented in this Section.

- *Energy efficiency:* UAVs are resource-constrained devices. Hardware and software design must take into account limited memory, storage and processing capabilities, as well as a limited power source. Thus, algorithms and communication protocols must be as energy-efficient as possible.
- *Heterogeneity:* the IoT will be formed with a variety of devices developed for various platforms and using different communication protocols, which can lead to compatibility and interoperability issues. This heterogeneity will also extend to the acquired data, that will most likely be non-structured. How data is gathered, distributed, stored and recovered has to be planned carefully in order to ensure real-time and security requirements.
- *GPS Security:* GPS spoofing attacks can cause the aircraft to completely lose control, which is a very critical issue, or help attackers to hijack UAVs.
- *Safety:* safety is an important aspect for the devices, which must be able to determine the “health” and the authenticity of both its internal and external components (PIGATTO, 2017) — something even more challenging if the internal components are also connected wirelessly.
- *Security:* a challenge in any communicating system, Security relates to all other aspects aforementioned. Limited resources demand efficient security algorithms that do not com-

promise performance or resource/power consumption; heterogeneity defies the idea of implementing a global security policy for all devices and data storage must also be covered by security in case the communication is interrupted or the physical integrity of the device compromised. Therefore, security solutions must permeate all layers of the architecture — reducing the breaches throughout the layers will consequently reduce the overall chances of attacks to the network.

In the remaining part of this Chapter, a more detailed discussion on different challenges is presented.

3.2 Energy Efficiency

For battery-powered wireless embedded systems like UAVs, energy efficiency is a key challenge, with considerable research effort has been devoted to energy optimization of such systems (RAGHUNATHAN *et al.*, 2005; Sravanthi Chalasani; CONRAD, 2008; HUANG *et al.*, 2017). Reducing energy consumption in UAVs leads to (1) increased autonomy and longer missions, encompassing bigger tasks, which results in (2) reduced number of flights to accomplish a task; (3) reduced battery disposal due to battery lifetime increase and (4) heavier payloads due to energy consumption reduction(ZENG; ZHANG; LIM, 2016).

In UAV domain, most published papers regarding energy efficiency focus on finding alternative energy sources (e.g. solar panels, piezoelectric-based approaches). Another solution would be the optimization of algorithms/communication/protocols to reduce the energy consumption and increase battery times (RANGANATHAN, 2010).

Several research reports focus on providing additional energy supply to unmanned vehicles. Energy scavenging concepts were reviewed and analyzed by (THOMAS; QIDWAI; KELLOGG, 2006) to determine their potential for supplementing the on-board energy of small electric unmanned systems. Experiments were performed with photonic (solar), kinetic-flow (wind), thermal, electromagnetic sources of energy, and autophagous structure-power concepts that allow for energy generation through self-consumption of system structure. Similarly, (MAGOTEAUX; SANDERS; SODANO, 2008) analyzed solar and piezoelectric energy harvesting techniques along with their basic functions, (ERTURK; RENNO; INMAN, 2009) analyzed piezoelectric energy harvesting with an application to UAVs, and (ANTON; INMAN, 2008) analyzed vibration and solar energy in a mini UAV. Moreover, (THOUNTHONG; RAËL; DAVAT, 2009) proposed an energy management of fuel cell/battery/supercapacitor hybrid power source for vehicle applications, which goes on the same direction of aforementioned works. Finally, another approach has considered acquiring energy from wind on UAVs (LAWRANCE; SUKKARIEH, 2009). Conclusions provided by these investigations were promising, however focused solely on acquisition of more energy supply during vehicle's operation.

Although efficient, the acquisition of power during operation may increase the weight of an unmanned vehicle and reduce its endurance time. Thus, some approaches focused on dealing with communication protocols towards the reduction of energy consumption. As pointed out by (BOLLA *et al.*, 2011), for disruptively boosting network energy efficiency, hardware enhancements must be integrated with ad-hoc mechanisms that explicitly manage energy saving, by exploiting network-specific features. Concerning routing protocols that can take into account energy constraints in embedded systems, (CHABAREK *et al.*, 2008) described the power and associated heat management challenges in routers, considering making power-awareness a primary objective in the design and configuration of networks, and in the design and implementation of network protocols.

Reduction of unnecessary energy consumption is a major concern in wired networking, due to the potential economical benefits and its expected environmental impact, as discussed by (BIANZINO *et al.*, 2012). Other works have focused on wired communications, such as (SHAABANA *et al.*, 2014) who provided energy minimization techniques for high-speed communication over optical networks. However, wired approaches when designed for UAVs may increase weight and become limiting too, leading to a natural interest in wireless communications.

Therefore, there is a growing tendency on replacing wired cables on internal communications in unmanned vehicles, specially UAVs (DANG; MIFDAOUI; GAYRAUD, 2012; Elgezabal Gomez, 2010) and few researches have been focusing on energy-aware wireless communications challenges as summarized by (ZENG; ZHANG; LIM, 2016) and also generally addressed in (KIM; CHANG, 2017; GHAZZAI *et al.*, 2017).

3.3 Service Exchange

As previously discussed, UAVs become a *thing* when they are integrated in the IoT infrastructure and all of its resources can be available to clients/users through an Internet connection. Service availability will depend on contextual information such as location, battery levels, available payload, and others (MAHMOUD; MOHAMED; AL-JAROODI, 2015).

UAV services should enable users to monitor the UAV state and access and monitor its resources. Since service clients can be either human or other machines, it is crucial to correctly identify all resources and services available (MAHMOUD; MOHAMED; AL-JAROODI, 2015). Even though some works such as (ROYO *et al.*, 2008; MAHMOUD; MOHAMED, 2015) and (ALWATEER; LOKE; RAHAYU, 2018) bring some service classification, it would be highly desirable that all services were properly described by a standardized set of definitions.

Also, UAVs need a way to allow service operations such as advertisement and autonomous discovery, request and provision so the system is self-managed. It is also highly desirable that security mechanisms are employed by the service provision platform. Service provision by autonomous agents such as robots and UAVs is not a novel study field, and more recently have been taking the IoT environment into account.

Authors from (ROYO *et al.*, 2008) present a system for UAV mission flexibility. Even though this work was published before the term "Internet of Things" was coined, the concept of services were used to implement all required mission functionality, classified as Flight, Mission, Payload or Awareness services. Each service can be deployed as Variable (short information with no deliver guarantee), Events (notifications with guaranteed delivery), Remote Invocations (task requests from one node to another) and File Transmission (for continuous data transmission such as camera streams). Even though there is no support for providing those services to external clients, the paper as whole makes very important considerations regarding UAV service provision.

Robotic services are also a topic of interest. The DAvinCi Cloud Computing framework presented in (ARUMUGAM *et al.*, 2010) fuses the sensor and mapping information of many robots that operate in the same large environment and create a map, which is then provided on demand for all robots on the system, discarding the necessity of each robot to perform the environment mapping. In (FURRER *et al.*, 2012) the authors present an open-source platform for robot services, in which unitary services can be combined to realize complex tasks in a Ambient-Assisted Living (AAL) case study. (DORIYA; CHAKRABORTY; NANDI, 2012) proposes a Cloud-based approach for robotic services, in which users request services such as navigation, map building and object recognition through a Web interface. A Cloud manager distribute those tasks to robot handlers, and manages user authorization for service requests. A prototype of the system was validated with robots inserted in a simulated environment. (VANELLI *et al.*, 2017) presents an architecture for robots as a service (RaaS) in the cloud, implemented over the MQTT (Message Queuing Telemetry Transport) protocol. The architecture is implemented in three layers. The Local layer is composed by robots and a bridging device connected through low-performance protocol; the gateway acts as a middleware between the Local layer and the Cloud layer, in which the client can connect themselves and access the robot data and remotely control it.

Back to UAVs, a Service-Oriented Middleware (SOM) in (MOHAMED; AL-JAROODI, 2013) employs virtualization so that all UAV hardware components are viewed as services. The middleware handle tasks such as service advertisement, discovery, invocation and exchange. Each service is described using Web Service Description Language (WSDL), and composite services provided by a set of UAVs can be configured. Each UAV keeps local and remote service information and other UAS components data in an internal broker. The subsequent work in (MAHMOUD; MOHAMED; AL-JAROODI, 2015) presents a UAV service provision to the Cloud through a REST communication architecture in which the UAV is a web server that answers to request from clients. The REST architecture is said to be better suited to the real world, since it is easy to use and to scale.

A REST architecture is also used in (COSTA; GONCALVES, 2016) to provide services for untrained personnel to operate and control robots through an Internet connection. The architecture is built in three layers: the *application layer* provides the interface for users; the *server layer* is composed by two units, one for monitoring and another for controlling the robots, which can be implemented in the same or in separate machines; finally, the *robot layer* provide the actual robotic services based on the Robotic Operating System (ROS). The use of REST architecture to provide services makes them available to the great majority of devices, since the only system demand of the is an Internet connection. ROS is a wide-employed operating system for robots, which makes this system very easy to be applied to many situations.

In (MAHMOUD; JAWHAR; MOHAMED, 2016), a software-based architecture for UAVs and Wireless Sensor Networks (WSNs) is proposed. In this system, a Controller layer abstracts the physical nodes (UAVs and sensors) and provide APIs for the intermediate Orchestration layer, which menages the missions. On the top of the architecture, the Application layer provides a user-friendly interface for mission requesting and monitoring. The authors sustain that loosely coupled architectures such as this makes network reconfiguration easier and support redundancy, increasing the system reliability.

The work from (YAPP; SEKER; BABICEANU, 2016) presents a framework to enable the UAV as a Service (UAVaaS) paradigm, in which UAVs in a hub are allocated through an Internet interface to provide services. The framework supports mission scheduling, UAV assignment and supervision during the mission. All communications and task management is handled by a centralized Cloud coordinator.

The Dronemap Planner presented in (KOUBAA *et al.*, 2017) is a Cloud-based management system that aims in providing UAV access and control through the Internet. It allows user to schedule missions dynamically, using the MAVLink protocol to establish the communication between UAVs and the system management.

Finally, (ALWATEER; LOKE; RAHAYU, 2018) presents an approach and a simulation study on UAV service provision. Each UAV has an embedded server that answers to request made by users through a smartphone app. The authors analyze through simulations how the number of UAVs, their relative distance from control stations and the frequency of client request impact on service attendance and delivery. It is stated that scheduling strategies that distributes the service load among the vehicles are more efficient than a simple queue.

3.4 Security

Security is always a challenge in any communicating system. In the IoT-enabled UAVs, each component has its own security vulnerabilities which can compromise the system as a hole. Even though the ecosystem components have different characteristics, some security requirements apply to all UAVs, IoT and Cloud Computing domains, as summarized by Table 2.

Moreover, each component's characteristics have a great impact on security solutions development. In the UAS domain, the limited computational resources of UAVs hinders the implementation of conventional security solutions, which are not efficient for real-time transmission required by UAVs (HE; CHAN; GUIZANI, 2017). The safety of UAS components is also important — the system must be able to determine the safety and the authenticity of both its internal and external components (PIGATTO *et al.*, 2016). Within the IoT, the high complexity of the systems will magnify the effect of its intrinsic vulnerabilities such as the resource-constrained devices and the use of an open transmission medium (AL-FUQAHA *et al.*, 2015; SICARI *et al.*, 2015). Also, device mobility will play a major role in IoT systems, making security even more challenging — it will be necessary to identify, recognize and authenticate new devices, as well as adapting devices and services to different environments and contexts (NAHRSTEDT *et al.*, 2016). The distributed, heterogeneous and virtualized environment of the Cloud make traditional network security measures no longer effective (HASHIZUME *et al.*, 2013).

Table 2 – Security requirements among the UAS – IoT – Cloud Computing ecosystem.

Availability	UAS	All elements in a UAS must perform without disruption during operation, even if the system is under attack (ALTAWY; YOUSSEF, 2016).
	IoT	Must cover *software availability*, providing services to users simultaneously, and *hardware availability*, ensuring the presence of devices at all times to provide the services (AL-FUQAHA *et al.*, 2015; LIN *et al.*, 2017).
	CC	Cloud products are available on-demand at any time (ZHOU *et al.*, 2010).
Confidentiality	UAS	The communication within the UAS cannot leak information (HE; CHAN; GUIZANI, 2017).
	IoT	Collected data should not be accessed by unauthorized parties. Customary solutions may not apply due to the enormous amount of data generated the resource limitation of many of its components (MIORANDI *et al.*, 2012; LIN *et al.*, 2017)
	CC	A client's data and computation tasks must be protected from both cloud provider and other clients (XIAO; XIAO, 2013).
Integrity	UAS	Exchanged data in communication links cannot be altered, intentionally or unintentionally (ALTAWY; YOUSSEF, 2016).
	IoT	Received data must be guaranteed to not be forged or tempered with, so that the system do not take wrong decisions which could cause damage to the system or users (LIN *et al.*, 2017)

	CC	*Data integrity* ensures that data is not lost, tampered with or compromised and *computational integrity* ensures that applications are executed without interference of malwares or malicious users (XIAO; XIAO, 2013).
Accountability	UAS	UASs need accountability mechanisms to ensure non-repudiation (ALTAWY; YOUSSEF, 2016) and make it possible to identify malicious intent in a compromised node or security breaches.
	IoT	IoT systems should be able to identify responsibility in case of non-compliance. This is a challenge due to the system heterogeneity (MAPLE, 2017).
	CC	The system should be able to track and verify the entity responsible for actions taken within the environment to handle incidents that may lead to security breaches (XIAO; XIAO, 2013).
Identification, Authentication and Access Control	UAS	The nodes within a UAS must be identified and authenticated before communicating. Access control policies are necessary to prevent unauthorized personnel from accessing sensitive data and/or control operations (HE; CHAN; GUIZANI, 2017; ALTAWY; YOUSSEF, 2016).
	IoT	In IoT, each *thing* has to be uniquely identified and properly authenticated. Access control policies are also necessary so that data collectors only provide information regarding a specific target to legitimate data consumers (SICARI *et al.*, 2015; LIN *et al.*, 2017).
	CC	Since the Cloud is distributed among different administrative domains, off-the-shelf identification and authentication strategies can not be enough for these systems (ALI; KHAN; VASILAKOS, 2015).
Privacy	UAS	There are two types of concern in UAS privacy: how to ensure the privacy of the user and its data (including location) (ALTAWY; YOUSSEF, 2016) and how to ensure UAVs will not violate other people's privacy with its camera (HAYAT; YANMAZ; MUZAFFAR, 2016).
	IoT	Sensed data can contain sensitive information regarding their owner or users should not be accessed by other parties (MIORANDI *et al.*, 2012; LIN *et al.*, 2017). However, privacy protection should not interfere with data source authentication (ITU-T Y.2060, 2012).

	CC	Client's data in a Cloud system is likely to reside among distrusted servers owned and maintained by the cloud provider, who must ensure the data privacy (XIAO; XIAO, 2013).
Built-in device security	UAS	The UAS should authenticate all of its software and hardware components. Intrusion detection Systems (IDs) can also be employed to enforce internal security (ALTAWY; YOUSSEF, 2016).
	IoT	Due to minimal human intervention and the sheer number of them, it is highly desirable that IoT devices are be able to detect, diagnose, isolate and even counterattack security threats (SICARI *et al.*, 2015; UHER *et al.*, 2016).
	CC	Due to Cloud virtualization, the hypervisor and the internal virtual networks demand their own set of security measures (HASHIZUME *et al.*, 2013; ALI; KHAN; VASILAKOS, 2015).
Trust	UAS	Many UAV systems are dependent on external signals (HARTMANN; STEUP, 2013), so the UAS must establish trust with signal providers for safer operation.
	IoT	Due to its on-demand connection characteristics, trust need to be negotiated and achieved between each IoT layer, between devices, between devices and applications and between users and applications for a safer operation (MIORANDI *et al.*, 2012; SICARI *et al.*, 2015; LIN *et al.*, 2017).
	CC	Trust is important between the Cloud provider and its clients, as well as between the provider and third-party software used by the Cloud (XIAO; XIAO, 2013).

Source: Rodrigues and Branco (2020).

Even though security and privacy are almost always pointed out as an important challenge on the field, many solutions still do not discuss security measures, There are, however, solutions that take one specific security aspect to implement. The work in (SHARIAT; TIZGHADAM; LEON-GARCIA, 2016) implements UAV-enabled publish/subscribe Video-as-a-Service (VaaS) for Intelligent Transportation Systems in which every exchange is signed. A broker is responsible for the key management system, allowing both access control (only verified publishers can share the data) and data confidentiality (only authorized users have access to the data). In work (HANNAN *et al.*, 2021), an authentication framework for Drone-based delivery service in which the customer is authenticated through a two-factor verification do the package is delivered.

More comprehensive security works include (MOHAMED *et al.*, 2017) and (LAGKAS *et al.*, 2018). In (MOHAMED *et al.*, 2017), the proposed UAVFog platform provides integration services, broker services, invocation services, location-based services, and security services.

Authorization and authentication and access control are envisioned among the security services provided, but at the time of publication, only broker, invocation and location-based services were included in the platform prototype. Authors from (LAGKAS *et al.*, 2018) propose a framework to protect IoT objects from "spoofing, signal-jamming, and physical attacks, RF and mobile-application hacking, protocol abusing, and firmware hacks/sabotage" with a lightweight security toolbox, vision-based solutions, and privacy prevention and anonymity techniques for mobile things. Security mechanisms include authentication, key management, firewalls and intrusion detection systems. Authors claims the framework was not yet validated, and no prototype was presented.

3.4.1 Context-Aware Security

UAVs can achieve greater flexibility and robustness through the use of *contextual knowledge* (BLOISI *et al.*, 2016). Context can help UAV operation in many aspects, such as user interface, path planning, task optimization and to adapt to different threat levels in the network.

Context can be defined as any information that characterizes the situation of an entity (ABOWD *et al.*, 1999). *Context awareness* represents the ability of using context information to adapt behaviors and operations without explicit intervention in order to provide better response to the users (TEMDEE; PRASAD, 2018).

Context information can be either *primary*, retrieved without any type of processing like sensor fusion or *secondary*, an information inferred or computed from primary context information. For example, the GPS position of a sensor is a primary context information, while the distance between two sensors is a secondary context information based on their location. It is important to point out that the same type of data can be a primary context information in one application and a secondary context information in another (PERERA *et al.*, 2014).

There are four essential stages in a context-aware application. *Sensing* stage focus on gathering relevant environmental data that reflects the current state of the system. In *Analysis* stage, the raw collected data need to be modeled into context data and represented in a way that can be understood by other machines. *Reasoning* stage focus on how to interpret of context data and, depending on the context, make a decision following a chosen technique to reach a desired state. Finally, the *Adaptation* stage focus on selecting and applying the necessary functions and/or changes to the system (TEMDEE; PRASAD, 2018).

There are several context modeling techniques such as key-value, markup schemes, graphical based, object-based, logic based, and ontology-based modeling. Likewise, different context reasoning techniques are available, involving different strategies like supervised learning, unsupervised learning, rule-based, fuzzy logic, ontological reasoning and probabilistic reasoning. For a broader vision on context modeling and reasoning, the reader can refer to (PERERA *et al.*, 2014). All techniques have advantages and disadvantages, so multiple techniques are recommended to provide context-aware applications (TEMDEE; PRASAD, 2018).

As systems become dynamic, so need their security, making static solutions no longer effective (YUAN; ESFAHANI; MALEK, 2014). In this case, context information can be used to reconfigure or enhance the system security, creating a *Context-Aware Security* solution (WRONA; GOMEZ, 2006).

Context-Aware Security enables UAVs to adapt to different kinds of environments, increasing its safety, overall security and survivability. In a benign environment, security mechanisms can be less robust and more efficient, while in malicious environment strong security solutions are needed (CHIGAN; LI; YE, 2005). It is possible then to keep communication safe more efficiently, a highly desirable property in UV systems.

Authors from (CHIGAN; LI; YE, 2005) propose a security framework which aims at providing necessary security services through all network layers while minimizing the potential security redundancy and resource consumption. For this, the selection is made in two steps: an offline selection model and an online self-adaptive control module. In order to select the protocol set, all protocols are classified with two indexes: a security index (higher SI means more resistant protocols) and a performance index (higher PI means lower performance cost). On deployment, the MANET nodes can perceive if the threat level increases or decreases, triggering a negotiation for a new set of protocols. SI and PI values were estimated for 6 different protocol sets.

The work in (KONG *et al.*, 2002) presents an adaptive security framework for battleground UAV-assisted networks. The authors discuss two different operation modes: in the infrastructure mode, UAVs act as a central authority for certificate issuing and authorization, while in the infrastructureless mode the functions are distributed among all network nodes. It is possible to switch between operation modes seamlessly by means of backup CA keys and employing e-voting systems in a distributed manner.

The work in (ZERMANI *et al.*, 2017) presents a health management model based on Bayesian Networks for adapting UAVs to failures during operation, making its firmware adaptable to context.

A very important point in context-aware applications is that any changed aspect will impact on the environment the entity is inserted in. Security enhancements are likely to degrade network performance, since extra measures can impact latency, overhead and traffic load (CHIGAN; LI; YE, 2005). Also, specially in critical systems, it is crucial that context information is trustworthy so the application do not take any measures based on uncertain or false information (WRONA; GOMEZ, 2006).

3.5 Final Remarks

In this Chapter, different aspects of UAV insertion in the IoT were presented. It is considered that UAVs can leverage on IoT in many different applications, and in some cases work as the IoT infrastructure themselves. There are, however, many challenges in this integration, some of them explored further in this thesis.

Even though there are research in UAV energy efficiency like energy harvesting and protocol efficiency, the gap which was not yet properly addressed on energy efficiency techniques for UAVs is the provision of a mission-aware approach that can potentially reduce energy consumption.

Enabling UAV telemetry through the Cloud is a common way research papers illustrates IoT-enabled UAVs: telemetry is either provided through a platform or the UAV is virtualized in a service-oriented architecture and made accessible through a Cloud/IoT infrastructure. However, this structure depends on the UAV having a direct connection with the server providing the telemetry, with no applications in which the UAV make use of relay nodes to deliver its services.

Security is always considered a very important challenge, but very few solutions are found, addressing specific security aspects. A usable, more encompassing security solution is yet to be found. This solution needs to be flexible so it can attend a heterogeneous, mostly resource-constrained environment. Different security measures may be needed depending on the situation the UAV is inserted in, so context-awareness is also desirable.

In this context, the HAMSTER (HeAlthy, Mobility and Security based data communication archiTEctuRe) was proposed, aimed at improving mobility, security and safety of Unmanned Systems. It consists of different platforms with specific functionality for communication security, device safety, task management and service management and provision. The high level design of HAMSTER is exposed in the next Chapter.

CHAPTER

4

SERVICE MANAGEMENT FOR UAVS

As stated before, SEMU (SErvice Management for UAVs is an extension of HAMSTER. The **H**e**A**lthy, **M**obility and **S**ecurity based data communication archi**TE**ctu**R**e (HAMSTER) is a data communication architecture designed to improve mobility, security and safety of unmanned vehicles and/or systems, presented in the PhD work by Pigatto (2017).

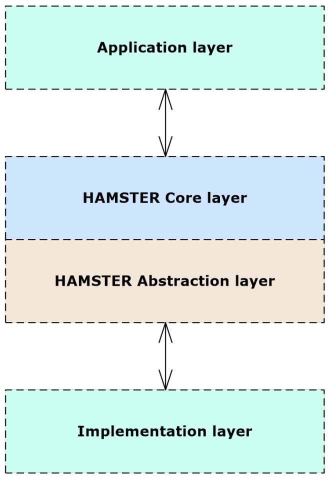
This PhD improved and expanded the architecture design by (a) allowing HAMSTER-enabled vehicles to exchange services through the Service Management Unit (SEMU) platform and (b) make use of security mechanisms to provide a more secure environment, including context-aware security, through the expanded *Cloud*–SPHERE platform and (c) make use of automated tasks through the improved NPplatform. This PhD also kick-started the development of a working prototype to can be used in real applications and future research. This Chapter provides a high-level description of HAMSTER and its platforms, including the ones developed during this work — SEMU, *Cloud*–SPHEREand NP.

4.1 HAMSTER

The HAMSTER is a data communication architecture designed to improve mobility, security and safety of unmanned vehicles and/or systems (PIGATTO *et al.*, 2016). HAMSTER provides communication, data security, device safety and service exchange functionality. In order to attend a highly heterogeneous environment with different processing, storage and communication capabilities, HAMSTER needs to have a great flexibility degree regarding all of its features. To achieve this, HAMSTER follows a layered model shown by Figure 3. As with communication networks, lower layers provide functionality to upper layers.

The *Application layer* is where the application resides, i.e., the navigation algorithm, the mission software, etc. The architecture itself has two layers. In the *Core Layer* lies all the HAMSTER platforms, besides message passing and logging features. This layer also defines an

Figure 3 – HAMSTER layered model.



Source: Rodrigues and Branco (2022).

Interface by which the user application may access HAMSTER functionality. The User Application is able to configure a UAV mission, services, security mechanisms and communication protocols and addresses. In order to be as flexible as possible and support different types of identification schemes, security mechanisms and service delivery models, HAMSTER has an *Abstraction Layer* in which generic interfaces are defined and invoked by the Core Layer to access concrete implementations provided by the *Implementation layer*. This way, the architecture allows the system engineer to couple whichever implementation is best for their application (e.g., communication technology or authentication/cryptography algorithms) without any modification on the Core or User Application layers.

4.1.1 The HAMSTER Unit

HAMSTER is based on the concept of a HAMSTER UNIT, which is a uniquely identified processing and communicating device that deploys HAMSTER. There are different TYPES of Unit depending on the device role in the system. A HAMSTER ENTITY represents an Unmanned Vehicle (UV) and is the main Unit in the system. The UV internal modules are HAMSTER OBJECTS if controlled by a different processing devices; a HAMSTER Object can either be a HAMSTER MODULE if the component has a single functionality/device or a HAMSTER CLUSTER if multiple devices are controlled by the same Unit. A HAMSTER SUPER ENTITY represents a central Entity in the system (such as a “leader UAV” or a GCS) which has additional management attributions. Finally, two support Unit types are envisioned up to this point to communicate with the aforementioned HAMSTER Units: a HAMSTER CLOUD, representing a Cloud/Edge node, and a STANDALONE Unit, representing a sensor/actuator able to communicate with other HAMSTER UNITS.

4.1.2 HAMSTER Functionality

HAMSTER Core Layer provides different functionality for the Application Layer, as shown by Figure 4: identification, communication security, service management and task management, discussed in this Chapter, and other auxiliary functionality such as message exchange, communication configuration and logging. Core Layer functionality makes use of interfaces available in the Abstraction Layer and made available by the Implementation Layer.

Part of HAMSTER functionality is implemented by specific platforms. The *Cloud*–SPHERE (Security and safety Platform for HEteRogeneous systEms connected to the Cloud) (RODRIGUES; PIGATTO; BRANCO, 2018a; RODRIGUES; BRANCO, 2020) is responsible for communication security and safety management; the SEMU (Service Exchange Management Unit) (RODRIGUES; BRANCO, 2020; RODRIGUES; BRANCO, 2019; RODRIGUES; BRANCO, 2021) provides service management and exchange functionality; the NCI (Node Criticality Index) (RODRIGUES; PIGATTO; BRANCO, 2020) is a rich index used to assess the operation criticality and enable better decision making on Quality of Service (QoS), security and safety based on data sensitiveness and device health; finally, the NP (Navigation Phases) Platform (RODRIGUES; PIGATTO; BRANCO, 2018b) manages an ENTITY task and allows its OBJECTS to be turned on or off according to the task stage, improving safety and energy efficiency.

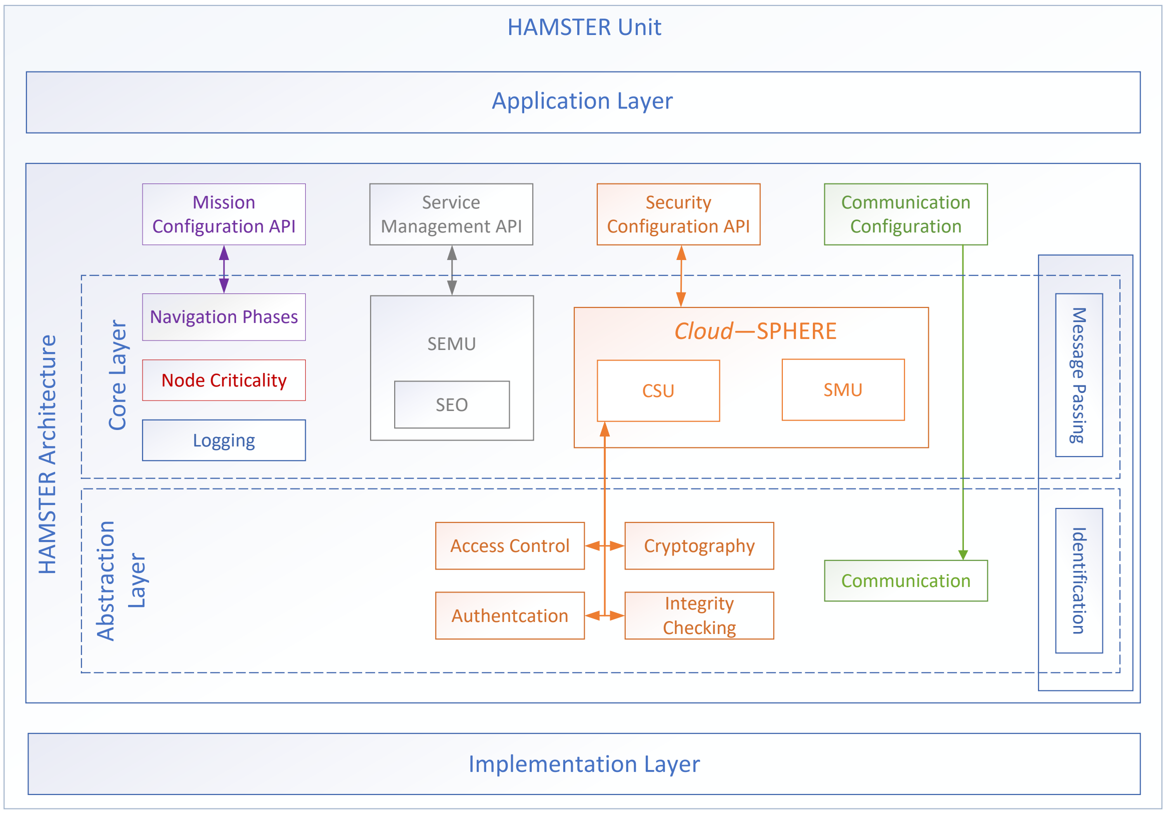
4.2 Identification in HAMSTER

Identification strategies will have a major importance in both UAV regulation and their integration into the IoT. Since public authorities like the FAA (Federal Aviation Administration) are likely to demand owners to register their UAVs, it would be possible for them to centralize the identification process. A governmental identification system has the advantages of permanence (manufacturers can go out of business) and local law compliance (while manufactures would need to adapt to all countries they sell products to). On the other hand, a global ordinance in what kind of information constitutes a proper identification (e.g., manufacturer, model, serial number) would be necessary so that any UAV can be correctly registered and identified in any country.

Digital certificates are another possibility for identifying UAVs, providing at the same time verifiable identification and public key cryptography support. However, this could be a costly choice for small UAVs. To embrace all UAV types, different policies could be adopted such as a multi-level UAV identification and registration process. Small, domestic-use UAVs would require a simpler identification (such as an ID/password pair stored in registration process), while commercial or bigger UAVs would require a more strict identification (such as with a verifiable identification authority), which could even include tracking information while in the air. Some examples of identification models are:

- SELF-IDENTIFIED – The UAV provides its own identification without any endorsement

Figure 4 – HAMSTER platforms and other abstractions illustrated in the layered architecture.



Source: Elaborated by the author.

from any authority.

- IDENTIFIED BY PROVIDER – The UAV manufacturer endorses the identification, but with no information on cryptography.
- CERTIFIED BY PROVIDER – The UAV has a digital certificate provided by its manufacturer.
- IDENTIFIED BY PUBLIC AUTHORITY – The UAV has its identification endorsed by a public authority, without verification of cartographic information.
- CERTIFIED BY AUTHORITY – The UAV has a digital certificate endorsed by a trusted Certificate Authority (CA).

Different identification models can also translate to different levels of device trust (for instance, self-identified devices would have a lower level of trust than certified devices and therefore could not access sensitive information from other UAVs), increasing the overall security of the system. The presence of an infrastructure aids in the identification process and facilitates digital certificates verification by accessing trusted CAs through an Internet connection.

In HAMSTER, all identification (Unit identification, service identification, algorithm identification) is dealt by a single interface that can be specialized to any identification structure necessary.

4.3 Communication Security in HAMSTER

Security is a paramount requirement in any communicating system that translate in specific requirements. Besides a verifiable identification, the HAMSTER UNIT needs to gain access to the system (Authentication). The system must control which units have access to each kind of information (Access Control) and all units must exchange data securely (Information Security).

4.3.1 Authentication

Authentication is the first step to secure communication and a key requirement for establishing trust between all actors in the communication (users, UAVs, infrastructure providers). Without proper authentication, the system can be illegally accessed and suffer false information injecting. The authentication should preferably be mutual and be done before the exchange of data between devices, continuing throughout the communication lifespan. HAMSTER adopts an “Almost Deny All” approach to authentication: all UNITS are considered unreliable until authenticated.

Similarly to identification, different authentication strategies can be implemented depending on the device computational capacity and the identification scheme employed, with each strategy being translated to different levels of trust. Other security-related tasks can be done together with the authentication, such as cryptography key exchange.

4.3.2 Access Control

After a node is authenticated, there might still be data or resources that are too sensitive to be securely transmitted with the authentication and/or cryptography algorithms used or should not be accessed by other UNITS at all. Therefore, the nodes need to determine which resources of the system, including both aircraft and the GCS, can be accessed.

This is done by implementing access control policies, which aims to prevent the illegal access of the GCS or aircraft by unauthorized personnel. Lack of access control policies can result in false information injection, unauthorized disclosure of information and other issues. The implementation can be identity-based, as done in the Cloud, or role-based, with rights being updated dynamically without changing role assignments.

4.3.3 Information Security

During a communication, security measures must be taken so the information (a) is not accessible to unauthorized parties (confidentiality), (b) is not modified accidentally or intentionally during its transmission (integrity) and (c) has a verifiable origin (authenticity).

These requirements are related to authentication and access control, and can be complemented with other security measures like proper authorization and cryptography.

4.3.4 The Cloud–SPHERE Platform

The **S**ecurity and safety **P**latform for **HE**te**R**ogeneous syst**E**ms connected to the **Cloud** (*Cloud*–SPHERE) is the HAMSTER security platform which allows the Application Layer to configure security strategies and algorithms. It handles communication security and safety management through two different units, CSU and SMU, as shown by Figure 5.

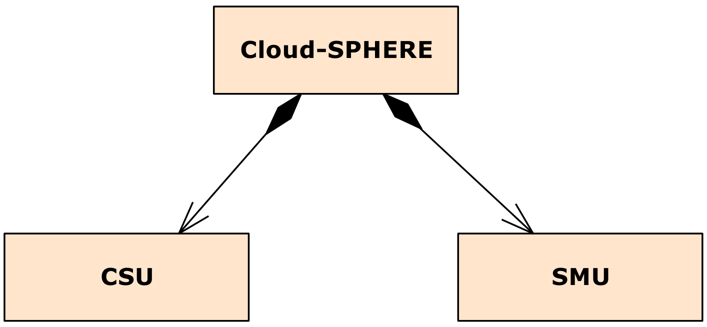


Figure 5 – *Cloud*–SPHERE and its Units, CSU and SMU.

Source: Rodrigues and Branco (2022).

4.3.4.1 Central Security Unit

The Central Security Unit (CSU) is responsible for securing both internal (i.e., between a HAMSTER ENTITY and its internal OBJECTS) and external (e.g, between ENTITIES) communication. Up to this point, three different tasks are delegated to the CSU: authentication, access control and information security. HAMSTER do not determine which strategy/policy/algorithm to use in any of these tasks, but rather let the system engineer to couple in the Implementation Layer which strategies are best for the application. The proposed flexibility allow the system to have different strategies available in the same system and using each of them depending on device trust and perceived risk.

4.3.4.2 Safety Management Unit

The Safety Management Unit (SMU) controls the overall safety of HAMSTER UNITS by monitoring its health and searching for misbehavior or failures. If any HAMSTER OBJECT malfunctions or fails, it can be blocked from communicating with others, improving the UNIT safety.

Health monitoring verifies the HAMSTER UNIT operation searching for errors or anomalies, and can be deployed locally within HAMSTER MODULES and HAMSTER CLUS-

TERS or remotely using the Cloud infrastructure to execute for example computational-costly Artificial Intelligence (AI) algorithms.

Overall UAV safety is implemented within HAMSTER entities, taking a global look in the system and taking appropriate action for these vehicles regarding its own safety and nearby systems or people. The communication between the modules and its entity can also monitored through a heartbeat signal, which would also function as a continuous authentication during the device operation.

4.4 Contex-Aware Security in HAMSTER

Given the heterogeneity of the UAS - IoT - Cloud ecosystem, there is no feasible way to provide a single strategy to ensure all information security requirements — a flexible solution is needed.

One way to achieve this goal is to establish different levels of information security, with each level requiring different security measures. For instance, less critical information such as temperature or altitude sensors would require simpler, if any, security mechanisms, while more critical information such as the GPS positioning or the camera feed would require a more sophisticated encryption scheme or authentication level.

There could be a number of ways to determine how sensitive or critical an information is. One of the possible solutions is to use *contextual* information that translate specific conditions of the UAV task in perceived risk.

Context-Aware Security enables a system to adapt itself to different contextual situations, increasing its safety, overall security and survivability. Security mechanisms can be employed dynamically depending on the application environment (e.g., a risky environment would apply stronger security than a benign environment). The functionality brings many challenges. First, stronger security measures are likely to degrade network performance and can impact latency, overhead and traffic load (CHIGAN; LI; YE, 2005). Also, context data has to be trusted — false context information can lead to disastrous consequences (WRONA; GOMEZ, 2006).

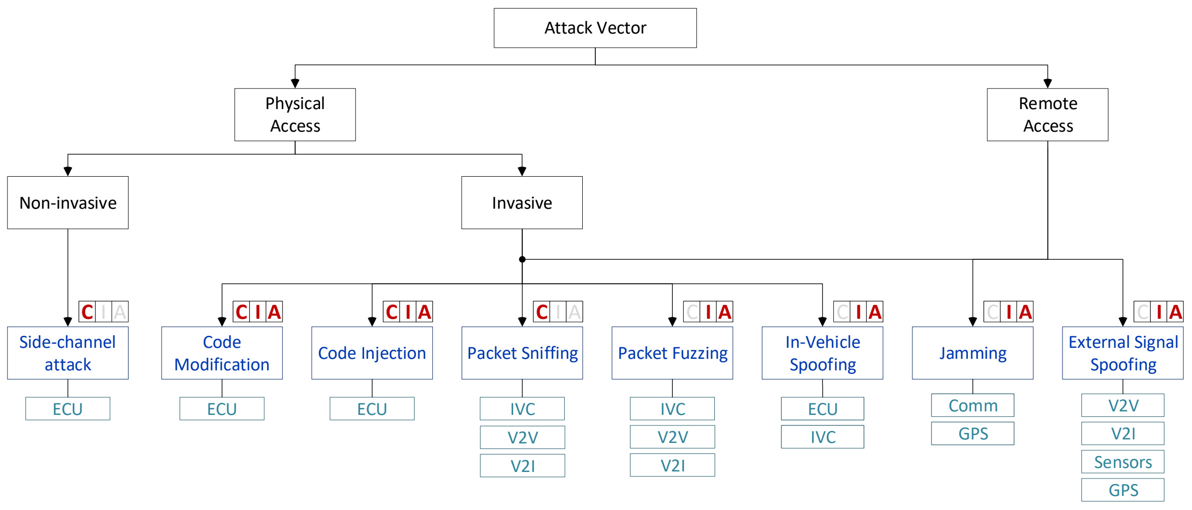
To deal with context for UAVs, HAMSTER deals with three different types of contextual information, based on the work presented in (TURNER, 1998) for an autonomous underwater vehicle and later in (BLOISI *et al.*, 2016) for an autonomous ground vehicles. In these works, contextual information is composed by three elements: environmental or *external context*, task-related or *mission context*, and operation-related or *internal context* information. All context information is translated to a Perceived Security Index (PSI) that can be used to make decisions regarding security mechanisms, service provision and consumption and for UAV safety and self-preservation.

4.4.1 External Security Context

External Security Context relates to threats that are not a direct consequence of UV operation, but rather caused by external agents. External context allows the UAV to perceive its surroundings and adapt itself accordingly. This is translated mainly in any cyber-threat the UAV is vulnerable to and also hijacking.

Figure 6 brings a taxonomy of attacks for autonomous vehicles provided by (THING; WU, 2016) and later expanded for UAVs by (KRISHNA; MURPHY, 2017). The taxonomy classifies the attacks according to physical or remote access to the UAV, and in invasive or non-invasive attacks. Figure 6 also brings possible targets of each attack, as in Electronic Control Units (ECU), Sensors and GPS and the communication links (Internal Vehicle Communication — IVC, Vehicle-to-Vehicle — V2V and Vehicle-to-Infrastructure — V2I). Moreover, there is an indication of which element of the CIA triad (Confidentiality, Integrity, Availability) could be compromised for each attack, based on UAV attack taxonomy presented in (JAVAID *et al.*, 2012).

Figure 6 – Taxonomy of attacks for UAVs.



Source: Adapted from Thing and Wu (2016).

HAMSTER has some in-built security mechanisms and policies that help mitigate some of the attacks presented. First, the use of an “Almost Deny All” authentication approach adopted by *Cloud*–SPHERE Platform to authenticate HAMSTER OBJECTS makes the UNIT unable to operate unless all critical components are properly authenticated. This makes attacks of code modification and code injection more difficult on those modules. Also, the use of cryptography in all types of communication (HAMSTER-specific or user-defined messages) greatly decreases the risk to confidentiality, provided that the key-distribution scheme is successfully executed with no disclosures.

4.4.2 Mission Security Context

Mission Security Context relates to the tasks the UV has to perform in the application it is being used, and the risks associated with them. There are many aspects that can be considered, such as the mission target and its path from initial location, sensitiveness of data and component usage.

Regarding information security, data sensitiveness is a very important aspect to be considered. Depending on the application, stored data need to be encrypted, and self-destruction policies may be necessary if the environment becomes hostile.

4.4.3 Internal Security Context

Internal Security Context relates to the UV internal state and is strongly connected to its safety and information reliability. Any component malfunction or failure impacts directly the internal context perceived risk. In HAMSTER, *Cloud*–SPHERE manages connection status and safety supervision of all UAV components through SMU. The component state contributes to its criticality index, measured by the ode Criticality Index (NCI), whose value depends on data sensitiveness, module importance and safety state.

4.4.3.1 The Node Criticality Index (NCI)

The Node Criticality Index (NCI) is a rich index to help determining single and global priorities for nodes within a network (PIGATTO, 2017). Its main goal is to provide a measurable level of criticality associated with individual nodes, allowing better decision making on Quality of Service (QoS), security, safety and prioritization approaches for modules, clusters of modules and entities. NCI is flexible enough to encompass different sets of goals based also on mission information.

NCI was designed to work in three different situations within an UAS: (i) the network connecting basic and mission-specific internal modules individually, (ii) the network connecting internal clusters of modules, and (iii) the external network among unmanned vehicles and eventual infrastructure entities

4.4.4 Perceived Risk Evaluation

As discussed before, HAMSTER aims at translating contextual information in a Perceived Security Index. For each HAMSTER UNIT in the scenario, PSI value is given by the combination of every contextual component:

$$\Psi = \sqrt{\frac{\psi_{ex}^2 + \psi_{mi}^2 + \psi_{in}^2}{3}}$$

where Ψ is the Perceived Security Index and ψ_c is a partial security index related to context c.

Initial PSI values are dependent on the vehicle type and application, assuming subjective values depending on the specialist doing the security analysis. An initial estimation of PSI for external context can be obtained by performing a risk evaluation for the attacks shown in Figure 6, taking into account their likelihood and impact. Available security mechanisms also have to be considered, as well as any indicator that the UV is suffering an attack. An initial estimation of PSI for mission context can be obtained by evaluating the localization of the UAV (for instance, a rural or populated area), risk of collision with other vehicles or people, and mission data sensitiveness. Finally, initial PSI estimation for internal context can be taken from NCI values for every UAV module.

Since HAMSTER leans heavily on abstractions, different strategies for security mechanisms like cryptography, identification or authentication can be made available to the application. Every strategy has an impact on the perceived security and has to be considered in PSI calculation.

PSI values are discretized to five levels (Table 3). $\Psi = 0$ means that the environment is hostile and there is none or insufficient security measures for UV operation; $\Psi = 5$ means a benign environment with more than sufficient security measures in place.

Table 3 – Perceived security levels in HAMSTER.

5	High security
4	Medium to high security
3	Medium security
2	Medium to low security
1	Low security
0	No security

Source: Rodrigues, Pigatto and Branco (2020).

HAMSTER platforms and user application can refer to these values when making decisions and assuring UAV security and safety. For instance, HAMSTER UNITS may have a minimum PSI to operate normally. If the PSI value falls below this security threshold, the UNIT initiates an emergency protocol — switch the cryptography algorithm for one stronger, re-authenticate all communicating peers or in the case of a HAMSTER ENTITY, turn off non-critical modules, abort the current task and return home.

General security decisions can also be leveraged by PSI values. For example, a UAV performing a mission on a non-populated area (lower PSI for external context) may use a symmetric cryptography algorithm, which is usually lighter and has a lower security index. If a threat is identified (PSI for external context increases), the UAV could switch to an asymmetric algorithm to improve PSI or decide to abort the mission for UAV self-preservation.

Service provision and consumption is also affected. Unauthorized or defective components are not eligible for providing reliable services. A malfunction or disconnection results in services being discontinued. Also, a HAMSTER UNIT can decide on a minimal PSI level for service exchange with other UNITS.

4.5 Service Management in HAMSTER

Services are one of the most important components in an IoT system; they make devices available to be accessed any time, anywhere, by anyone.

One of the main challenges for autonomous service exchange is how to sort between available services the one that fits the device's necessity the most. Service discovery can be done by both human and machines users. Human IoT users will not require a service "video feed from UAV with serial number 123456 at GPS position X", but "live traffic condition on Main St.". Machines, however, would need a natural language interpreter to request services the same way.

Also, different providers may offer the same service with different names and descriptions, or with incomplete/erroneous descriptions, which could lead to improper service requests by the mislead device. Even though some UAV service classification such as found in (MAHMOUD; MOHAMED, 2015) exist, a standardized way of describing services to both human and machine users is necessary. One way to achieve this is to propose a standard set of definitions for each type of services that can be provided by UAVs or UASs. Depending on the formality level, it can be done by a simpler glossary or a formal ontology. There are many attributes that can be used to categorize a UAV service, such as:

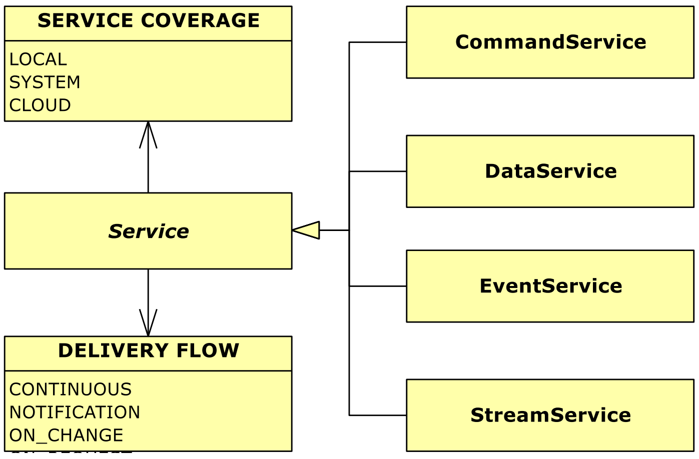
- FANTASY NAME – Human-friendly description of the service. This would be used by human clients to discover, request an consume a service.
- SERVICE CODE – A machine-readable code correspondent to a service. This would be used by machine clients to discover, request and consume a service.
- OFFICIAL NAME – Technical description of the service, so that developers know exactly which service they are providing and/or consuming.
- DATA SENSITIVENESS – Determines the sensitiveness level of delivered data, which could assume as many levels as necessary, such as *low*, *medium*, *high* and *critical*.
- MINIMUM AUTHORIZATION LEVEL – Defines which is the minimum security clearance the client need to have in order to access the service (for example, no authentication required, or verified digital certificate required).
- BILLING SYSTEM – Describe if and how the service will be billed.
- COMPOSITION – Describe if the service is simple (a single source of data) or composite (many services aggregated in a new service).

4.5.1 Services in HAMSTER

Figure 7 shows how services work in HAMSTER. Up to this point, there are four service types: a *Command Service* enables a client to send commands to a device (e.g., an actuator); a

Data Service is focused on data sharing (e.g., publishing data from a sensor); an *Event Service* notify clients about events of interest (e.g., an asynchronous task was completed); finally, a *Stream Service* focus on the transmission of continuous data (e.g., camera stream). Besides its type, each Service has an ID, a coverage and a delivery flow.

Figure 7 – Services in HAMSTER.



Source: Rodrigues and Branco (2022).

Ideally, every service has a unique identification, which can include human-friendly service description to alphanumerical codes for Machine-to-Machine communication.

The *service coverage* tells the service availability reach in the system. A *local* service is only available to internal Objects of a Unit; a *system* service is available to other Entities in the same system and its respective Objects; a *cloud* service is provided in a Cloud/Edge server and can be accessed by any authorized user. In situation when a Unit provides a service with a grater scope (e.g., a HAMSTER MODULE providing a CLOUD service), HAMSTER has created the concept of *service propagation*, in which one Unit acts as a proxy to the services from another Unit located in a lower network hierarchical level. As an example, let's imagine a UAV (HAMSTER ENTITY) and its GPS board (HAMSTER MODULE) providing data as a *Cloud Service*. It would be unfeasible and even illogical to have all service clients connected directly to the GPS board. In HAMSTER, the GPS position service is *propagated*, being offered by its containing UAV and whichever Unit the UAV is connected to in higher network levels (e.g., a GCS or a Cloud server). With propagation, only UAV internal Objects connect directly to the GPS board, allowing the use of more modest devices in Objects and improving network traffic.

The *delivery flow* determines how frequently a service is delivered. In the *continuous* flow, the service is delivered periodically to the client; in *notification* flow, the client is notified of an event occurrence; in *on change* flow, the service is delivered every time there is a change in its value; in *on request* flow, the service is delivered on demand. Not all delivery flows are available to different service types. Table 4 shows the valid combinations of service types and delivery flows.

Table 4 – Valid delivery flows for each service type.

Delivery Flow	Service Type			
	Command	Data	Event	Stream
Continuous	✗	✓	✗	✓
Notification	✗	✗	✓	✗
On change	✗	✓	✗	✗
On request	✓	✓	✗	✓

Source: Rodrigues and Branco (2021).

For a *service provider*, HAMSTER enables service registration, propagation, management and delivery.

- *Service Registration:* is done by the service provider during UNIT initialization by the Application Layer, which can register services of all envisioned types described in Figure 7. Different service types require different information during registration.
- *Service Propagation:* the last step on HAMSTER initialization is to start all services, analyzing their *coverage* and making the service propagation if necessary.
- *Service Management:* the service provider is able to suspend or restore a given service at any time. If the service is suspended, all clients are notified and all connections are closed. If the service is restored, the client needs to reconnect with the provider.
- *Service Delivery:* how the service is delivered depends on its *delivery flow*. In the continuous flow, a *service message* is sent periodically to the client; in the notification flow, the message is sent only when the Application Layer makes a notification; in on change flow, a message is sent when the service history changes; finally, in on request flow, the message is delivered once the request is finished.

For a *service client*, HAMSTER enables service discovery, request and consumption.

- *Service Discovery:* allows a client to make broadcast requests for a service in a dedicated communication channel — because of that, services should be identifiable in a way tat these queries are responded accurately taking into account their coverage.
- *Service Request:* once there is a desired service provided by another HAMSTER UNIT, the service client can request it to the service provider. The answer to these requests will depend upon service provision coverage, data sensitiveness and minimum authorization level required from the service consumer. Therefore, these information will have to be embedded upon service request.
- *Service Consumption:* service delivery can be done either synchronously or asynchronously depending on the chosen delivery flow. The service client must be ready to receive the

different service messages that can arrive in response to the request made.

4.5.2 The SEMU Platform

HAMSTER allows any UNIT to register, manage, provide, discover, request and consume services. For these functions, HAMSTER provides the Service Exchange Management Unit (SEMU) (RODRIGUES; BRANCO, 2019; RODRIGUES; BRANCO, 2020; RODRIGUES; BRANCO, 2021) platform and the Service Exchange Operator (SEO). The first deals with service registration and/or propagation and provides an interface between the Application Layer and the SEO, while the second deals with the service requests, management and delivery. In the future, security aspects dealt by *Cloud*–SPHERE and NCI can be integrated to service management and exchange to make the data exchange security-aware.

4.5.3 Considerations on Security

It is important to point out how the service provision can affect the overall security of the system. Unmanned Aerial Systems (UASs) have some inherent vulnerabilities, and can be attacked by its physical elements or the communication links. The wireless channel is intrinsically vulnerable to many attacks, such as eavesdropping and masking (HE; CHAN; GUIZANI, 2017). Sensors are vulnerable to spoofing or falsification (ARCHIBALD; SCHWALM; BALL, 2017), which can compromise the entire UAV operation and safety. The control station can be targeted by viruses, malwares and other attacks common to software-based units (JAVAID *et al.*, 2012). UAVs are resource-constrained devices, and therefore are not able to support conventional security solutions. Besides, the real-time transmission required by UAVs require more efficient, resource-saving security mechanisms (HE; CHAN; GUIZANI, 2017).

IoT technology also present many vulnerabilities. Since the majority of IoT are resource-constrained devices, sometimes unable to implement security mechanisms at all. Also, the system makes use of the wireless medium, intrinsically insecure, to share information between the nodes (AL-FUQAHA *et al.*, 2015) and insecure web/Cloud/mobile interfaces that can be easily compromised (LI, 2017).

Even though the Cloud environment has the potential to be a great ally in enabling IoT and helping UAVs to be integrated into it, it also has many vulnerabilities. The distributed, heterogeneous and virtualized environment of the Cloud make traditional network security measures no longer effective. The use of insecure interfaces can also present many problems for systems relying on this technology (HASHIZUME *et al.*, 2013).

HAMSTER can help improving security by design. First of all, service registration must be done after HAMSTER unit authentication. This way, any registered or propagated service comes from a minimally trusted source. Different authentication strengths may result in different service trust levels. A requesting user will be able to require a service while aware of its security and trust level.

Also, each service may require from its clients a minimum *Security Level* to accept their request, which will depend on data sensitiveness and necessary client security clearance. The client security level is related with device trust, and it will depend on the client domain, the identification/authentication techniques used and the security mechanisms available (such as cryptography). The Security Level of each device can be handled by *Cloud*–SPHERE, and the final objective is to translate trust in a quantitative manner that can be ranked, similar in what is done with PSI.

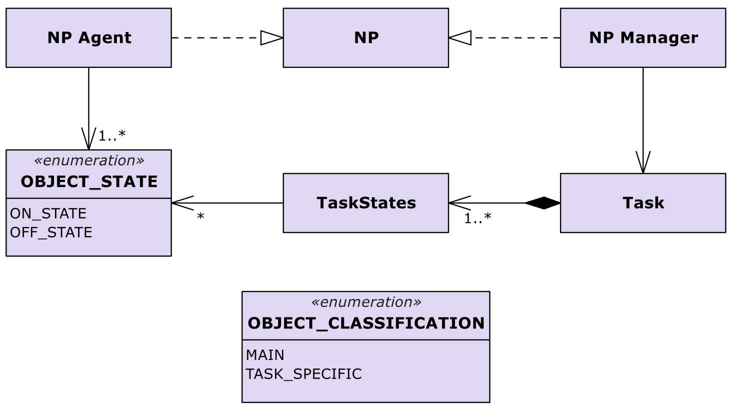
Given its flexibility with abstractions, a more computational-resourceful UNIT may have available different strategies / algorithms for authentication, cryptography and integrity checking. This enable this UNIT to provide and consume services with different security levels for clients from different origins — an "internal" client (i.e., a client part of the same system) is more likely to be perceived as more trusted and able to access more sensitive data, while an "external" client would not be able to access the same type of data.

4.6 Task Management in HAMSTER

HAMSTER allows the Application layer to manage a HAMSTER ENTITY task and control the state of its OBJECTS for every task stage through the Navigation Phases (NP) platform (RODRIGUES; PIGATTO; BRANCO, 2018b). NP makes it possible for a vehicle to change the object states so the overall task execution is more optimized and energy efficient. Together with services, all actions of a HAMSTER enabled system can be done autonomously.

Figure 8 presents the Navigation Phases platform. This is the only Platform in HAMSTER that has clear division depending on the UNIT TYPE. The NP is called an NP MANAGER or an NP AGENT if the Unit is an ENTITY or an OBJECT, respectively.

Figure 8 – Navigation Phases (NP) platform.



Source: Rodrigues and Branco (2022).

The NP MANAGER is responsible for handling the Unit's TASK. Each Task has different *stages* with an associated TASK STATE. In NP MANAGER, each Object is classified as a *MAIN* component (an essential Object) or a *TASK SPECIFIC* component (a non-essential Object). For each of these groups, the TASK STATE has an OBJECT STATE that describes how each group should operate. The NP AGENT is present in each Object and can store the instantaneous OBJECT STATE or the states for all Task stages. Each Object also has an *Emergency State* that describes its operation in emergency situations.

When the Task start is triggered, the NP Manager can either *distribute* the Tasks States from all Task stages to all Objects or send the current expected state for every Object in the Unit through the NP Agent. When the Task stage is changed, the NP Manager will either notify the Agents that a new stage has begun (if the mission was distributed earlier) or notify all Objects of their new Object State. The NP Manager also notify all NP Agents in a Unit if an Emergency state has been triggered. The NP Agent then take the appropriate actions to conduct the object to its configured state in the specific stage.

4.7 HAMSTER Operation Scenarios

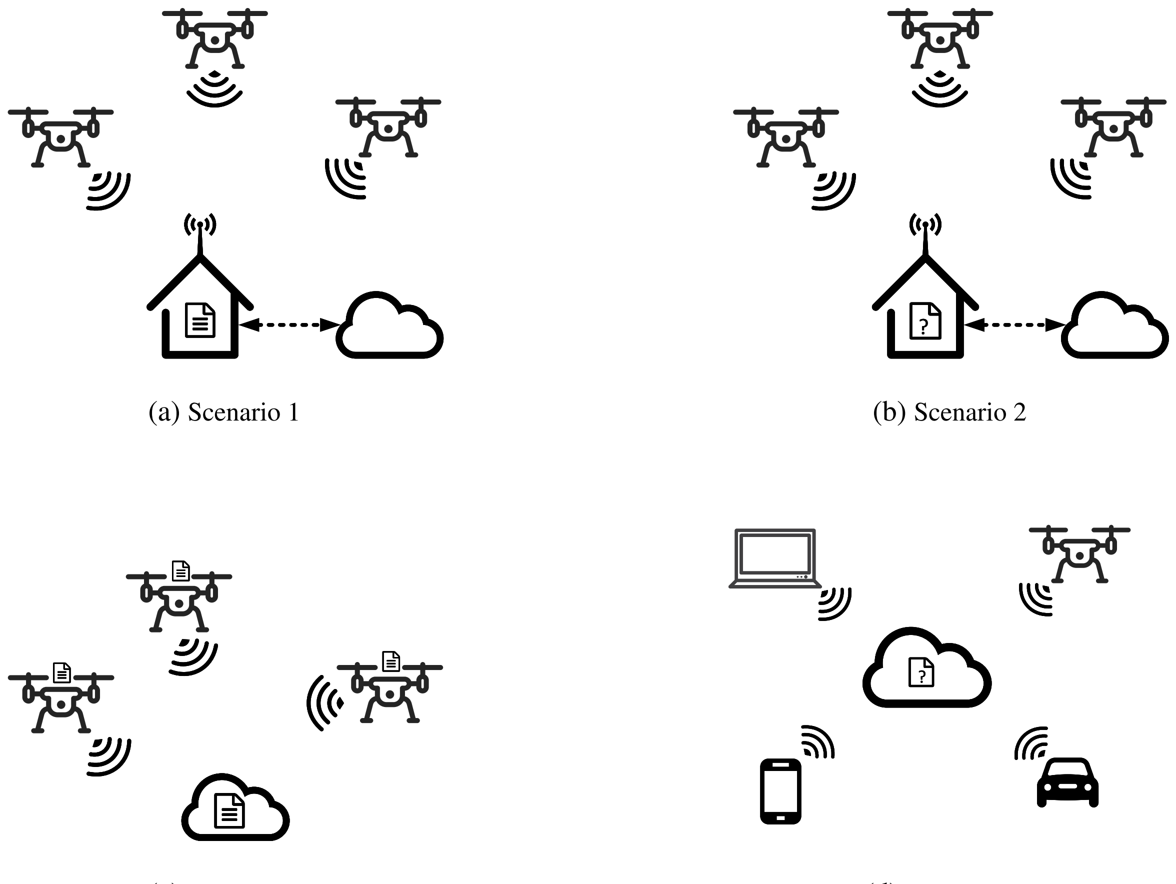
Given the situations in which a HAMSTER-enabled UAV can operate, there are two distinct characteristics that greatly impact the system operation: (1) the presence of a CENTRAL UNIT in the system and (2) if each Unit knows beforehand who its communicating PEERS are going to be.

A central Unit enables a *centralized operation* in which the central ENTITY is responsible for identification, authentication and cryptographic key distribution. Usually, the system can be more secure because of the smaller attack surface, but it can be completely compromised if the central ENTITY is breached or have its availability or performance compromised due to the communication bottleneck. In contrast, *decentralized operation* relays much more on each individual Unit, which needs to identify/authenticate all peers, which can take longer, consume more power resources and degrade network performance. Decentralized operation have at the same time better survival and higher risk due to the larger attack surface.

In a *known formation operation* the Units know its communicating peers in advance, and both peer identification and service discovery becomes unnecessary. The system can also establish secure exchanges from the very beginning if authentication and cryptography keys are known beforehand. On the other hand, a compromised node may easily perform an injection attack. An *unknown formation operation* makes it much more challenging to establish trusted connections between Units.

Given the previous discussion, four operation scenarios for HAMSTER are envisioned, presented in Figure 9 and discussed next.

Figure 9 – HAMSTER operation scenarios.



(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

Source: Rodrigues, Pigatto and Branco (2018a).

4.7.1 Known Formation, Centralized Operation (Figure 9a)

In this scenario, the central Unit is able to store all the system's identification and security-related information such as cryptographic keys. Even service information can be centralized and service discovery becomes unnecessary. The central UNIT may also act as a mediator between other Units, i.e., if UNIT A needs to communicate with UNIT B, the authentication and key exchanged can be mediated by their central Unit. Examples of this scenario is a Unmanned Aerial System (UAS) with different UAVs and a Ground Control Station (GCS) centralizing the communication or the internal UAV operation, between HAMSTER MODULES or CLUSTERS and the HAMSTER ENTITY they are part of.

4.7.2 Unknown Formation, Centralized Operation (Figure 9b)

In this scenario, the central Unit may still act as a mediator, but only between Peers which have already connected and authenticated themselves. Since no prior information is known, an identification procedure is mandatory before authenticating any Unit. An example of this scenario is a collaborative Cloud application to which unknown UAVs connect to a Cloud/Edge server to consume services (like image processing or other resource-demanding tasks).

4.7.3 Known Formation, Decentralized Operation (Figure 9c)

In this scenario, each Unit knows its Peers, but there is no central Unit to mediate any operation. Therefore, all Units must identify/authenticate themselves and exchange keys with one another, which can cause an initial network performance degradation. An example of this scenario is a UAV swarm flying collaboratively to complete a task without Line of Sight (LoS) with any GCS.

4.7.4 Unknown Formation, Decentralized Operation (Figure 9d)

In this scenario, a Unit may receive requests from unknown Peers, handling identification and authorization procedures by its own. This is how the IoT scenario most likely will play out, with services being broadcasted and consumed by unknown Peers on-demand. Mutual trust is essential to perform service consumption securely, making this scenario the most challenging of all.

4.8 HAMSTER in IoT-connected UAV scenarios

HAMSTER can provide security to a large number of IoT-connected UAV scenarios. In scenarios where the Cloud is a manager from which external clients can request UAV services, the Cloud would be a HAMSTER SUPER ENTITY, any UAV would be a HAMSTER ENTITY and, if any UAV module such as GPS or camera has a separate processing unit, would be a HAMSTER OBJECT. Either the Entity or the Object would provide services that can either be simple services offered directly to the client (i.e, services with *cloud* coverage that are propagate through the system) or that are offered to the Cloud (*system* coverage) and then are combined in composed services offered directly from the Cloud unit to its clients (not necessarily though HAMSTER). This scenario is demonstrated in HAMSTER results (Section 6.5) with a UAV-enabled delivery service. The Cloud/GCS is the HAMSTER SUPER ENTITY, the UAV is the HAMSTER ENTITY and its Navigation Unit is a HAMSTER OBJECT that provide navigation services.

In scenarios where UAVs work as part of IoT infrastructure, different possibilities are possible. In a scenario where the service is persistent, the UAV could be the HAMSTER SUPER ENTITY providing computation and storage services to an IoT sensor as a HAMSTER ENTITY. If, however, the idea is to send IoT data to a Cloud service, the Cloud would then be the HAMSTER SUPER ENTITY, the UAV its HAMSTER ENTITY and all IoT sensors could operate as STANDALONE Units. This scenario is demonstrated in HAMSTER results (Section 6.6) with a UAV acts as a data collector (or a *data mule*) of different sensors. In this scenario, a

GCS or Cloud server is the HAMSTER SUPER ENTITY, the UAV is a HAMSTER ENTITY and has two HAMSTER OBJECTS: a Navigation Unit and a Connection Unit — this last one responsible to connect to IoT sensors acting as STANDALONE Units.

4.9 Final Remarks

In this Chapter, HAMSTERwas presented together with the main points developed during this PhD so the architecture evolved to SEMU. First, security aspects in this work was presented, including the *Cloud*–SPHERE platform and how the system could make use of contextual information to improve overall vehicle security and safety. Then, service management aspects were discussed and a description on how they are dealt in SEMU were presented. Also, it was shown how the applications can use automatic task management in their application through the NP platform. Finally, different feasible UAV application scenarios were presented and discussed.

This work make it feasible for UAVs to exchange services making use of available security mechanisms and automated task management to build complex applications. There is still a gap from theory to real life applications, which this work filled by developing a prototype presented in the next Chapter.

CHAPTER 5
THE PROTOTYPE

As previously said, HAMSTER is designed to be adaptable and flexible so devices with diverse resources and technologies can implement it. This is a key characteristic for dealing with heterogeneous systems. The prototype leans heavily on *abstraction* to provide generic interfaces (the Abstraction Layer from Figure 3) that are specialized to different algorithms and technologies (the Implementation Layer from Figure 3).

The prototype implementation is being done in C++, which allows great optimization and easy conditional compilation. This first version implements the *Centralized, known formation* scenario and supports authentication, cryptography, task management and service management and exchange.

In this Chapter, the operation and design model of every HAMSTER piece presented is detailed. Section 5.1 presents the HAMSTER UNIT; Section 5.2 presents the communication interface and describes how Units connect to one other; Section 5.3 explains the message exchange in HAMSTER; Section 5.4 presents the design of *Cloud*–SPHERE Platform; Section 5.5 presents the design of SEMU platform; finally, Section 5.6 presents the design of NP platform.

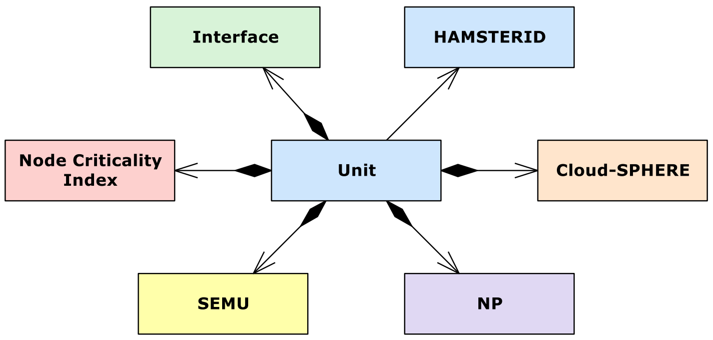
5.1 The HAMSTER Unit

A HAMSTER UNIT is any device/node that implements HAMSTER, and provides a HAMSTER Core Layer interface to the User Application. Each Unit is uniquely identified by an HAMSTER ID and has a specific TYPE depending on its role in the system:

- **STANDALONE**: represents a standalone component (sensor/actuator).
- **HAMSTER MODULE**: represents a component with single functionality/device (e.g., a navigation module).
- **HAMSTER CLUSTER**: represents a component with multiple devices (e.g., a sensor board);

- **HAMSTER ENTITY**: represents a Unmanned Vehicle (UV);
- **HAMSTER SUPER ENTITY**: represents a Central Unit in the UAS, such as a "leader UAV" or a GCS — this unit is present only in centralized scenarios (Section 4.7);
- **HAMSTER CLOUD**: represents a Cloud server that communicate with different Units or entire Unmanned Systems (USs);

Figure 10 – Generic view of a HAMSTER UNIT.



Source: Rodrigues and Branco (2022).

Each HAMSTER UNIT may contain different HAMSTER PLATFORMS, each of them deploying a different part of HAMSTER functionality. In Figure 10, a generic view of the envisioned HAMSTER UNIT is presented. Besides its unique HAMSTER ID, every Unit may contain the NCI, SEMU, NIMBLE, NP and *Cloud*–SPHERE platforms. In order to communicate with other Units, at least one communication INTERFACE must be present.

Not all platforms are present in every Unit TYPE. Table 5 relates which platforms are present in each Unit type.

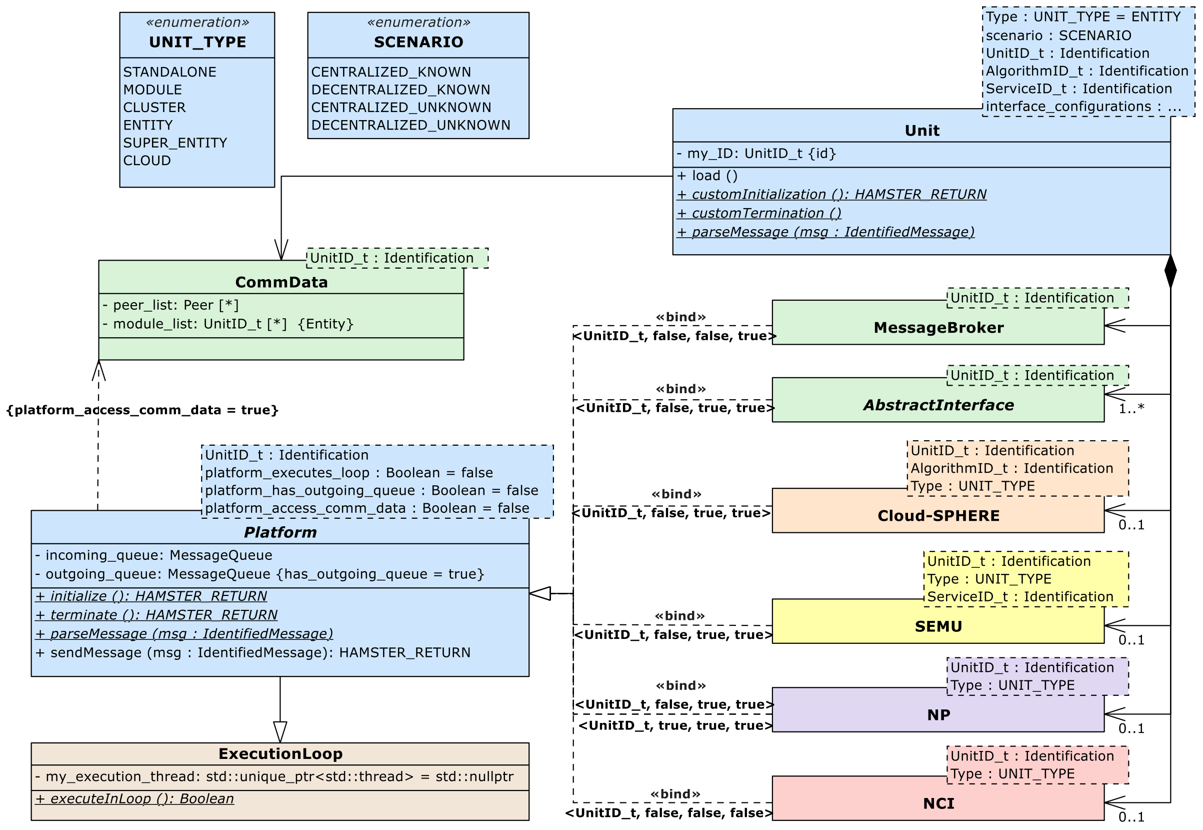
Table 5 – HAMSTER PLATFORMS for each Unit TYPE.

UNIT TYPE	*Cloud*–SPHERE	NCI	NP	SEMU
STANDALONE	✓	✗	✗	✓
MODULE	✓	✓	✓	✓
CLUSTER	✓	✓	✓	✓
ENTITY	✓	✓	✓	✓
SUPER ENTITY	✓	*	*	✓
CLOUD	✓	✗	✗	✓

* The presence of these platforms will depend if the Super Entity is an UV or not.

Source: Rodrigues and Branco (2022).

Figure 11 – Unit model in HAMSTER.


«enumeration»
UNIT_TYPE
STANDALONE
MODULE
CLUSTER
ENTITY
SUPER_ENTITY
CLOUD
«enumeration»
SCENARIO
CENTRALIZED_KNOWN
DECENTRALIZED_KNOWN
CENTRALIZED_UNKNOWN
DECENTRALIZED_UNKNOWN
Type : UNIT_TYPE = ENTITY
scenario : SCENARIO
UnitID_t : Identification
AlgorithmID_t : Identification
ServiceID_t : Identification
interface_configurations : ...
Unit
- my_ID: UnitID_t {id}
+ load ()
+ customInitialization (): HAMSTER_RETURN
+ customTermination ()
+ parseMessage (msg : IdentifiedMessage)
UnitID_t : Identification
CommData
- peer_list: Peer [*]
- module_list: UnitID_t [*] {Entity}
{platform_access_comm_data = true}
UnitID_t : Identification
platform_executes_loop : Boolean = false
platform_has_outgoing_queue : Boolean = false
platform_access_comm_data : Boolean = false
Platform
- incoming_queue: MessageQueue
- outgoing_queue: MessageQueue {has_outgoing_queue = true}
+ initialize (): HAMSTER_RETURN
+ terminate (): HAMSTER_RETURN
+ parseMessage (msg : IdentifiedMessage)
+ sendMessage (msg : IdentifiedMessage): HAMSTER_RETURN
ExecutionLoop
- my_execution_thread: std::unique_ptr<std::thread> = std::nullptr
+ executeInLoop (): Boolean
«bind»
<UnitID_t, false, false, true>
UnitID_t : Identification
MessageBroker
«bind»
<UnitID_t, false, true, true>
UnitID_t : Identification
AbstractInterface
1..*
UnitID_t : Identification
AlgorithmID_t : Identification
Type : UNIT_TYPE
«bind»
<UnitID_t, false, true, true>
Cloud-SPHERE
0..1
UnitID_t : Identification
Type : UNIT_TYPE
ServiceID_t : Identification
«bind»
<UnitID_t, false, true, true>
SEMU
0..1
UnitID_t : Identification
Type : UNIT_TYPE
«bind»
<UnitID_t, false, true, true>
<UnitID_t, true, true, true>
NP
0..1
UnitID_t : Identification
Type : UNIT_TYPE
«bind»
<UnitID_t, false, false, false>
NCI
0..1

Source: Rodrigues and Branco (2022).

5.1.1 Model and Implementation

Figure 11 presents the HAMSTER UNIT model with all its Platforms. The `Unit` class is the representation of a HAMSTER UNIT and has six different templates that allow the system engineer to configure how the Unit will operate. The `Type` parameter determines the Unit type as described in this Section, and its value is one of the included in `UNIT_TYPE` enumeration. The `scenario` parameter determines the operation scenario the Unit is inserted in. The parameter value comes from `SCENARIO` enumeration and corresponds to the HAMSTER operation scenarios described in Section 4.7. Parameters `UnitID_t`, `AlgorithmID_t` and `ServiceID_t` are `Identification` types used to identify Units, Algorithms and Services in the system, respectively. Finally, `interface_configurations` is a variadic template containing the classes used for communication interfaces in the Implementation Layer (Figure 3); the Unit may receive any number of these structs, each of them corresponding to a different communication interface.

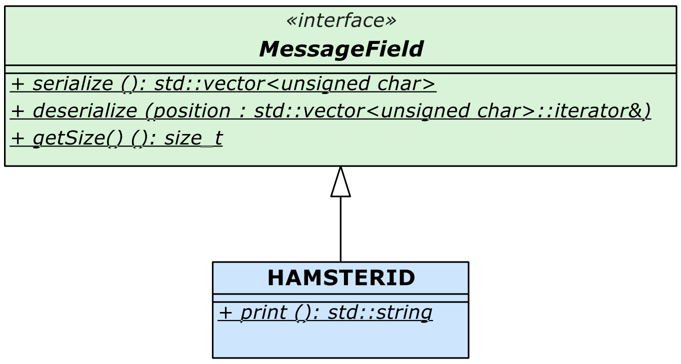
Any device that implements HAMSTER must inherit from the `Unit` class, composed by different Platforms. A HAMSTER PLATFORM represented by `Platform` class is a software unit that can send and receive messages and has its own thread of execution. There are three Boolean template parameters that configures the behavior of a Platform: `platform_executes_-`

`loop` configures if the Platform performs any looping action besides waiting for new messages; `platform_has_outgoing_queue` configures if the platform sends messages (all of them receive); finally, `platform_access_comm_data` configures the access of that specific Platform to the `CommData` instance in that Unit, which encapsulates all peer-related data and configuration.

In the diagram, all possible platforms of a Unit are illustrated: the `MessageBroker` distributes the messages in and out of a Unit (see Section 5.3 for message distribution inside HAMSTER); the `AbstracInterface` (Section 5.2) is an abstraction for all communication interfaces inside the Unit; classes `Cloud-SPHERE` (Section 5.4), `SEMU` (Section 5.5), `NP` (Section 5.6), and `NodeCriticalIndex` represent the different HAMSTER PLATFORMS presented in the previous Chapter. The presence or not of a Platform in a Unit follows the scheme described by Table 5.

Unique identification is one of the bases the HAMSTER prototype is based on, and IDs are frequently passed between Units in communication, meaning they must be serialized and deserialized from message buffers. Because of that, the classes used for identification in the system must be a `Identification` type that can be (a) *hashable*; (b) *ordable*; (c) *serializable* and *deserializable*. For that, the used class must either be an integral type (except bool type), a `std::string` or a user class that follows the interface provided by class `HAMSTERID` (Figure 12).

Figure 12 – `HAMSTERID` interface.


«interface»
MessageField
+ serialize (): std::vector<unsigned char>
+ deserialize (position : std::vector<unsigned char>::iterator&)
+ getSize() (): size_t
HAMSTERID
+ print (): std::string

Source: Rodrigues and Branco (2022).

5.1.2 Operation

In a typical application, the system engineer creates a HAMSTER UNIT by inheriting from `Unit` class — this will provide access to all configuration available (network addresses, authentication and cryptographic data, etc). After all necessary configurations are done, the Unit is initialized by invoking the `load()` method. The Unit initialization will (a) initialize all platforms; (b) invoke the Application Layer initialization method (`customInitialization()`), so application-specific initial steps can be performed; (c) if an Entity, wait for all of its modules to be connected and authenticated; (d) if in a Centralized scenario, connect to its Central Unit and (e) finally, initialize the registered services.

During the Unit operation, the Application Layer can exchange messages by using the Application Message (Section 5.3.1) interface (methods `sendMessage()` and `parseMessage()`). HAMSTER Core messages are processed automatically by the platforms.

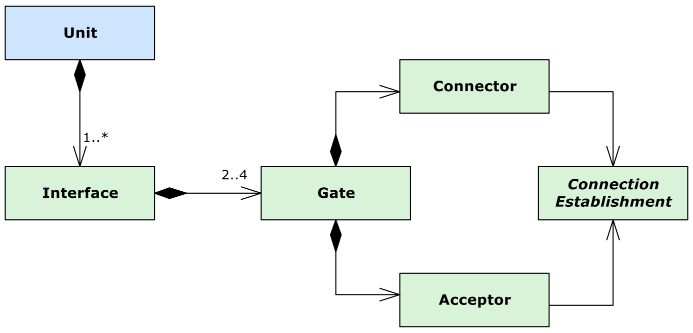
The Unit termination process follows the initialization steps in inverse order:(a) the registered services are terminated; (b) if a Unit, all objects are notified to terminate; (c) after modules are disconnected, disconnect from central Unit; (d) invoke Application Layer termination (`customTermination()`); finally, terminate all internal platforms.

5.2 Communication between Units

Communication is a key component in HAMSTER, and a great care was taken to make its abstractions highly configurable so many technologies can be used in the architecture. This Section presents and explains the communication Interfaces design and the connection between Units.

Figure 13 brings the high-level communication model in HAMSTER. Each HAMSTER UNIT has at least one communication INTERFACE, which can be realized in different technologies and deploy different protocols depending on the available options.

Figure 13 – Communication model in HAMSTER.

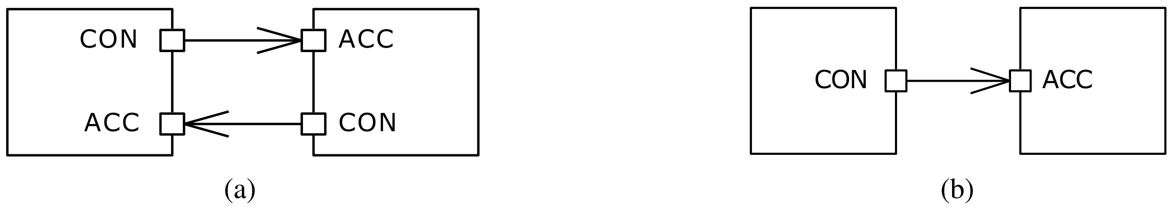


Source: Rodrigues and Branco (2022).

Every Interface is composed by GATES, a means of communication entrance and/or exit. Obligatorily, every interface has at least two gates for HAMSTER communication: one for control message exchanges (HAMSTER GATE) and another for HAMSTER broadcast queries (BROADCAST GATE). There are also two extra types of gates, that can be included and configured at compile time: (1) an APPLICATION GATE for application message exchange and (2) SERVICE GATE for service data exchange.

A gate is envisioned to have two modes of operation, illustrated in Figure 14. The operation mode is defined while configuring the Interface the gate belongs to. A FULL-DUPLEX Gate (Figure 14a) has two communication paths, with each communicating entity having a point to accept connections and another to connect to other peers. A HALF-DUPLEX Gate (Figure 14b) has a single communication path — the entity either starts the communication or accept it. The Gate operation mode has nothing to do with communication synchronization — the use of synchronous or asynchronous communication is defined by the designer based on the communication implementation.

Figure 14 – Gate operation modes in HAMSTER Interfaces.



Source: Rodrigues and Branco (2022).

Based on its operation mode, the Gate may have two types of *components*: an ACCEPTOR (an entry point that accept connections from other Units) and/or a CONNECTOR (an exit point that connects to other Units). For each Gate component, the designer can configure if (a) it needs a specific address and (b) if a specific execution thread is needed. For instance, in a socket communication, the Acceptor (the server) would need both an address (with IP and port, so clients can connect to it) and a specific execution thread (so it can continuously listen its port). The socket Connector (the client), however, does not need any of them.

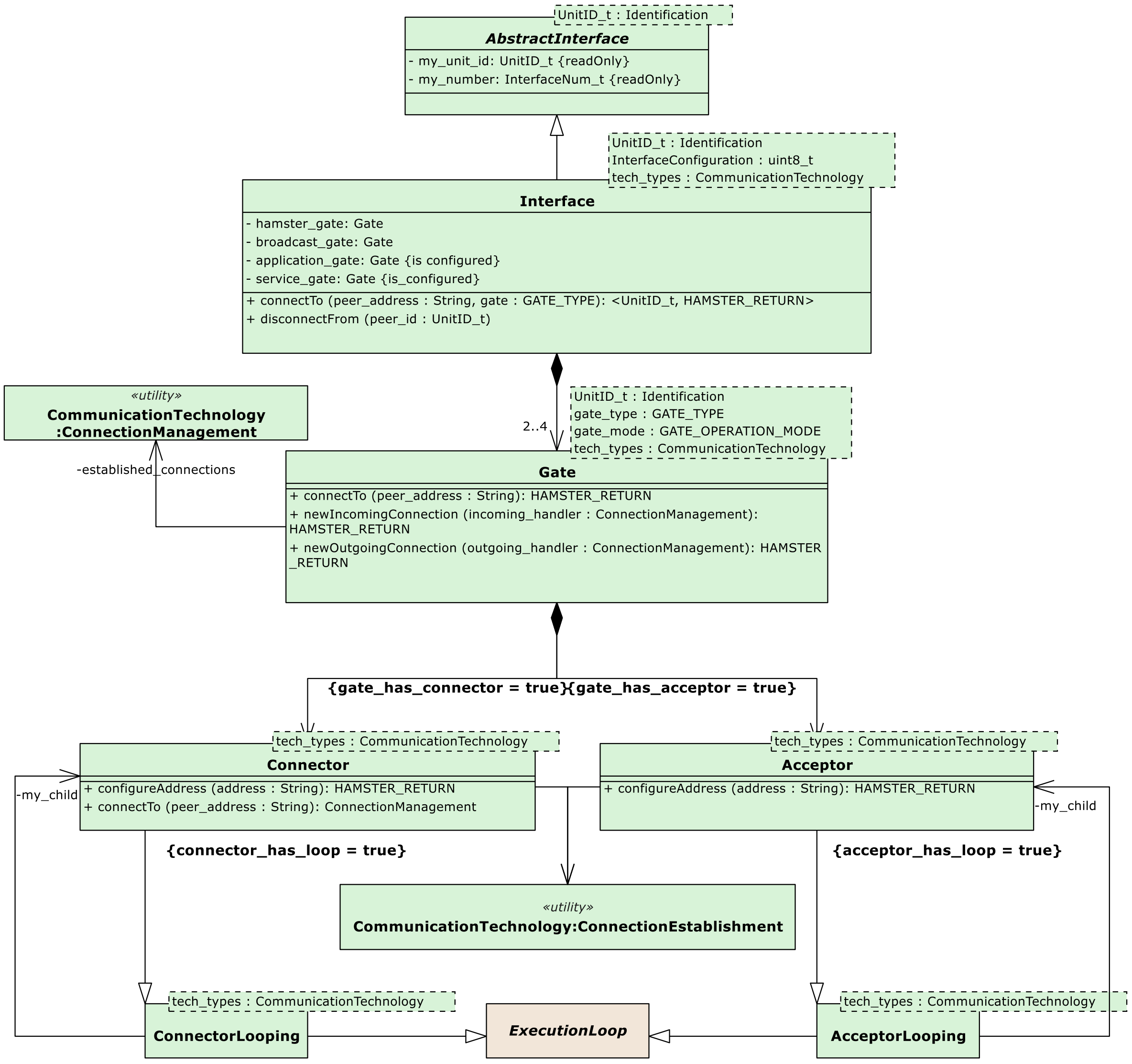
The Gate components are associated with connection abstractions and do not deploy any specific communication protocol. To make the bridge between the HAMSTER Core Layer and any specific implementation, the HAMSTER Abstraction Layer defines a *Communication Technology* abstraction, which links the architecture to any technology implementation provided by the system designer.

Network addresses are passed along HAMSTER components as a string, allowing the designer to pass any kind of information from the Application to the Implementation layer. A CONNECTION ESTABLISHMENT abstraction provides the methods invoked by Connectors and Acceptors for connecting and receiving connections to and from other Units, respectively. Once a connection is established, a CONNECTION MANAGEMENT object deals with managing message exchanging and closing the connection if necessary. These objects are associated with a PEER so it can be retrieved when communication occurs.

5.2.1 Model and Implementation

Figure 15 presents the communication interface model in HAMSTER. At the top center is the `AbstractInterface`, which is a communication-technology-independent interface abstraction that stores only its number and Unit ID. The `Interface` itself has two additional template

Figure 15 – Communication Interface model in HAMSTER.


UnitID_t : Identification
AbstractInterface
- my_unit_id: UnitID_t {readOnly}
- my_number: InterfaceNum_t {readOnly}
UnitID_t : Identification
InterfaceConfiguration : uint8_t
tech_types : CommunicationTechnology
Interface
- hamster_gate: Gate
- broadcast_gate: Gate
- application_gate: Gate {is configured}
- service_gate: Gate {is_configured}
+ connectTo (peer_address : String, gate : GATE_TYPE): <UnitID_t, HAMSTER_RETURN>
+ disconnectFrom (peer_id : UnitID_t)
«utility»
CommunicationTechnology :ConnectionManagement
-established_connections
2..4
UnitID_t : Identification
gate_type : GATE_TYPE
gate_mode : GATE_OPERATION_MODE
tech_types : CommunicationTechnology
Gate
+ connectTo (peer_address : String): HAMSTER_RETURN
+ newIncomingConnection (incoming_handler : ConnectionManagement): HAMSTER_RETURN
+ newOutgoingConnection (outgoing_handler : ConnectionManagement): HAMSTER _RETURN
{gate_has_connector = true}
{gate_has_acceptor = true}
tech_types : CommunicationTechnology
Connector
+ configureAddress (address : String): HAMSTER_RETURN
+ connectTo (peer_address : String): ConnectionManagement
-my_child
tech_types : CommunicationTechnology
Acceptor
+ configureAddress (address : String): HAMSTER_RETURN
-my_child
{connector_has_loop = true}
{acceptor_has_loop = true}
«utility»
CommunicationTechnology:ConnectionEstablishment
tech_types : CommunicationTechnology
ConnectorLooping
ExecutionLoop
tech_types : CommunicationTechnology
AcceptorLooping

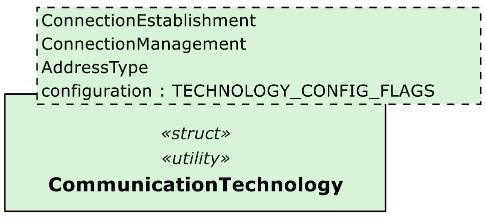
Source: Rodrigues and Branco (2022).

parameters. `InterfaceConfiguration` is a 8-bit integer used to configure the Interface; the `TechnologyImplementation` template parameter is a struct that aggregates all communication technology classes in the Implementation Layer.

All possible Gates an Interface can have are implemented by **`Gate`** class, which is of a certain **`GATE_TYPE`** and has a specific **`GATE_OPERATION_MODE`** that configures if the Gate has an **`Acceptor`** and/or a **`Connector`**. As with Interfaces, Gates also have a technology-independent abstraction (**`AbstractGate`**) and holds a list of established connections that can be managed by **`ConnectionManagement`** abstraction.

Depending on the communication technology used, the Acceptor and Connector may execute a dedicated thread (encapsulated in classes **`AcceptorLooping`** and **`ConnectorLooping`**, respectively) and/or need an address to operate. The addresses are stored in their concrete implementation of `ConnectionEstablishment` abstraction.

Figure 16 – CommunicationTechnology struct.



Source: Rodrigues and Branco (2022).

Many classes such as `Interface` and `Gate` have a struct template parameter of type `CommunicationTechnology` (Figure 16). This utility struct combines abstractions used by the communication technology and its configuration (Section 5.2.2) so HAMSTER supports necessary addressing and looping for connection establishment and management. All elements are template parameters — the struct is a convenient way to pass various types without creating any object. Abstraction `AdressType` encapsulates eventual network addresses, while classes `ConnectionEstablishment` and `ConnectionManagement` encapsulates the interface for establishing and manage connections, respectively.

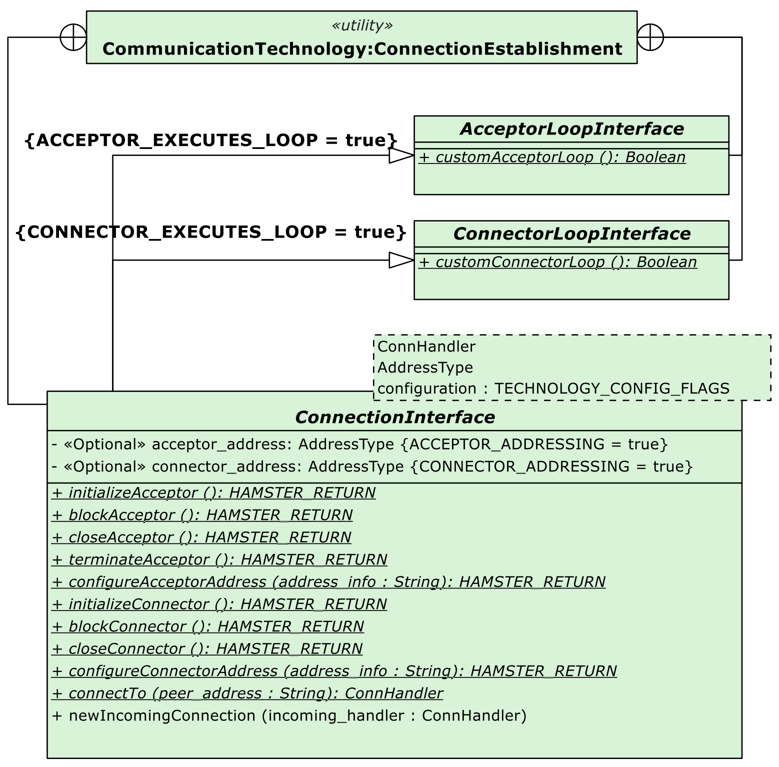
The `ConnectionEstablishment` abstraction nests the classes related to establishing new connections, as shown by Figure 17. Abstract class `ConnectionInterface` contains the methods used by the Acceptor and Connector to establish new connections. If any of them is configured to have a dedicated thread of execution, the inheritance from classes `AcceptorLoopInterface` and `ConnectorLoopInterface` should also be implemented by the communication technology classes in the Implementation Layer.

Similarly, `ConnectionManagement` (Figure 18) nests classes related to exchanging messages and closing established connections. Abstract class `ManagementInterface` encapsulates the functionality methods used by *handlers* (created by new connections) to exchange messages. If an *incoming handler* — created by an Acceptor — or an *outgoing handler* — created by a Connector —- need a dedicated thread of execution, the inheritance from classes `IncomingHandlerLoopInterface` and `OutgoingHandlerLoopInterface` should also be implemented by the communication technology classes in the Implementation Layer.

Since communication is the central point in HAMSTER, peer-related communication information has to be accessed in many different point of the architecture. Because of that, peer-related data is encapsulated by a specific class (`CommData`, Figure 19), whose reference is shared among all Platforms configured to access it (`platform_access_comm_data = true`).

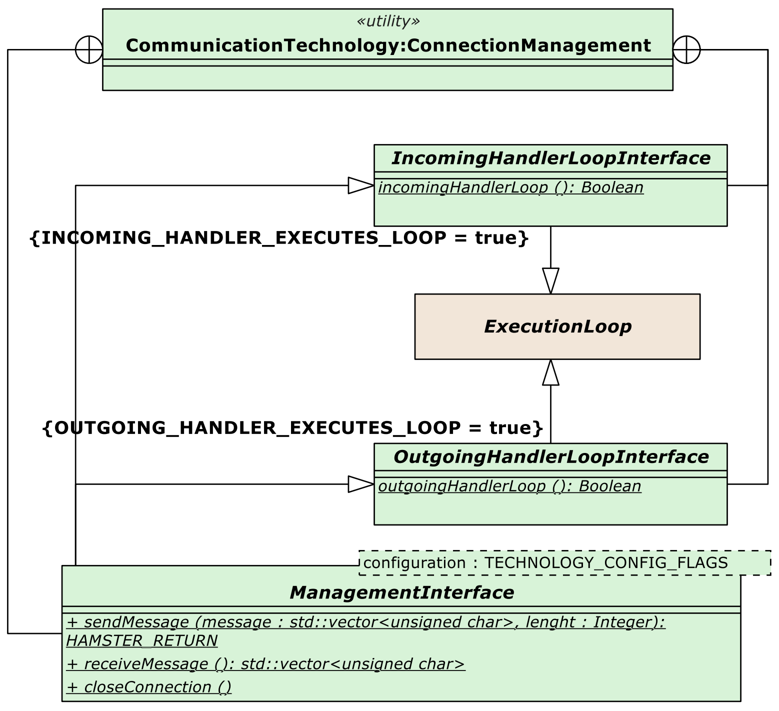
Class CommData has a list with all Peers with whom the Unit is communicating to. Class `Peer` concentrate all communication-related data from each peer: identification, addresses for

Figure 17 – `ConnectionEstablishment` model in HAMSTER.


«utility»
CommunicationTechnology:ConnectionEstablishment
{ACCEPTOR_EXECUTES_LOOP = true}
AcceptorLoopInterface
+ customAcceptorLoop (): Boolean
{CONNECTOR_EXECUTES_LOOP = true}
ConnectorLoopInterface
+ customConnectorLoop (): Boolean
ConnHandler
AddressType
configuration : TECHNOLOGY_CONFIG_FLAGS
ConnectionInterface
- «Optional» acceptor_address: AddressType {ACCEPTOR_ADDRESSING = true}
- «Optional» connector_address: AddressType {CONNECTOR_ADDRESSING = true}
+ initializeAcceptor (): HAMSTER_RETURN
+ blockAcceptor (): HAMSTER_RETURN
+ closeAcceptor (): HAMSTER_RETURN
+ terminateAcceptor (): HAMSTER_RETURN
+ configureAcceptorAddress (address_info : String): HAMSTER_RETURN
+ initializeConnector (): HAMSTER_RETURN
+ blockConnector (): HAMSTER_RETURN
+ closeConnector (): HAMSTER_RETURN
+ configureConnectorAddress (address_info : String): HAMSTER_RETURN
+ connectTo (peer_address : String): ConnHandler
+ newIncomingConnection (incoming_handler : ConnHandler)

Source: Rodrigues and Branco (2022).

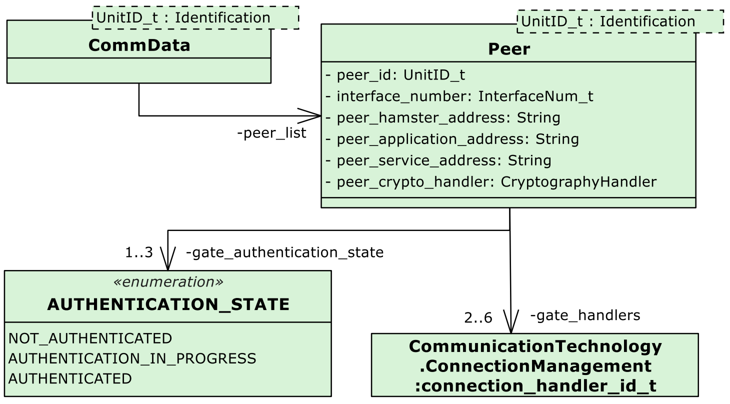
Figure 18 – `ConnectionManagement` model in HAMSTER.


«utility»
CommunicationTechnology:ConnectionManagement
IncomingHandlerLoopInterface
incomingHandlerLoop (): Boolean
{INCOMING_HANDLER_EXECUTES_LOOP = true}
ExecutionLoop
{OUTGOING_HANDLER_EXECUTES_LOOP = true}
OutgoingHandlerLoopInterface
outgoingHandlerLoop (): Boolean
configuration : TECHNOLOGY_CONFIG_FLAGS
ManagementInterface
+ sendMessage (message : std::vector<unsigned char>, lenght : Integer): HAMSTER_RETURN
+ receiveMessage (): std::vector<unsigned char>
+ closeConnection ()

Source: Rodrigues and Branco (2022).

each Gate type, the Interface used to communicate with it, the `AUTHENTICATION_STATE` and Communication Handler (`connection_handler_id_t`) of every connected Gate as well as the cryptographic handler for the specific connection. At this point, only one handler is available, i.e., the cryptography algorithm is the same for all gates, but it is envisioned to add support for different cryptography algorithms for every gate. A new Peer can only be inserted (1) by the Unit itself in known formations and (2) by the HAMSTER Gate when a new connection is established (Section 5.2.4).

Figure 19 – Peer management in HAMSTER.


UnitID_t : Identification
CommData
UnitID_t : Identification
Peer
- peer_id: UnitID_t
- interface_number: InterfaceNum_t
- peer_hamster_address: String
- peer_application_address: String
- peer_service_address: String
- peer_crypto_handler: CryptographyHandler
-peer_list
1..3
-gate_authentication_state
«enumeration»
AUTHENTICATION_STATE
NOT_AUTHENTICATED
AUTHENTICATION_IN_PROGRESS
AUTHENTICATED
2..6
-gate_handlers
CommunicationTechnology
.ConnectionManagement
:connection_handler_id_t

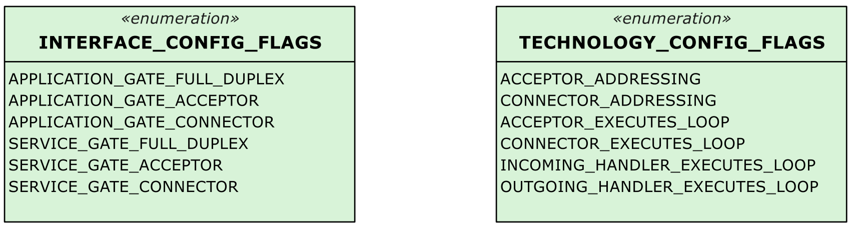
Source: Rodrigues and Branco (2022).

5.2.2 *Configuration*

HAMSTER communication is highly configurable and allows the system engineer to deploy the needs of their application in different technologies. There are two configurations to be done regarding communication: the interface and the technology. INTERFACE CONFIGURATION is done through `INTERFACE_CONFIG_FLAGS` enumeration (Figure 20a) and determines if the interface will have Application and Service Gates and the Operation Mode of each one as described in Figure 14 (HAMSTER and Broadcast Gates are not configurable). TECHNOLOGY CONFIGURATION is done through `TECHNOLOGY_CONFIG_FLAGS` enumeration (Figure 20b) and tells HAMSTER about the implementation needs regarding network addresses and dedicated threads for the Acceptor, Connector and Connection Handlers.

As an example, consider an Interface that has a Full-Duplex Service Gate and a Half-Duplex Application Gate that accepts connections from other Units; the technology being used are connection-oriented socket-based (i.e., TCP sockets). Considering that for this technology (a) the server (Acceptor) needs a pre-established network address, but not the client (Connector);(b) that the server needs a dedicated thread to listen to its port and (c) a server established connection (Incoming Handler) needs a dedicated thread to wait for new messages, Interface and Technology Configuration Flags would be as described in Source code 1 (enumeration names were suppressed for readability).

Figure 20 – Configuration flags for HAMSTER communication.



(a) (b)

Source: Rodrigues and Branco (2022).

Source code 1 – Communication configuration example.

```
interface_configuration =
    SERVICE_GATE_FULL_DUPLEX |
    APPLICATION_GATE_ACCEPTOR;

technology_configuration =
    ACCEPTOR_ADDRESSING | ACCEPTOR_EXECUTES_LOOP |
    INCOMING_HANDLER_EXECUTES_LOOP;
```

5.2.3 Operation

In an application, the system engineer must provide in the Implementation Layer classes to realize the three communication abstractions present in HAMSTER: a class to encapsulate network addresses, a class that makes connection and create objects of a third class that will manage the connection. These classes must inherit from `AddressInfo`, `ConnectionInterface` and `ManagementInterface` respectively.

The communication is initialized in UNIT's `load()` method together with all other Platforms. In the HAMSTER CORE, the Unit initialize each Interface, which in turn initializes all existing Gates, which in turn initializes their Acceptor and/or Connector. The classes provided in the Implementation Layer are also initialized as the last chain link.

The Interface operation is event-driven based on messages sent and received. When the Unit needs to transmit a message, the Interface routes it to the appropriate Gate and the message is sent through a Connection Handler. If the Unit receives a message in one Connection Handler, a chain of notification goes up to the HAMSTER Core and properly routed to any Platform or the Application Layer (for more information on messaging, see Section 5.3).

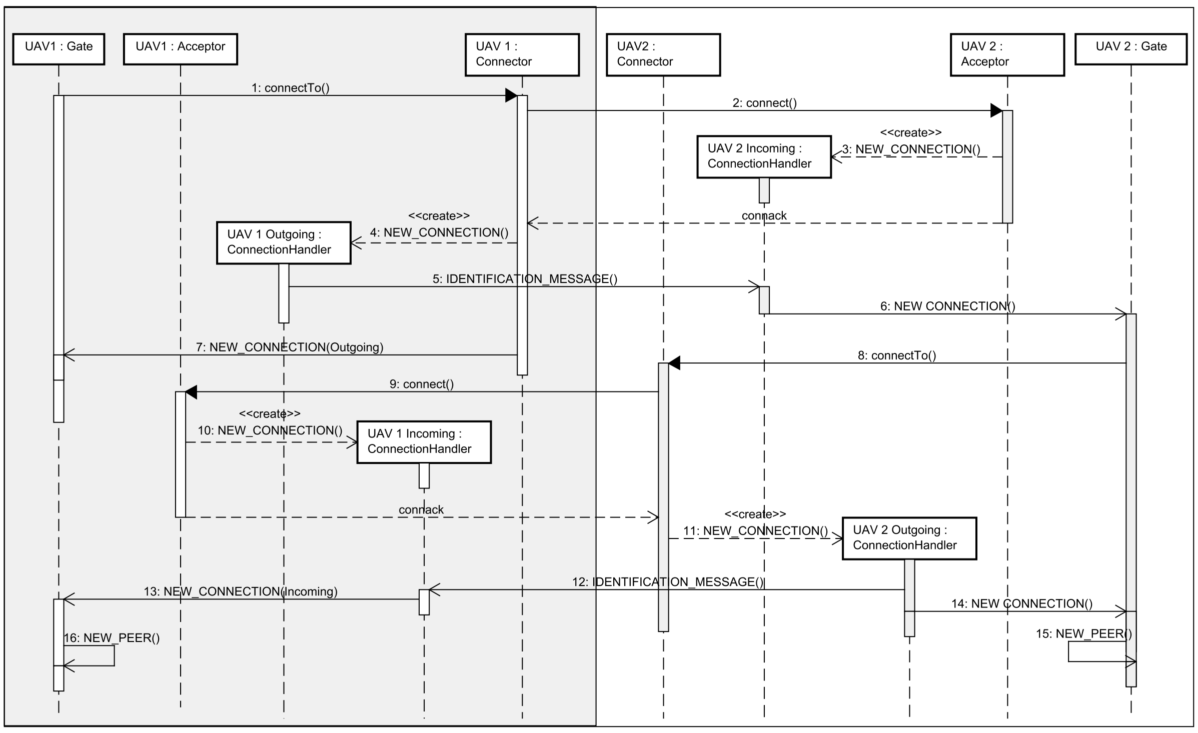
When an Interface is terminated, all established connections in all gates are closed, and the `terminate()` methods from classes in the Implementation Layer is invoked together with of the classes in the HAMSTER Core Layer.

5.2.4 The Connection between Units

HAMSTER Core Layer provides two different communication services to the Application Layer: (1) application message exchange and (2) service exchange, both of them implemented in dedicated Gates between the Units.

Before establishing a connection in either Application Gate or Service Gate, however, HAMSTER needs to properly identify and authenticate the Units through the HAMSTER Gate. In *known formation scenarios* (Section 4.7), the connection establishment and authentication in HAMSTER GATE is done automatically while the Unit is being loaded, whereas in *unknown formation scenarios* this connection is made on demand when the application requests a message or service exchange. Only after the HAMSTER GATE is connected and authenticated Application and Service Gates may connect and authenticate.

Figure 21 – Connection Establishment in HAMSTER GATE.



Source: Rodrigues and Branco (2022).

The sequence diagram in Figure 21 details how a full-duplex connection is established between the HAMSTER GATES of two different Units. In this figure, UAV1 is trying to connect to UAV2.

First, the UAV 1:GATE (a HAMSTER Gate) starts the process by requesting a connection to its Connector (1). UAV 1:CONNECTOR connects to UAV 2:ACCEPTOR through the `ConnectionInterface` (Section 5.2.3) implementation of the specific communication technology used for this Interface. When receiving the connection request, UAV 2:ACCEPTOR creates a new *Incoming Connection Handler* (3) from `ManagementInterface` system implementation (Section 5.2.3) which will handle any new messages that arrive to the new connection.

Upon receiving the connection acknowledgment, the Connector creates an *Outgoing Connection Handler* (4) and notifies the Gate on the new outgoing connection (7), who in turn adds the handler to its connection list. The Connection Handler sends an *Identification Message* (5) to UAV 2 INCOMING:CONNECTIONHANDLER with the necessary data for the connection to be completed. When this Identification Message arrives, the Incoming Connection Handler notifies UAV 2:GATE of a new incoming connection (6). Since this is a Full-Duplex Gate, UAV 2 now has to connect to UAV 1. This triggers the same process, but now with UAV 2 being the connection "starter" (8-14). When the connection is complete (13 for UAV 1 and 14 for UAV2), the GATE for both units register the new connection Peer (15 and 16).

When this process is finished, both Units can use the HAMSTER GATE to authenticate themselves and exchange messages as explained in the next Section.

5.3 Message Exchange in HAMSTER

All data exchange between HAMSTER Units and Platforms is done by HAMSTER MESSAGES (Section 5.3.1), which has a fixed structure and a unique code that identify the message and the Platform it is destined to. Any Platform can send and receive messages; HAMSTER also enables the exchange of Application Layer messages, which are forwarded encapsulated in a specific HAMSTER Message.

The MESSAGE BROKER is responsible for distributing messages of a HAMSTER Unit. It can redirect either (1) Application Layer messages to another Unit; (2) Platform messages destined to another Platform in the same Unit; (3) Platform messages destined to a Platform in another Unit and (4) Application Layer and/or Platform messages from other Units to the Application Layer or correct Platform, respectively.

The Message Broker does not analyze the message destination or code to distribute it, but its *direction*. An INTERNAL message is destined to the Application Layer or another Platform of its own Unit and it is redirected appropriately; and EXTERNAL message is meant for another Unit and is redirected to the appropriate Interface to be sent through the network.

The messages come in and out from any HAMSTER Platform (including the Message Broker and Communication Interfaces) thorough *Queues*. Any time a Platform sends a Message, it goes to the Message Broker *Incoming Queue*; the broker then parses the message and puts it in the Incoming Queue of a Platform (Internal Messages) or of an Interface (External Messages).

5.3.1 HAMSTER Messages

A HAMSTER MESSAGE has a static structure composed by a header and a payload, as shown by Figure 22. The header includes the *Message Code* and the *Payload Size*. The Message Code is an integer that aggregates two information:(1) which message is being transmitted and (2) to which platform(s)/application the message is destined to; the Payload Size indicate how many bytes are in the message payload.

Figure 22 – HAMSTER Message Structure.

Header		Payload
Message Code & Direction	Payload Size	...

Source: Rodrigues and Branco (2022).

The Payload is composed by MESSAGE FIELDS, which can be anything (i.e., the value of a specific variable or all data members of a class) that is shared by a message. All fields are serialized into a buffer, transmitted through the network, and deserialized on its destiny.

Data portability is a crucial feature when designing communicating systems with heterogeneous platforms. Some problems may arise such as memory alignment, byte endianness, size of native types, and floating-point arithmetic (HOOK, 2005). Because of that, HAMSTER has some guidelines for dealing with message fields:

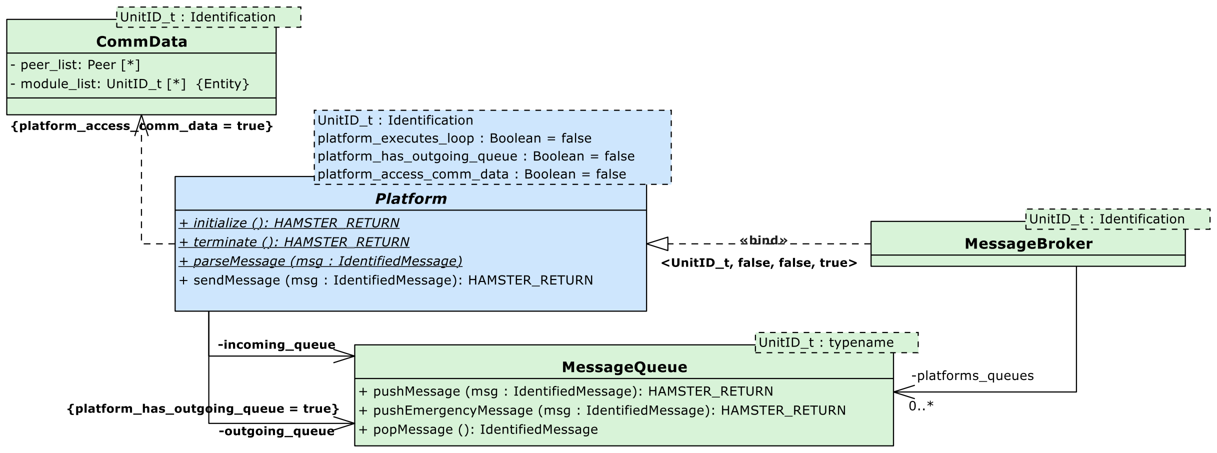
1. A class must not be serialized as a hole, but having each of its members serialized individually;
2. Data is to be serialized in little-endian format;
3. All integer types should be of fixed sizes provided by `stdint` library (such as `int8_t`, `uint8_t`, `int16_t` and so on);
4. Floating-point variables are always sent formatted as a `std::string` in order to prevent standard conflicts.

Evidently, these guidelines can be disregarded and the message payload built as considered best by the application. In this situation, the application shall be responsible for data portability.

5.3.2 *Model and Implementation*

Figure 23 brings the Message Broker modeling in HAMSTER. As any Platform, the **`MessageBroker`** has an `incoming_queue` in which messages arrive to be redirected to the proper **`Platform`**, which can be a HAMSTER Platform (*Cloud*–SPHERE, SEMU, etc.) or an Interface, depending on the Message direction.

Figure 23 – Communication Broker model in HAMSTER.


UnitID_t : Identification
CommData
- peer_list: Peer [*]
- module_list: UnitID_t [*] {Entity}
{platform_access_comm_data = true}
UnitID_t : Identification
platform_executes_loop : Boolean = false
platform_has_outgoing_queue : Boolean = false
platform_access_comm_data : Boolean = false
Platform
+ initialize (): HAMSTER_RETURN
+ terminate (): HAMSTER_RETURN
+ parseMessage (msg : IdentifiedMessage)
+ sendMessage (msg : IdentifiedMessage): HAMSTER_RETURN
«bind»
<UnitID_t, false, false, true>
UnitID_t : Identification
MessageBroker
-incoming_queue
UnitID_t : typename
MessageQueue
+ pushMessage (msg : IdentifiedMessage): HAMSTER_RETURN
+ pushEmergencyMessage (msg : IdentifiedMessage): HAMSTER_RETURN
+ popMessage (): IdentifiedMessage
{platform_has_outgoing_queue = true}
-outgoing_queue
-platforms_queues
0..*

Source: Rodrigues and Branco (2022).

The Broker incoming `MessageQueue` reference is shared among all other HAMSTER Platforms to be used as their outgoing queue. In practice, when any Platform invokes the method `sendMessage()`, it is actually pushing the message into the Broker's incoming queue, which has concurrent access protection.

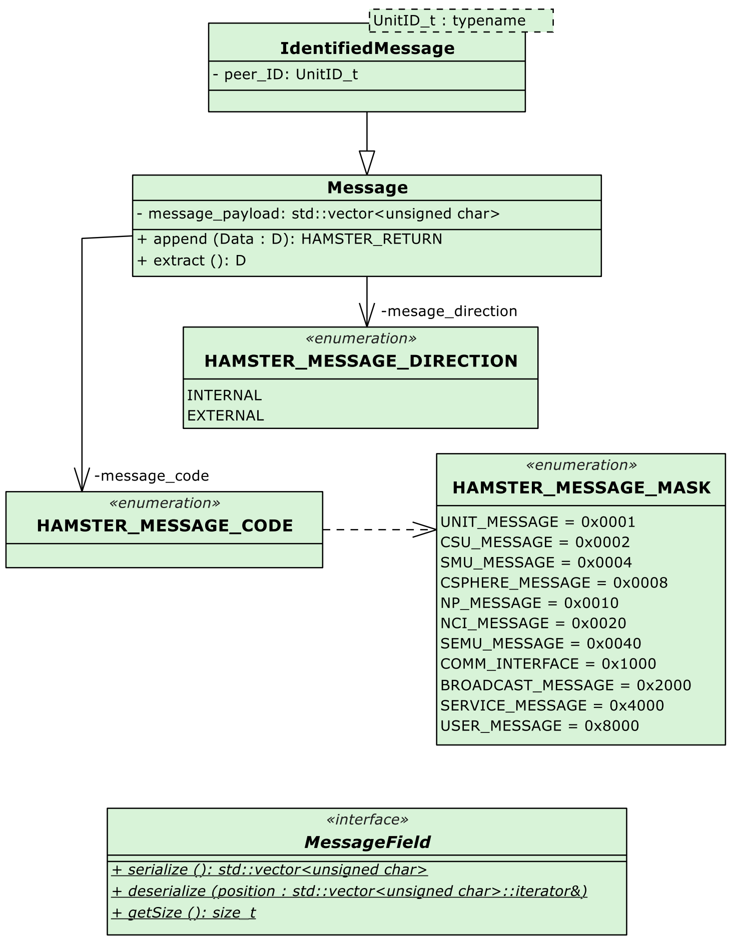
Differently from other Platforms, the Broker does not have an outgoing queue, but rather a list of Platform's incoming message queues, each of them identified by the corresponding Platform Message Mask (Figure 24). When the Broker parses a message it received in the incoming queue, it is redirected to the appropriate Platform (if an INTERNAL message) or to the Peer's corresponding Interface (if an EXTERNAL message) — the Broker has access to Peer information in `CommData` class through `Platform` inheritance.

Figure 24 brings more details on HAMSTER Messages. There are two different classes used to encapsulate messages in HAMSTER: class `Message` is used during Unit connection (Section 5.2.4), since the Units do not know each other's ID. After the connection is established, HAMSTER uses the `IdentifiedMessage`, which contains the Peer from/to which the massage came/was sent.

Besides following the fixed structure already discussed in this Section, the Message has three important attributes: the `message_code`, which has a unique numeric value for each message; the `message_payload`, which contains message data, including its header; and the `message_direction`, which determines to which Platform the Message will be redirected by the Broker.

Class `HAMSTER_MESSAGE_CODE` provide the unique codes that identify each message. It is a 32-bit number made by two parts: the first 16 bits incorporates the `HAMSTER_MESSAGE_MASK` of the Message destination, and the second 16 bits are enumerated sequentially. This provides unique identification and incorporates the Message destination at the same time.

Figure 24 – Message model in HAMSTER.



Source: Rodrigues and Branco (2022).

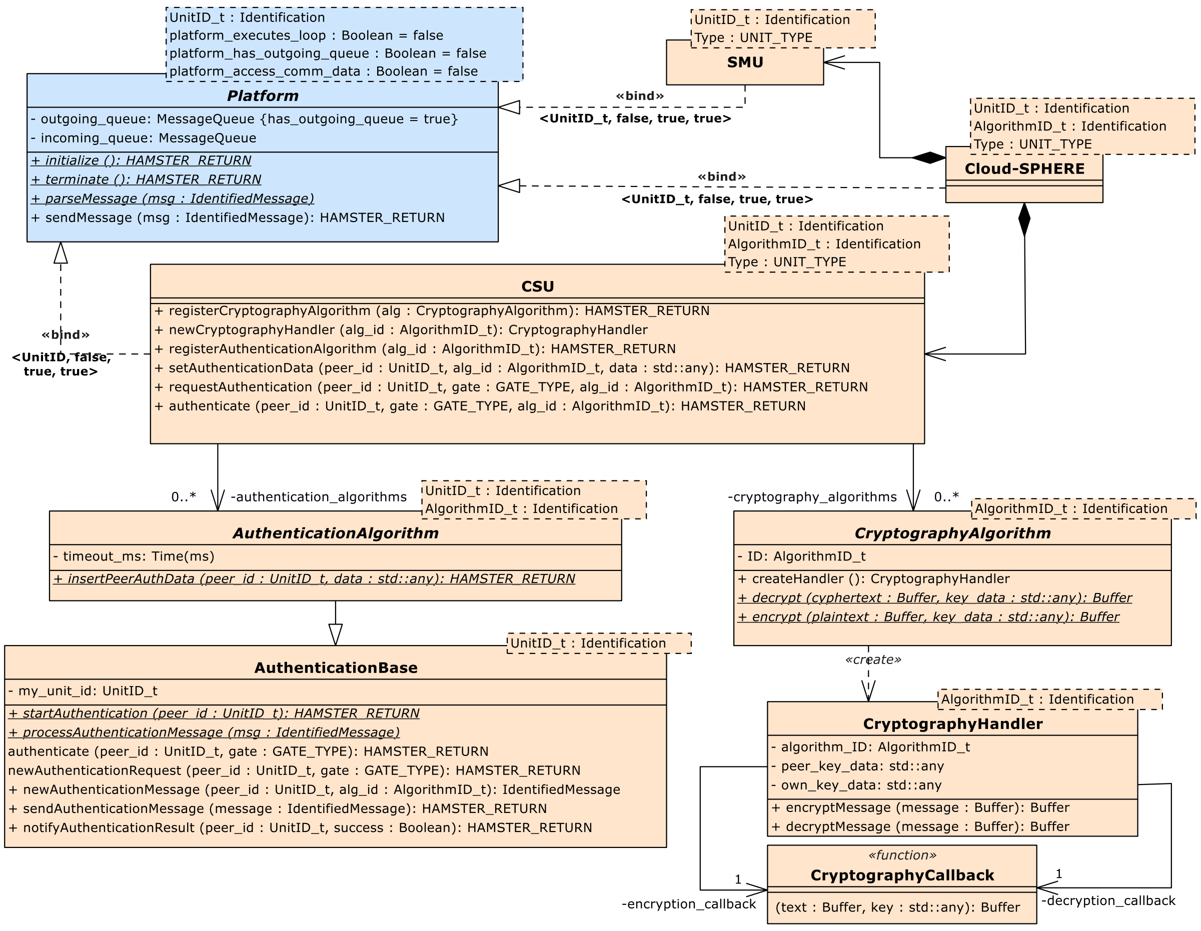
The **Message** class has built-in elements to add and extract *fields* to and from its payload — method append() is invoked to write data into message payload, while method extract() reads data into message payload. These two messages uses serialization utilities already included in HAMSTER to serialize and deserialize data of primitive types. If the application needs to transmit an object as a Message Field, this data type D must inherit from **MessageField** so the aforementioned methods can invoke the necessary interface.

5.3.3 Operation

Message-related operation in HAMSTER is very straightforward. Every Platform has a loop that checks for new messages and can either perform specific loop actions (setting the template flag `platform_executes_loop = true`) or sleep until a new message arrives (setting the template flag `platform_executes_loop = false`).

If there is any new Message, it is popped from the Message Queued and used to invoke the parseMessage() method. The parsing implementation is particular for every Platform. For

Figure 25 – *Cloud*–SPHERE model in HAMSTER.


UnitID_t : Identification
platform_executes_loop : Boolean = false
platform_has_outgoing_queue : Boolean = false
platform_access_comm_data : Boolean = false
Platform
- outgoing_queue: MessageQueue {has_outgoing_queue = true}
- incoming_queue: MessageQueue
+ initialize (): HAMSTER_RETURN
+ terminate (): HAMSTER_RETURN
+ parseMessage (msg : IdentifiedMessage)
+ sendMessage (msg : IdentifiedMessage): HAMSTER_RETURN
UnitID_t : Identification
Type : UNIT_TYPE
SMU
«bind»
<UnitID_t, false, true, true>
UnitID_t : Identification
AlgorithmID_t : Identification
Type : UNIT_TYPE
Cloud-SPHERE
«bind»
<UnitID_t, false, true, true>
UnitID_t : Identification
AlgorithmID_t : Identification
Type : UNIT_TYPE
CSU
+ registerCryptographyAlgorithm (alg : CryptographyAlgorithm): HAMSTER_RETURN
+ newCryptographyHandler (alg_id : AlgorithmID_t): CryptographyHandler
+ registerAuthenticationAlgorithm (alg_id : AlgorithmID_t): HAMSTER_RETURN
+ setAuthenticationData (peer_id : UnitID_t, alg_id : AlgorithmID_t, data : std::any): HAMSTER_RETURN
+ requestAuthentication (peer_id : UnitID_t, gate : GATE_TYPE, alg_id : AlgorithmID_t): HAMSTER_RETURN
+ authenticate (peer_id : UnitID_t, gate : GATE_TYPE, alg_id : AlgorithmID_t): HAMSTER_RETURN
«bind»
<UnitID, false, true, true>
0..*
-authentication_algorithms
UnitID_t : Identification
AlgorithmID_t : Identification
AuthenticationAlgorithm
- timeout_ms: Time(ms)
+ insertPeerAuthData (peer_id : UnitID_t, data : std::any): HAMSTER_RETURN
UnitID_t : Identification
AuthenticationBase
- my_unit_id: UnitID_t
+ startAuthentication (peer_id : UnitID_t): HAMSTER_RETURN
+ processAuthenticationMessage (msg : IdentifiedMessage)
authenticate (peer_id : UnitID_t, gate : GATE_TYPE): HAMSTER_RETURN
newAuthenticationRequest (peer_id : UnitID_t, gate : GATE_TYPE): HAMSTER_RETURN
+ newAuthenticationMessage (peer_id : UnitID_t, alg_id : AlgorithmID_t): IdentifiedMessage
+ sendAuthenticationMessage (message : IdentifiedMessage): HAMSTER_RETURN
+ notifyAuthenticationResult (peer_id : UnitID_t, success : Boolean): HAMSTER_RETURN
-cryptography_algorithms
0..*
AlgorithmID_t : Identification
CryptographyAlgorithm
- ID: AlgorithmID_t
+ createHandler (): CryptographyHandler
+ decrypt (cyphertext : Buffer, key_data : std::any): Buffer
+ encrypt (plaintext : Buffer, key_data : std::any): Buffer
«create»
AlgorithmID_t : Identification
CryptographyHandler
- algorithm_ID: AlgorithmID_t
- peer_key_data: std::any
- own_key_data: std::any
+ encryptMessage (message : Buffer): Buffer
+ decryptMessage (message : Buffer): Buffer
«function»
CryptographyCallback
(text : Buffer, key : std::any): Buffer
1
-encryption_callback
1
-decryption_callback

Source: Rodrigues and Branco (2022).

instance, SEMU and NCI make operation based on the received messages; communication Interfaces redirect those messages to the outside world in the appropriate Gate and the Message Broker makes the message distribution in this method. In many cases, a received Message requires a response, which can be sent to the Message Broker by invoking method `sendMessage()`.

5.4 The *Cloud*–SPHERE Platform

The Security and safety Platform for HEteRogeneous systEms connected to the Cloud (*Cloud*–SPHERE) is the central security point in HAMSTER, responsible for supervising the Unit's health or safety through the Safety Management Unit (SMU) and handling security aspects such as authentication, access control and communication security through the Central Security Unit (CSU). Currently, the HAMSTER Prototype supports the use of authentication and cryptography algorithms inside the CSU. Other security measures and SMU implementations are to be added in future projects.

5.4.1 Model and Implementation

Figure 25 shows the model for the *Cloud*–SPHERE Platform. `Cloud-SPHERE` class is the `Platform` composed by two units: the Safety Management Unit (`SMU`), not yet implemented, and the Central Security Unit (`CSU`), which currently supports both cryptography and authentication operations.

A different number of authentication and cryptography algorithms can be registered in the CSU. Whichever concrete implementation is registered needs to follow the specific algorithm interfaces, i.e., classes used for those operations must inherit from `AuthenticationAlgorithm` or `CryptographyAlgorithm`. Since authentication is used only when the peers are connecting and cryptography is used in every exchanged message, authentication and cryptography data are handled differently in HAMSTER. Authentication-related data must be dealt and stored by the concrete implementation itself and can be initialized from the Application Layer through method `setAuthenticationData()`. The authentication begins when the Unit invokes method `authenticate()` in the CSU. On the other hand, cryptography operations are deployed through a `cryptographyHandler` stored with Peer information (Figure 19) — the handler has access to the algorithm's encryption and decryption methods and stores the cryptography data (e.g., the keys).

5.4.2 Operation

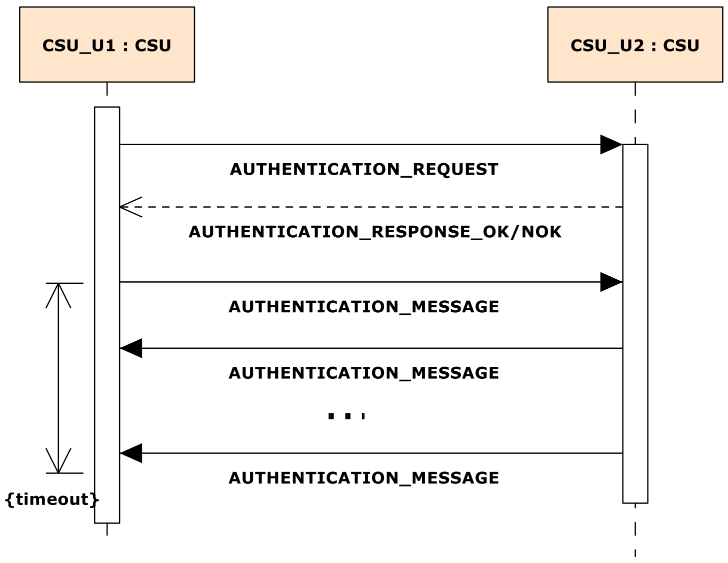
CSU handles the registration of both cryptography and authentication algorithms, which are uniquely identified and saved in a list. In the future, it is envisioned that Units are to query which algorithms are available and with that reach an agreement in which authentication or cryptography will be used for this pair of peers.

Message encryption and decryption are straightforward: when the Interface receives or has to send a Message, it retrieves the Cryptography Handler of that Peer to decrypt or encrypt the message, respectively.

Authentication is more complicated, since the strategies can vary greatly. As with other elements in HAMSTER, the authentication is based in abstractions. A HAMSTER UNIT starts the authentication by invoking method `authenticate()` in its CSU, which in turn sends an AUTHENTICATION_REQUEST message to the Peer (CSU_U2), as shown by Figure 26. The Peers responds with message AUTHENTICATION_RESPONSE_OK if the authentication can take place and AUTHENTICATION_RESPONSE_NOK otherwise. At this point, both Units are aware of the ongoing authentication process.

The implemented authentication algorithm provided by the Implementation Layer must use a specific AUTHENTICATION_MESSAGE to exchange data and authenticate Units. The Units can exchange as many messages as necessary, and the authentication process ends when (1) the Units are able to authenticate each other or (2) if the algorithm timeout specified at registration

Figure 26 – Authentication process in HAMSTER.



Source: Rodrigues and Branco (2022).

expires.

5.5 The Service Exchange Management Unit

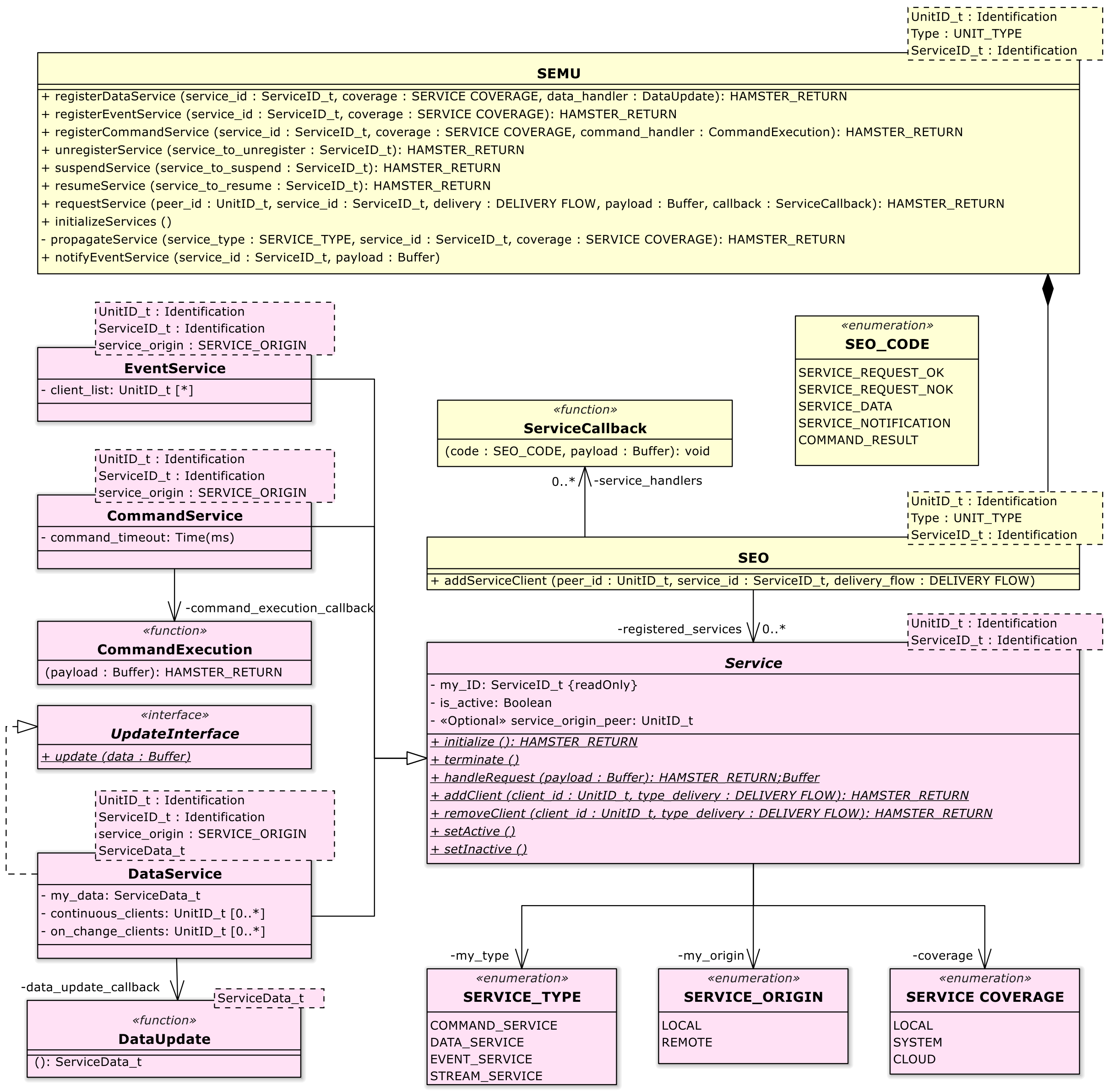
The main objective of the Service Exchange Management Unit (SEMU) is to enable HAMSTER Units to exchange different services in order to realize more complex tasks and achieve a higher integration with the IoT. The SEMU Platform and its operational module SEO (Service Exchange Operator) allow a service provider to register, propagate, manage and deliver a given service, while service clients are able to discover, request and consume those services.

5.5.1 Model and Implementation

In HAMSTER, services are abstracted to be as generic as possible and able to handle commands, events and different types of data. Figure 27 presents how the HAMSTER Prototype deals with services.

The `SEMU` Platform provides the Service-related interface with the Application Layer. Through it, it is possible to register, unregister, suspend and resume service provision, as well as performing service requests. In order to isolate service exchange from service management in HAMSTER, SEMU has an internal sandbox that exchange service-specific messages called Service Exchange Operator (SEO). Except for service propagation which is done exclusively by SEMU, all other service-related methods are also present at the SEO (the methods were not duplicated in Figure 27 for better readability).

Figure 27 – Service model in HAMSTER.


UnitID_t : Identification
Type : UNIT_TYPE
ServiceID_t : Identification
SEMU
+ registerDataService (service_id : ServiceID_t, coverage : SERVICE COVERAGE, data_handler : DataUpdate): HAMSTER_RETURN
+ registerEventService (service_id : ServiceID_t, coverage : SERVICE COVERAGE): HAMSTER_RETURN
+ registerCommandService (service_id : ServiceID_t, coverage : SERVICE COVERAGE, command_handler : CommandExecution): HAMSTER_RETURN
+ unregisterService (service_to_unregister : ServiceID_t): HAMSTER_RETURN
+ suspendService (service_to_suspend : ServiceID_t): HAMSTER_RETURN
+ resumeService (service_to_resume : ServiceID_t): HAMSTER_RETURN
+ requestService (peer_id : UnitID_t, service_id : ServiceID_t, delivery : DELIVERY FLOW, payload : Buffer, callback : ServiceCallback): HAMSTER_RETURN
+ initializeServices ()
- propagateService (service_type : SERVICE_TYPE, service_id : ServiceID_t, coverage : SERVICE COVERAGE): HAMSTER_RETURN
+ notifyEventService (service_id : ServiceID_t, payload : Buffer)
UnitID_t : Identification
ServiceID_t : Identification
service_origin : SERVICE_ORIGIN
EventService
- client_list: UnitID_t [*]
«enumeration»
SEO_CODE
SERVICE_REQUEST_OK
SERVICE_REQUEST_NOK
SERVICE_DATA
SERVICE_NOTIFICATION
COMMAND_RESULT
«function»
ServiceCallback
(code : SEO_CODE, payload : Buffer): void
0..*
-service_handlers
UnitID_t : Identification
ServiceID_t : Identification
service_origin : SERVICE_ORIGIN
CommandService
- command_timeout: Time(ms)
UnitID_t : Identification
Type : UNIT_TYPE
ServiceID_t : Identification
SEO
+ addServiceClient (peer_id : UnitID_t, service_id : ServiceID_t, delivery_flow : DELIVERY FLOW)
-command_execution_callback
«function»
CommandExecution
(payload : Buffer): HAMSTER_RETURN
-registered_services
0..*
UnitID_t : Identification
ServiceID_t : Identification
Service
- my_ID: ServiceID_t {readOnly}
- is_active: Boolean
- «Optional» service_origin_peer: UnitID_t
+ initialize (): HAMSTER_RETURN
+ terminate ()
+ handleRequest (payload : Buffer): HAMSTER_RETURN;Buffer
+ addClient (client_id : UnitID_t, type_delivery : DELIVERY FLOW): HAMSTER_RETURN
+ removeClient (client_id : UnitID_t, type_delivery : DELIVERY FLOW): HAMSTER_RETURN
+ setActive ()
+ setInactive ()
«interface»
UpdateInterface
+ update (data : Buffer)
UnitID_t : Identification
ServiceID_t : Identification
service_origin : SERVICE_ORIGIN
ServiceData_t
DataService
- my_data: ServiceData_t
- continuous_clients: UnitID_t [0..*]
- on_change_clients: UnitID_t [0..*]
-data_update_callback
ServiceData_t
«function»
DataUpdate
(): ServiceData_t
-my_type
«enumeration»
SERVICE_TYPE
COMMAND_SERVICE
DATA_SERVICE
EVENT_SERVICE
STREAM_SERVICE
-my_origin
«enumeration»
SERVICE_ORIGIN
LOCAL
REMOTE
-coverage
«enumeration»
SERVICE COVERAGE
LOCAL
SYSTEM
CLOUD

Source: Rodrigues and Branco (2022).

In order to provide its functionality, the `SEO` stores the list of all `registered_services` of a *service provider* and manages their requests and client list. Also, `SEO` also manages all `service_handlers` created from service requests made by *service clients* (more details on service registration and requests are presented in the next Section). One thing to note here is that the HAMSTER Unit assumes the role of both provider and client, being able to provide and consume services at the same time.

Every `Service` has a unique ID which follows the same rules as Unit identification: to be an integral type (except bool type), a `std::string` or a user class that follows the interface provided by class `HAMSTERID` (Figure 12). Besides the already mentioned `SERVICE_TYPE` and

`SERVICE_COVERAGE`, another important characteristic of a service is the `SERVICE_ORIGIN`: a `LOCAL` service is delivered by the Unit who registered it; a `REMOTE` service means the service is delivered by a Unit propagating the service.

Currently, the prototype supports Command, Data and Event services. Each has different characteristics and unique registration processes. Regardless of the type, service exchange is done through *service messages* between the SEO Platforms of different Units. Services messages are sent through the Service Gate, isolating them from HAMSTER control messages (sent through the HAMSTER Gate) and Application messages (sent through the Application Gate). This improves security and mitigate a possible network bottleneck, improving availability.

The **`EventService`** is the simplest service to register (it only needs its ID and coverage). This service function as a notification for its `client_list` formed from client requests. The Unit is able to notify the clients of Event Services through method `notifyEventService()`, which supports an optional payload to provide extra information to the clients.

A **`CommandService`** is used for requesting a command execution from a service provider. Because of that, a **`CommandExecution`** callback method (of type `HAMSTER_RETURN(Buffer)`, it is provided by the Application Layer to execute the service) must be provided in service registration besides its ID and coverage. This type of service does not have a client list — currently, a request for this type of service makes the SEO to start the callback method asynchronously and wait until the specified timeout or the command completion to send a response.

Finally, the **`DataService`** is the most complex so far. Besides ID and coverage, its registration also demands a **`DataUpdate`** callback method (of type `ServiceData_t()`) from the Application Layer that is used to update the service data cached value. This service type has two different client lists: one for those which requested the service `ON_CHANGE` and another for those which requested the service delivery flow to be `CONTINUOUS`. In order to update its clients periodically, **`DataService`** also follows the **`UpdateInterface`**.

5.5.2 Operation

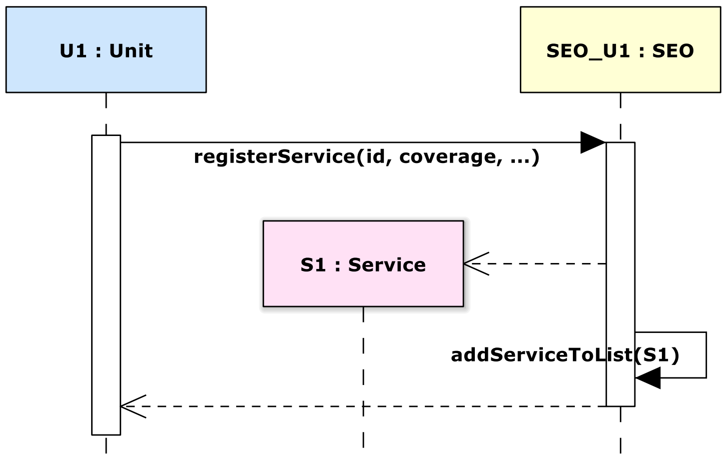
Once a Service is registered, HAMSTER makes it simple to manage it and for other Units to request it. The Application Layer uses a HAMSTER Core Layer API to manage and operate services, while providing callbacks to implement them.

5.5.2.1 Service Registration

Even though each Service Type require different registration elements (Data Services need a callback to retrieve the data and Command Services need a callback to execute the service), the registration process is similar for all Services, as illustrated by the diagram shown in Figure 28. The **`U1`** object request the registration of a service S1; the request is forwarded to its

SEO (`SEO_U1`) through the SEMU Platform (not present in this diagram). `SEO_U1` then creates the `S1` object representing the registered service and add it to its `registered_services` list.

Figure 28 – Service Registration process.



Source: Rodrigues and Branco (2022).

5.5.2.2 *Service Request*

In HAMSTER, service requests differ depending on the service type and its delivery flow. Figure 29 illustrates Unit `U2` requesting the Service `S1` registered by Unit `U1` previously in Figure 28.

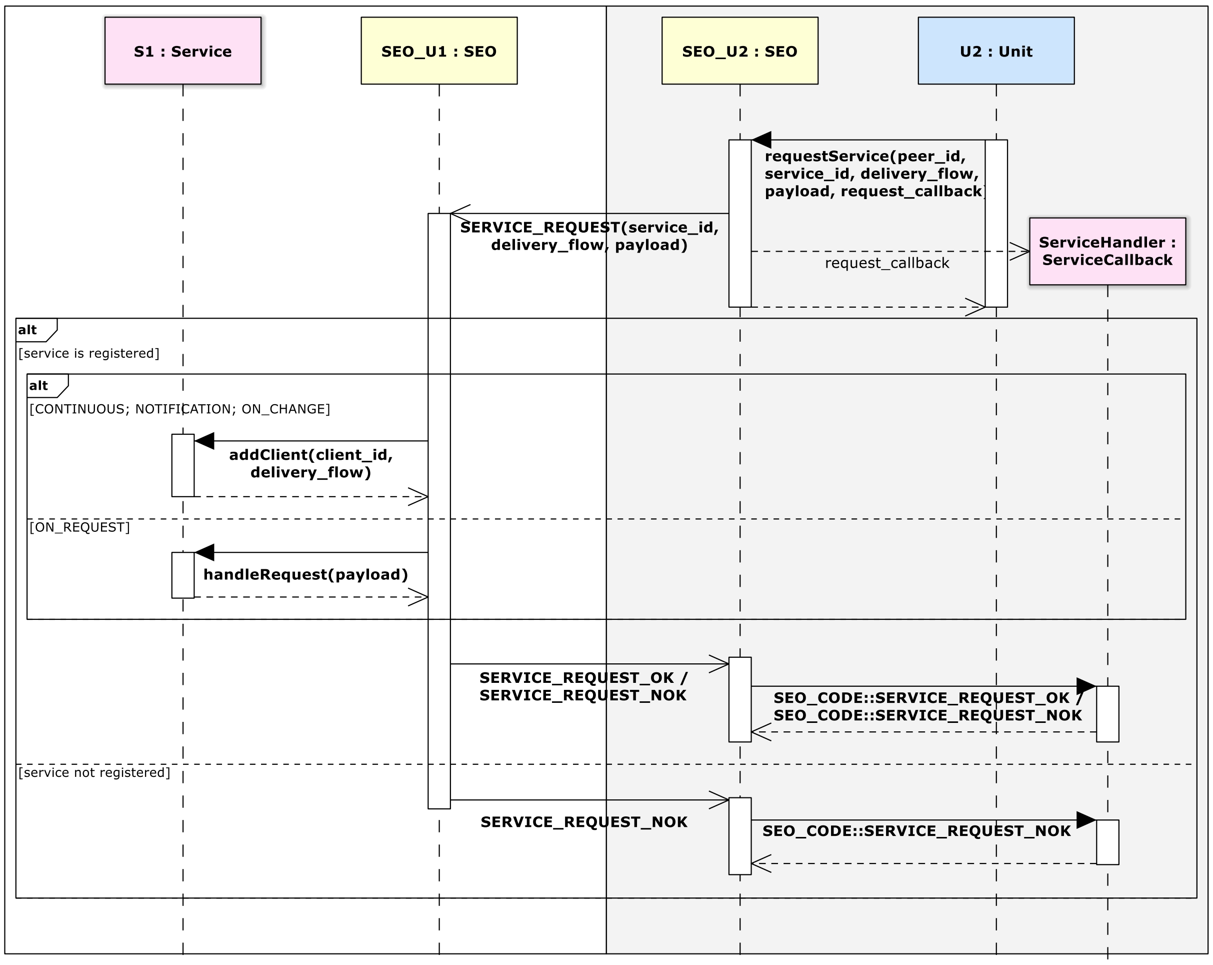
5.5.2.2.1 Service client request

The Application Layer of client `U2` requests a service to its SEO (`SEO_U2`) by calling `requestService()`, providing (1) the Unit ID from which the service is being requested; (2) the Service identification; (3) the desired delivery flow; (4) an optional payload/buffer with data for the request and (5) a callback to be invoked every time a message related to this request arrives at `SEO_U2`. The `SEO_U2` prepares the SERVICE_REQUEST message. If the message is successfully sent, the Platform creates a `ServiceHandler` with the provided callback and returns a positive or negative return to the Application Layer.

5.5.2.2.2 Service provider response

Upon receiving the SERVICE_REQUEST message, the SEO Platform will verify if there is any registered service with the desired ID. If so, `SEO_U1` will take different actions depending on the service delivery flow: for `CONTINUOUS`, `NOTIFICATION` and `ON_CHANGE` requests, the Client ID is added to `S1` client list; for `ON_REQUEST` flow, `SEO_U1` starts the `CommandExecution` callback method in a new thread and waits until service timeout or completion. `SEO_U1` answers with SERVICE_REQUEST_OK on operation success or SERVICE_REQUEST_NOK on fail. This

Figure 29 – Service request between HAMSTER Units.


S1 : Service
SEO_U1 : SEO
SEO_U2 : SEO
U2 : Unit
requestService(peer_id, service_id, delivery_flow, payload, request_callback)
SERVICE_REQUEST(service_id, delivery_flow, payload)
request_callback
ServiceHandler : ServiceCallback
alt
[service is registered]
alt
[CONTINUOUS; NOTIFICATION; ON_CHANGE]
addClient(client_id, delivery_flow)
[ON_REQUEST]
handleRequest(payload)
SERVICE_REQUEST_OK / SERVICE_REQUEST_NOK
SEO_CODE::SERVICE_REQUEST_OK / SEO_CODE::SERVICE_REQUEST_NOK
[service not registered]
SERVICE_REQUEST_NOK
SEO_CODE::SERVICE_REQUEST_NOK

Source: Rodrigues and Branco (2022).

result is forwarded by `SEO_U2` to the `ServiceHandler` created by the service request to deal with the result.

5.5.2.3 Service Deliver

This stage is very simple, but very particular for every service type/delivery flow. As already mentioned, `ON_REQUEST` services (*Data* or *Command* services, as shown by Table 4) execute its action upon receiving message SERVICE_REQUEST and can reply with either message SERVICE_REQUEST_OK or SERVICE_REQUEST_NOK depending on its result — response message support a payload to be sent to the service Client. `ON_NOTIFICATION` services (*Event* Services) deliver a SERVICE_NOTIFICATION message to all its clients when method `notifyEventService()` is invoked from the Application Layer. Finally, services with `CONTINUOUS` and `ON_CHANGE` delivery (basically *Data* services not on request) send a SERVICE_DATA message to all its clients when necessary: periodically or when the data value changes.

5.5.2.4 Service Consumption

When the SEO receives any delivery message, it locates the `ServiceHandler` created on the service request and invokes its callback with a correspondent `SEO_CODE` value, allowing the Application Layer to consume the service. It is important to point out that when requesting a service the Application Layer must be prepared to handle all possible values in its request callback implementation.

5.6 The Navigation Phases Platform

As said before, the Navigation Phases (NP) platform is the only HAMSTER Platform that has significant differences between the implementation of a HAMSTER Entity and a HAMSTER Object. Therefore, these implementations are referred as NP MANAGER and NP AGENT, respectively.

The NP Platform is responsible for managing the state of all Objects in an Entity while executing a Task. Currently, NP manages the ON/OFF state (i.e., if the object is turned on or off) of object groups (essential/MAIN or TASK SPECIFIC) for simplification and performance.

5.6.1 Model and Implementation

Figure 30 shows the model for the Navigation Phases Platform. Classes `NPManager` and `NPAgent` are implemented differently, with the first managing all task-related functionality, while the second deals only with the Object State during each Task phase.

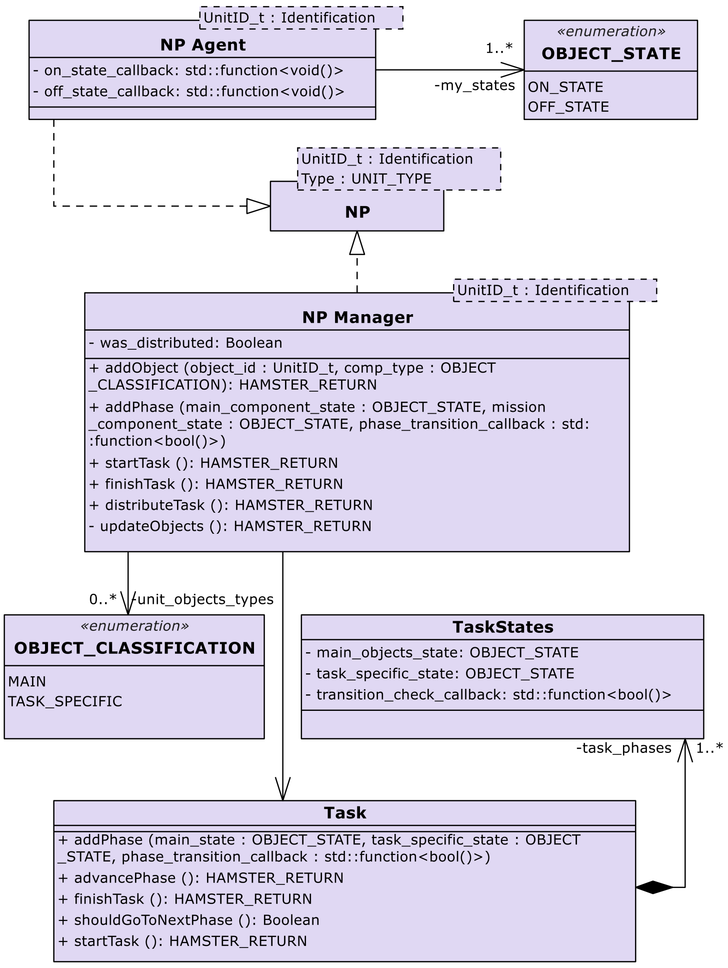
As already described, the `NPManager` manages the Entity's `Task`, which is consisted of different *phases*. Each phase has an associated `TaskState` in which both MAIN and TASK SPECIFIC Objects have an `OBJECT_STATE` and a *transition callback* given by the Application Layer that is invoked periodically to verify if the current phase has finished. Through the `NPManager`, the Entity is able to classify its Objects, include the necessary phases into the Task and *distribute* the Task, i.e., communicate every object of its state in all Task phases in advance.

The `NPAgent` is responsible for making the Object to switch states if necessary. For that, it counts with two different callbacks which are invoked when the Object is supposed to be turned on (`on_state_callback`) and off (`off_state_callback`). The agent is also responsible for storing the current or all Object states, if the Task was distributed.

5.6.2 Operation

Figure 31 illustrates the basic NP operation. The Application Layer of an Entity represented in `U1` can, in different points in time, add an Object ID and classification in a specific list in `NPManager` (`addObject()`), add different phases to the task (`addPhase()`) and distribute the Task to all Objects in the aforementioned list (`distributeTask()`). Task distribution is done

Figure 30 – The Navigation Phases Platform model.


UnitID_t : Identification
NP Agent
- on_state_callback: std::function<void()>
- off_state_callback: std::function<void()>
1..*
-my_states
«enumeration»
OBJECT_STATE
ON_STATE
OFF_STATE
UnitID_t : Identification
Type : UNIT_TYPE
NP
UnitID_t : Identification
NP Manager
- was_distributed: Boolean
+ addObject (object_id : UnitID_t, comp_type : OBJECT_CLASSIFICATION): HAMSTER_RETURN
+ addPhase (main_component_state : OBJECT_STATE, mission_component_state : OBJECT_STATE, phase_transition_callback : std::function<bool()>)
+ startTask (): HAMSTER_RETURN
+ finishTask (): HAMSTER_RETURN
+ distributeTask (): HAMSTER_RETURN
- updateObjects (): HAMSTER_RETURN
0..*
-unit_objects_types
«enumeration»
OBJECT_CLASSIFICATION
MAIN
TASK_SPECIFIC
TaskStates
- main_objects_state: OBJECT_STATE
- task_specific_state: OBJECT_STATE
- transition_check_callback: std::function<bool()>
-task_phases
1..*
Task
+ addPhase (main_state : OBJECT_STATE, task_specific_state : OBJECT_STATE, phase_transition_callback : std::function<bool()>)
+ advancePhase (): HAMSTER_RETURN
+ finishTask (): HAMSTER_RETURN
+ shouldGoToNextPhase (): Boolean
+ startTask (): HAMSTER_RETURN

Source: Rodrigues and Branco (2022).

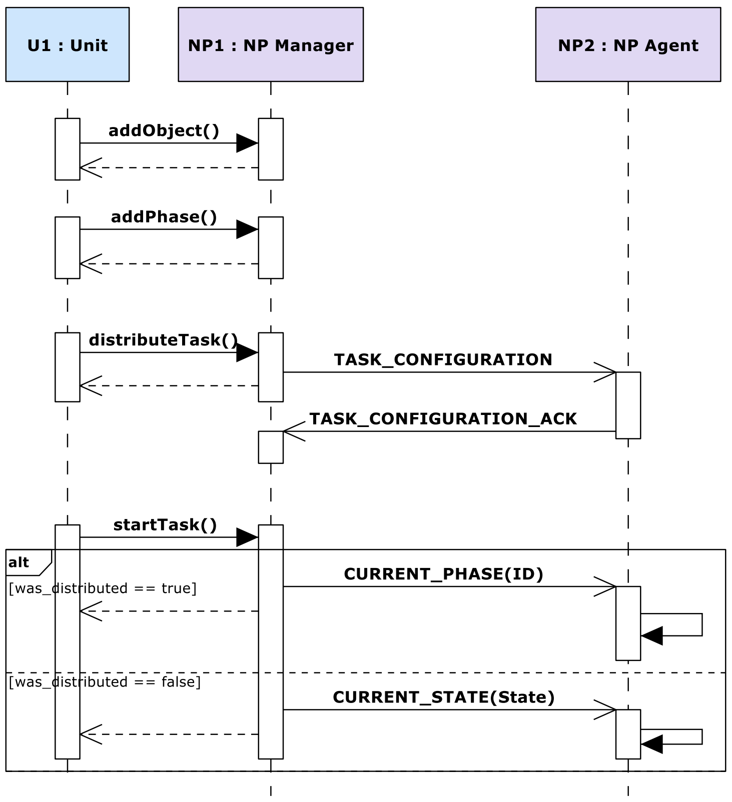
through message TASK_CONFIGURATION sent from the Manager to all Agents from Objects added to the Manager's list.

Once the task is started (`startTask()`), the Manager will notify the **`NPAgent`** of all Entity's Objects of the new phase (CURRENT_PHASE) if the Task was previously distributed or the new state (CURRENT_STATE) if not. Regardless of the received message, the Agent calls the appropriate callback if the Object is supposed to be in ON or OFF state. The messages and subsequent callback invocation happens every time a new Task phase starts, which is determined by invoking `shouldGoToNextPhase()` method from Task periodically.

5.7 Final Remarks

This Chapter provided a detailed description of the HAMSTER prototype. The details on both design an operation were discussed and the abstractions used to make HAMSTER a flexible security solution to be deployed in complex and diverse systems were shown.

Figure 31 – The Navigation Phases Platform operation.



There are still many opportunities ahead for HAMSTER. The inclusion of other security measures such as Access Control mechanisms in *Cloud*–SPHERE and Context-Aware security with NCI as discussed in Section 4.4.3.1 can increase the overall security of the system. The SMU Platform is still to be modeled and implemented in the prototype, increasing overall safety of HAMSTER Units.

Also, only one scenario (centralized architecture with known formation) is currently implemented in the prototype. Extending HAMSTER to other operating scenarios as described in Section 4.7 implies in tackling many challenges.

Unknown formation scenarios will demand a negotiation on which algorithms to use for Cryptography and other security mechanisms. In this case, an adequate model would provide a good abstraction so different strategies may be deployed (power-aware, security-aware, etc) depending on the application. Services and available security mechanisms from a Unit can be discovered through the envisioned Broadcast Gate, but both the Gate and a protocol for this discovery needs to be implemented. Also, key management and how to certify identities are also challenges in decentralized scenarios and, as with IoT, demand constraint-aware solutions.

Centralized scenarios, specially in UAV swarms in which there is a ‘leader UAV’, have to deal with the possibility of a leadership handover. The swarm leader would probably consume more power due to extra responsibilities, and with the use of smaller UAVs to compose a swarm a leadership handover would be necessary. In this case, an adequate abstraction is also necessary so diverse techniques could be applied depending on the application context.

Finally, modifying the prototype so network configuration parameters (such as network addresses) and task stages can be configured dynamically would allow the application of HAMSTER in UAV-enabled service hubs, in which the swarm that will provide the service is dynamically chosen depending on onboard sensors, battery level, etc.

Even though there are still many opportunities for improvement, the current prototype already enabled a different number of real-life applications, as shown by the proof of concepts shown by next Chapter.

CHAPTER 6

RESULTS

In this Chapter, the results obtained with HAMSTER are exposed and detailed. Each result covers one different aspect or application of the HAMSTER and function as a Proof-of-Concept (PoC) scenarios that HAMSTER can aid UAVs and other cyber-physical systems to be integrated in a communicating system like the IoT and provide services with secure communication.

Since the prototype handles only one of the four operation scenarios envisioned for HAMSTER operation, all cases refer to a *Known formation, centralized operation*. This means that every HAMSTER UNIT already has the identification and authentication-related information of all its peers, and there is a Ground Control Station (GCS) that acts as the Central Unit of the system.

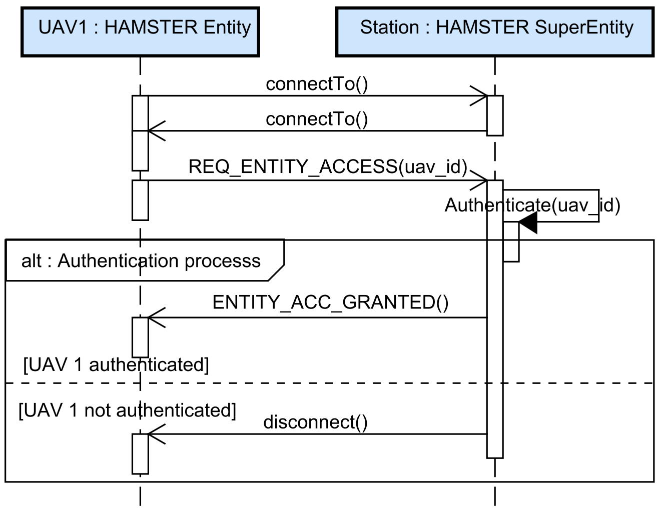
6.1 PoC 1: HAMSTER Authentication

This PoC was published in (RODRIGUES; PIGATTO; BRANCO, 2018a) and demonstrates the authenticated connection between two HAMSTER ENTITIES mediated by the Central Unit, which holds the authentication information of all ENTITIES in the system.

Figure 32 shows the protocol for a UAV authentication to the GCS in a centralized architecture, known formation scenario. After connecting to the station, the UAV sends message REQ_ENTITY_ACCESS with its own authorization information. The CSU module will then invoke the proper Authenticator. If the authentication is successful, the station will answer with a ENTITY_ACC_GRANTED message. Otherwise, the station closes the connection with the UAV. As already discussed, the authentication process may result in different access levels depending on the trust between Units.

Figure 33 shows the protocol for an authenticated connection between two HAMSTER ENTITIES (`UAV1` and `UAV2`) with the HAMSTER SUPER ENTITY (`Station`) as a media-

Figure 32 – HAMSTER ENTITY authentication in the HAMSTER SUPER ENTITY.



Source: Rodrigues, Pigatto and Branco (2018a).

tor for the connection and authentication. The ENTITY must request the relevant connection data of the desired peer to the SUPER ENTITY, given the PEER identification.

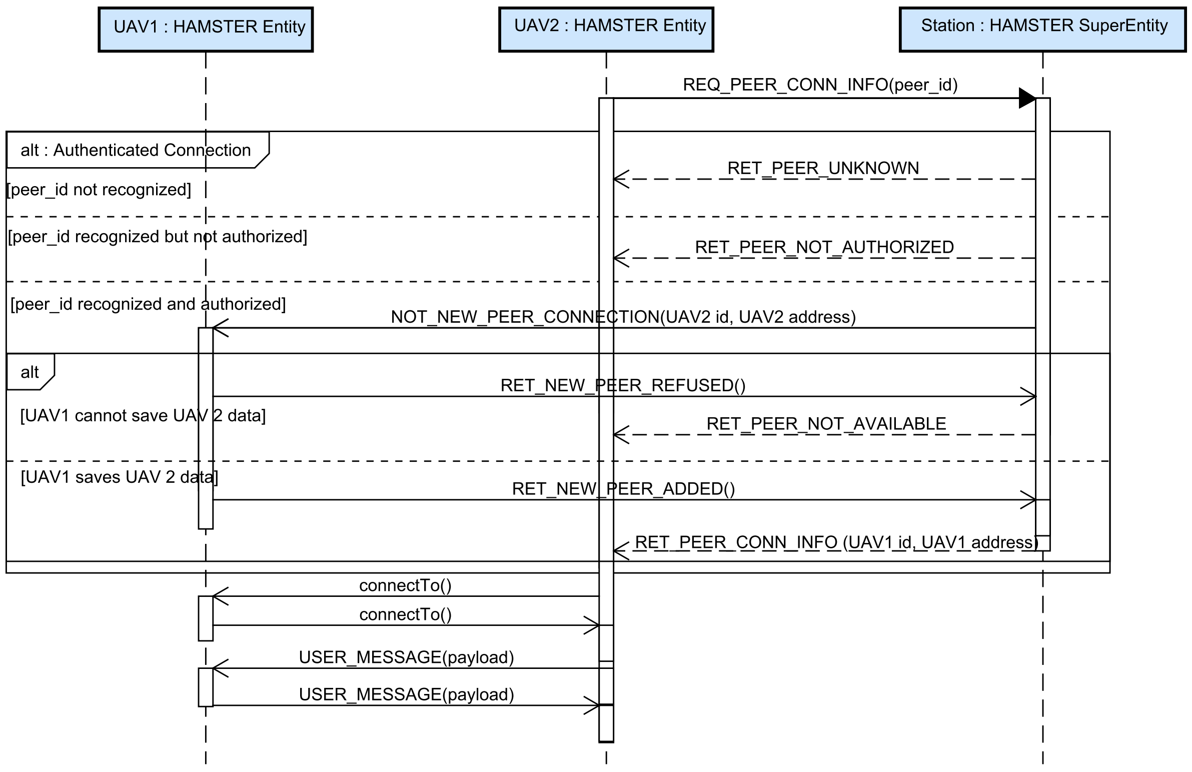
There are more than one way to obtain the PEER identification. It can be provided along with mission parameters in a swarm mission, for example, or it can be queried from both the SUPER ENTITY in a centralized architecture or from the network in a decentralized architecture.

The authenticated connection process begins with message REQ_PEER_CONN_INFO sent from an ENTITY (`UAV2` in Figure 33) to the SUPER ENTITY (`Station` in Figure 33) with the `HAMSTER ID` of the peer (`UAV1` in Figure 33). Upon receiving this message, the SUPER ENTITY's CSU verifies if the desired peer is connected and authenticated in the system. If the SUPER ENTITY does not recognize the peer as belonging to the system, it returns the message RET_PEER_UNKNOWN. If the peer is recognized but it is not connected/authorized, the SUPER ENTITY returns message RET_PEER_NOT_AUTHORIZED.

If the peer ID is recognized and is authorized, the SUPER ENTITY then notifies the peer (`UAV1`) of a new connection request by sending message NOT_NEW_PEER_CONNECTION. Upon receiving this message, the CSUin the `UAV1` will try to add the communication and security information of the ENTITY which made the request (in this case, `UAV2`). If the insertion fails (for instance, `UAV2` uses a cryptographic algorithm not available in `UAV1`), the ENTITY answers the notification with RET_NEW_PEER_REFUSED, and the SUPER ENTITY answers the initial connection request with message RET_PEER_NOT_AVAILABLE. If the insertion succeeds, response RET_NEW_PEER_ADDED is sent, and message RET_PEER_CONN_INFO is sent by the SUPER ENTITY with all necessary information to establish a connection, including network parameters such as IP address and security configurations like cryptography keys.

After receiving the desired communication info, the ENTITY follows with the connection

Figure 33 – *Cloud*–SPHERE platform and its modules.


UAV1 : HAMSTER Entity
UAV2 : HAMSTER Entity
Station : HAMSTER SuperEntity
REQ_PEER_CONN_INFO(peer_id)
alt : Authenticated Connection
[peer_id not recognized]
RET_PEER_UNKNOWN
[peer_id recognized but not authorized]
RET_PEER_NOT_AUTHORIZED
[peer_id recognized and authorized]
NOT_NEW_PEER_CONNECTION(UAV2 id, UAV2 address)
alt
RET_NEW_PEER_REFUSED()
[UAV1 cannot save UAV 2 data]
RET_PEER_NOT_AVAILABLE
[UAV1 saves UAV 2 data]
RET_NEW_PEER_ADDED()
RET_PEER_CONN_INFO (UAV1 id, UAV1 address)
connectTo()
connectTo()
USER_MESSAGE(payload)
USER_MESSAGE(payload)

Source: Rodrigues, Pigatto and Branco (2018a).

process. In this step, an optional ENTITY mutual authentication can be performed for additional security. After the connection is established, both ENTITIES can securely exchange user-specific messages defined by the application.

6.1.1 Environment Setup

For this PoC, the HAMSTER ID abstraction was resolved in an Universally Unique Identifier (UUID). The authentication is ID-based and only checks if the ID sent in the message is one present in the Super Entity known entities. The Communication Interface abstraction was implemented using network TCP sockets, but there is no other security measure like support for cryptography. The demonstration is performed in a desktop environment.

The test environment is composed by a HAMSTER SUPER ENTITY (`Station`) and three HAMSTER ENTITIES (`UAV1`, `UAV2` and `UAV3`). Every ENTITY behaves differently in order to cover different situations of the authentication protocol:

- `UAV3` connects to Station but never authenticates itself.

- UAV2 authenticates itself in Station, tries to connect with a random UUID, then tries to connect to UAV3, connects itself with UAV1 and send the message *"Hello, my friend!"*.
- UAV1 authenticates itself in Station and waits for any connection request and messages. Upon receiving message *"Hello, my friend!"*, it answers with message *"Hello back!"*.

For better log readability, a predefined order of execution was defined. First, the Station is loaded, followed by UAV1. After UAV1 is authenticated in station, UAV3 is loaded. Finally, UAV2 is loaded and perform the actions described above.

6.1.2 Obtained Results

Test results shown in logs (Some lines were suppressed for better readability) expose a successful entity authentication and message exchange between entities. In Station log (Figure 35), there is a first incoming connection from UAV1 followed by the message pair characterizing a successful Entity authentication (messages REQ_ENTITY_ACCESS and its response ENTITY_ACC_GRANTED). A following connection with no authentication messages states UAV3 connection, whose log is suppressed. A last connection is exposed, followed by UAV2 authentication.

Next, it is possible to discern the messages of the connection attempts made by UAV2 (Figure 34): the first, to an ID unknown by the Station, returns message RET_PEER_UNKNOWN; the second, to UAV3, returns message RET_PEER_NOT_AUTHORIZED; the last, to UAV1, causes the exchange of messages between Station and UAV1 to setup the authorized communication, with response RET_PEER_CONN_INFO being sent next. In UAV2 log, it is possible to see the record of custom messages *"Hello, my friend!"* and *"Hello back!"* between the UAVs.

Figure 34 – Log for UAV2 in the proposed authentication scenario.

```
0000005 ~ [EVENT] Trying to connect to random UUID.
0000006 ~ [EVENT] Message REQ_PEER_CONN_INFO sent.
0000007 ~ [EVENT] Message RET_PEER_UNKNOWN received.
0000009 ~ [EVENT] Trying to connect to UAV3.
0000010 ~ [EVENT] Message REQ_PEER_CONN_INFO sent.
0000011 ~ [EVENT] Message RET_PEER_NOT_AUTHORIZED received.
0000013 ~ [EVENT] Trying to connect to UAV1.
0000014 ~ [EVENT] Message REQ_PEER_CONN_INFO sent.
0000015 ~ [EVENT] Message RET_PEER_CONN_INFO received.
0000018 ~ [EVENT] New connection, client port = 6383
0000019 ~ [EVENT] User-defined message "Hello, my friend!" sent.
0000022 ~ [EVENT] User-defined message "Hello back!"received.
```

Source: Rodrigues, Pigatto and Branco (2018a).

Figure 35 – Log for `Station` in the proposed authentication scenario.

```
0000002 ~ [EVENT] New connection, client port = 64750
0000004 ~ [EVENT] Message REQ_ENTITY_ACCESS received.
0000005 ~ [EVENT] Message ENTITY_ACC_GRANTED sent.
0000006 ~ [EVENT] New connection, client port = 239
0000008 ~ [EVENT] New connection, client port = 751
0000010 ~ [EVENT] Message REQ_ENTITY_ACCESS received.
0000011 ~ [EVENT] Message ENTITY_ACC_GRANTED sent.
0000012 ~ [EVENT] Message REQ_PEER_CONN_INFO received.
0000013 ~ [EVENT] Message RET_PEER_UNKNOWN sent.
0000014 ~ [EVENT] Message REQ_PEER_CONN_INFO received.
0000015 ~ [EVENT] Message RET_PEER_NOT_AUTHORIZED sent.
0000016 ~ [EVENT] Message REQ_PEER_CONN_INFO received.
0000017 ~ [EVENT] Message NOT_NEW_PEER_CONNECTION sent.
0000018 ~ [EVENT] Message RET_NEW_PEER_ADDED received.
0000019 ~ [EVENT] Message RET_PEER_CONN_INFO sent.
```

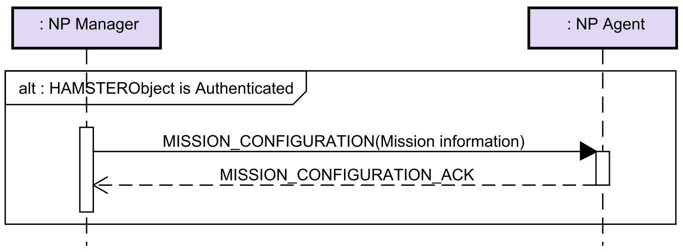
Source: Rodrigues, Pigatto and Branco (2018a).

6.2 PoC 2: The Navigation Phases (NP) Platform

This PoC was published in (RODRIGUES; PIGATTO; BRANCO, 2018b) and demonstrates how the NP Platform can be used to manage the ON/OFF state of HAMSTER OBJECTS during a task (called "Mission" at that time).

The mission information distribution takes place after the connection of all HAMSTER OBJECTS. The `NPManager` in the HAMSTER ENTITY relays the mission information for its OBJECTS which were successfully authenticated, as shown by Figure 36.

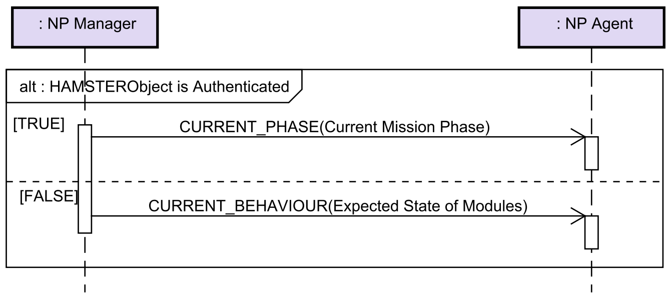
Figure 36 – NP Mission Phases Distribution.


: NP Manager
: NP Agent
alt : HAMSTERObject is Authenticated
MISSION_CONFIGURATION(Mission information)
MISSION_CONFIGURATION_ACK

Source: Rodrigues, Pigatto and Branco (2018b).

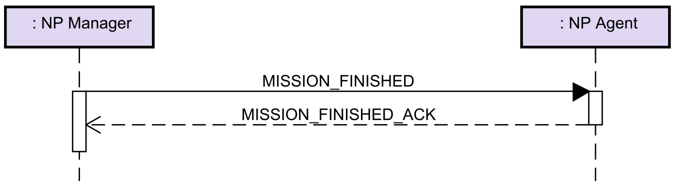
During mission operation (Figure 37), `NPManager` notifies the authenticated modules of the current phase through message CURRENT_PHASE, and commands the ON/OFF state of unauthorized components through message CURRENT_BEHAVIOUR. The prior distribution to authenticated (and probably more critical) Objects make the process of phase switching very brief, contributing to time-constrained operation scenarios. After the mission is finished, the HAMSTER Objects are notified through message MISSION_FINISHED (Figure 38).

Figure 37 – NP operation during a mission.


: NP Manager
: NP Agent
alt : HAMSTERObject is Authenticated
[TRUE]
CURRENT_PHASE(Current Mission Phase)
[FALSE]
CURRENT_BEHAVIOUR(Expected State of Modules)

Source: Rodrigues, Pigatto and Branco (2018b).

Figure 38 – NP operation at the end of the mission.



Source: Rodrigues, Pigatto and Branco (2018b).

6.2.1 Environment Setup

For this PoC, the HAMSTER ID abstraction was resolved in an UUID and the Communication Interface abstraction was implemented using network TCP sockets, but there is no other security measures like cryptography support. The demonstration is performed in a desktop environment.

The test environment is composed by a HAMSTER ENTITY (`UAV`) containing four HAMSTER OBJECTS (`Autopilot`, `RGBCamera`, `MultispectralCamera` and `SensorUnit`) summarized by Table 6. The Sensor Unit is a CLUSTER with three sensors: an Altitude Sensor, a Dynamic Pressure Sensor and a Temperature sensor. The first two are considered necessary for UAV operation. Also, `MultispectralCamera` is the only module that do not authenticate itself. Since it is not considered crucial for UAV operation, the mission can be executed.

The UAV mission and detailed information on active components for this demonstration is shown by Table 7. During initialization, all modules are ON. During cruising to mission site and back to base, mission-specific modules are turned OFF. Once the vehicle arrives at mission site, all modules are turned ON and the mission performed. Back to base, all modules stay ON for health checking and mission data acquisition.

Table 6 – HAMSTER OBJECTS included in the demonstration.

Object	**Unit Type**	**Classification**
Autopilot	HAMSTER MODULE	`MAIN`
RGB Camera	HAMSTER MODULE	`MISSION SPECIFIC`
Multispectral	HAMSTER MODULE	`MISSION SPECIFIC`
Sensor Unit	HAMSTER CLUSTER	`MAIN`

Source: Rodrigues, Pigatto and Branco (2018b).

Table 7 – Mission Phases for the demonstration.

Number	**Phase**	**Active Objects**
1	Initialization	All
2	Cruising	Main only
3	Mission execution	All
4	Return to base	Main only
5	Finalization	All

Source: Rodrigues, Pigatto and Branco (2018b).

6.2.2 Obtained Results

Test results shown in logs (Some lines were suppressed for better readability) shows the NP messages exchanged between the HAMSTER ENTITY and its HAMSTER OBJECTS.

Figure 39 shows the log for the HAMSTER ENTITY (`UAV`). It is possible to see the connection of the four HAMSTER OBJECTS in different ports. Once all of them are connected, the `NPManager` distributes the mission to all three authenticated modules through the message pair MISSION_CONFIGURATION / MISSION_CONFIGURATION_ACK. During the mission, three messages CURRENT_PHASE are sent to the authenticated HAMSTER OBJECTS, while message CURRENT_BEHAVIOUR is sent to the Multispectral camera. When the mission finishes, `NPManager` notifies `NPAgent` through the MISSION_FINISHED / MISSION_FINISHED_ACK message pair.

Figure 40 shows the messages received by the Multispectral camera, which did not authorize itself. Instead of receiving mission information, the `NPAgent` receives message CURRENT_BEHAVIOUR for each mission phase with the necessary behaviour information.

Finally, Figure 41 shows the messages for the Sensor Unit. The mission is received by `NPAgent` in message MISSION_CONFIGURATION. During mission execution, both altitude and dynamic pressure sensors stay turned on during the entire operation, while the temperature sensor goes off during cruising. The end of the mission is signaled by message MISSION_FINISHED.

Figure 39 – Log in the UAV.

```
0000001 ~ [EVENT] New connection, client port = 1049
0000003 ~ [EVENT] New connection, client port = 1561
0000005 ~ [EVENT] New connection, client port = 2073
0000008 ~ [EVENT] New connection, client port = 2585
0000019 ~ [EVENT] Message MISSION_CONFIGURATION sent.
0000020 ~ [EVENT] Message MISSION_CONFIGURATION sent.
0000021 ~ [EVENT] Message MISSION_CONFIGURATION sent.
0000022 ~ [EVENT] Message MISSION_CONFIGURATION_ACK received.
0000023 ~ [EVENT] Message MISSION_CONFIGURATION_ACK received.
0000024 ~ [EVENT] Message MISSION_CONFIGURATION_ACK received.
0000025 ~ [EVENT] Current phase number = 1
0000026 ~ [EVENT] Message CURRENT_PHASE sent.
0000027 ~ [EVENT] Message CURRENT_BEHAVIOUR sent.
0000028 ~ [EVENT] Message CURRENT_PHASE sent.
0000029 ~ [EVENT] Message CURRENT_PHASE sent.
0000030 ~ [EVENT] Current phase number = 2
                                    ...
0000045 ~ [EVENT] Current phase number = 5
0000046 ~ [EVENT] Message CURRENT_PHASE sent.
0000047 ~ [EVENT] Message CURRENT_BEHAVIOUR sent.
0000048 ~ [EVENT] Message CURRENT_PHASE sent.
0000049 ~ [EVENT] Message CURRENT_PHASE sent.
0000050 ~ [EVENT] Message MISSION_FINISHED sent.
0000051 ~ [EVENT] Message MISSION_FINISHED sent.
0000052 ~ [EVENT] Message MISSION_FINISHED sent.
0000053 ~ [EVENT] Message MISSION_FINISHED_ACK received.
0000054 ~ [EVENT] Message TERMINATE_MISSION sent.
0000055 ~ [EVENT] Message MISSION_FINISHED_ACK received.
0000056 ~ [EVENT] Message MISSION_FINISHED_ACK received.
0000057 ~ [EVENT] Message MISSION_FINISHED_ACK received.
```

Source: Rodrigues, Pigatto and Branco (2018b).

Figure 40 – Log in the Multispectral Camera.

```
0000003 ~ [EVENT] Message CURRENT_BEHAVIOUR received.
0000004 ~ [EVENT] Component Multispectral Camera ON.
0000005 ~ [EVENT] Message CURRENT_BEHAVIOUR received.
0000006 ~ [EVENT] Component Multispectral Camera OFF.
0000007 ~ [EVENT] Message CURRENT_BEHAVIOUR received.
0000008 ~ [EVENT] Component Multispectral Camera ON.
0000009 ~ [EVENT] Message CURRENT_BEHAVIOUR received.
0000010 ~ [EVENT] Component Multispectral Camera OFF.
0000011 ~ [EVENT] Message CURRENT_BEHAVIOUR received.
0000012 ~ [EVENT] Component Multispectral Camera ON.
0000013 ~ [EVENT] Message TERMINATE_MISSION received.
```

Source: Rodrigues, Pigatto and Branco (2018b).

6.3 PoC 3: Securing data exchange

This PoC was published in (RODRIGUES; BRANCO, 2020) and demonstrates the integration of HAMSTER-enabled vehicles in an IoT-alike scenario. In this demonstration, shown in Figure 42, two HAMSTER-enabled UAVs send telemetry data to their Ground Control Station, which in turn publishes the received UAV data in a server which can be accessed by

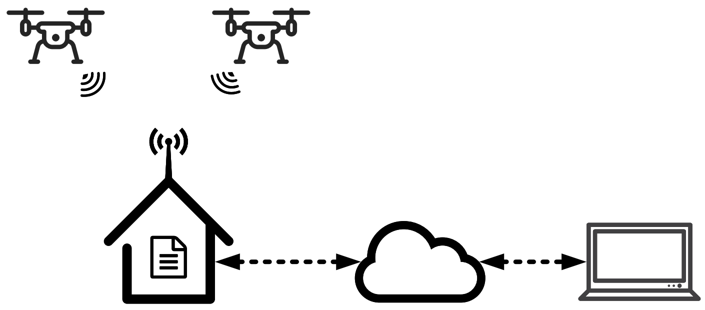
Figure 41 – Log in the `SensorUnit`.

```
0000005 ~ [EVENT] Message MISSION_CONFIGURATION received.
0000006 ~ [EVENT] Message MISSION_CONFIGURATION_ACK sent.
0000007 ~ [EVENT] Message CURRENT_PHASE received.
0000008 ~ [EVENT] Current phase number = 1
0000009 ~ [EVENT] Component Altitude Sensor ON.
0000010 ~ [EVENT] Component Dynamic Pressure Sensor ON.
0000011 ~ [EVENT] Component Temperature Sensor ON.
0000012 ~ [EVENT] Message CURRENT_PHASE received.
0000013 ~ [EVENT] Current phase number = 2
0000014 ~ [EVENT] Component Altitude Sensor ON.
0000015 ~ [EVENT] Component Dynamic Pressure Sensor ON.
0000016 ~ [EVENT] Component Temperature Sensor OFF.
0000017 ~ [EVENT] Message CURRENT_PHASE received.
0000018 ~ [EVENT] Current phase number = 3
0000019 ~ [EVENT] Component Altitude Sensor ON.
0000020 ~ [EVENT] Component Dynamic Pressure Sensor ON.
0000021 ~ [EVENT] Component Temperature Sensor ON.
0000022 ~ [EVENT] Message CURRENT_PHASE received.
0000023 ~ [EVENT] Current phase number = 4
0000024 ~ [EVENT] Component Altitude Sensor ON.
0000025 ~ [EVENT] Component Dynamic Pressure Sensor ON.
0000026 ~ [EVENT] Component Temperature Sensor OFF.
0000027 ~ [EVENT] Message CURRENT_PHASE received.
0000028 ~ [EVENT] Current phase number = 5
0000029 ~ [EVENT] Component Altitude Sensor ON.
0000030 ~ [EVENT] Component Dynamic Pressure Sensor ON.
0000031 ~ [EVENT] Component Temperature Sensor ON.
0000032 ~ [EVENT] Message MISSION_FINISHED received.
0000033 ~ [EVENT] Mission finished.
0000034 ~ [EVENT] Message MISSION_FINISHED_ACK sent.
```

Source: Rodrigues, Pigatto and Branco (2018b).

service clients.

Figure 42 – Test scenario.



Source: Rodrigues and Branco (2020).

The UAV ⇔ Station communication links are handled by HAMSTER platform, while the data exchange represented by a dashed line are done by the Message Queue Telemetry Transport (MQTT) Protocol, which is considered one of the suitable protocols for IoT operation.

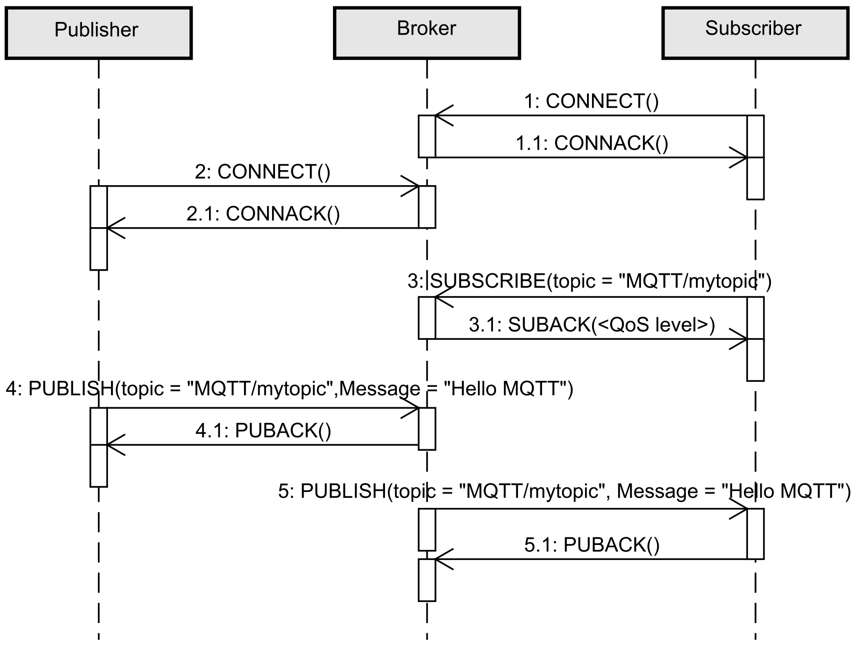
6.3.1 MQTT Protocol

MQTT or Message Queue Telemetry Transport Protocol (MQTT 3.1.1, 2014) is a lightweight topic–based publish/subscribe protocol with many-to-many communication (AL-FUQAHA *et al.*, 2015) whose main purpose is telemetry or remote monitoring.

Publish/Subscribe systems have three main components: the *publishers* which share information, the *subscribers* which consume this information, and a *broker* that forwards the information from publishers to subscribers. In order to receive data, the subscribers *register* their interest in a event. When an event receives data, the broker propagates it to all subscribers (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

MQTT has a small set of messages to manage device connection and the publish/-subscriber scheme (MQTT 3.1.1, 2014). Figure 43 illustrates the message exchange between publisher and subscribers with QoS level 1 (At least once).

Figure 43 – Message Exchange in MQTT Protocol.


Publisher
Broker
Subscriber
1: CONNECT()
1.1: CONNACK()
2: CONNECT()
2.1: CONNACK()
3: SUBSCRIBE(topic = "MQTT/mytopic")
3.1: SUBACK(<QoS level>)
4: PUBLISH(topic = "MQTT/mytopic",Message = "Hello MQTT")
4.1: PUBACK()
5: PUBLISH(topic = "MQTT/mytopic", Message = "Hello MQTT")
5.1: PUBACK()

Source: Rodrigues and Branco (2020).

Publishers and subscribers connect to the broker through the CONNECT Package (messages 1-2 in Figure 43), which carries connection configuration parameters. The broker in turn send the CONNACK Package (messages 1.1-2.1) stating if the connection was accepted or the reason it was refused. To subscribe to a topic, subscribers send the SUBSCRIBE Package, which is answered with a SUBACK Package (messages 3-3.1) by the broker. To publish in a topic, publishers send the PUBLISH Package, whose broker answer will depend on the chosen Qos. For QoS level 1 (At least once), the broker answers with a PUBACK package (messages 4-4.1) and forward it to clients subscribed on that topic (messages 5-5.1).

6.3.2 Environment Setup

To demonstrate the test scenario, HAMSTER ID was resolved in an Universally Unique Identifier (UUID). The Authenticator is ID-based and only checks if the ID sent in the message is one present in the Super Entity known entities. The Communication Interface abstraction was implemented using network TCP sockets. All HAMSTER Units uses Eliptic Curve Cryptography Algorithms implemented by RELIC toolkit (RELIC, 2018) version 0.3.3 built with a 160-bit key. For this demonstration, the abstraction of Heartbeat Handlers was not included.

The test environment is composed by a HAMSTER SUPER ENTITY (`Station`), two HAMSTER ENTITIES (`UAV1` and `UAV2`), a MQTT broker and a MQTT client. The HAMSTER UNITS were implemented as different processes at a desktop on Windows 10, communicating through TCP sockets at localhost. `Station` also communicated through Ethernet with a Mosquitto (ECLIPSE, 2018a) MQTT broker instance on a Hardkernel Odroid XU4 (HARDKERNEL, 2018) running Ubuntu 16.04, and a MQTT client using PAHO library (ECLIPSE, 2018b) was set up on a notebook on Windows 8, also communicating with the MQTT broker through an Ethernet connection.

At first, the `Station` connects with the MQTT broker and waits for both `UAV` to connect and authenticate themselves. Once authenticated, both `UAV`s start sending telemetry data (for this test, the UAV flight dataset available on (MOKHTARZADEH; COLTEN, 2015) was used) to the `Station`.

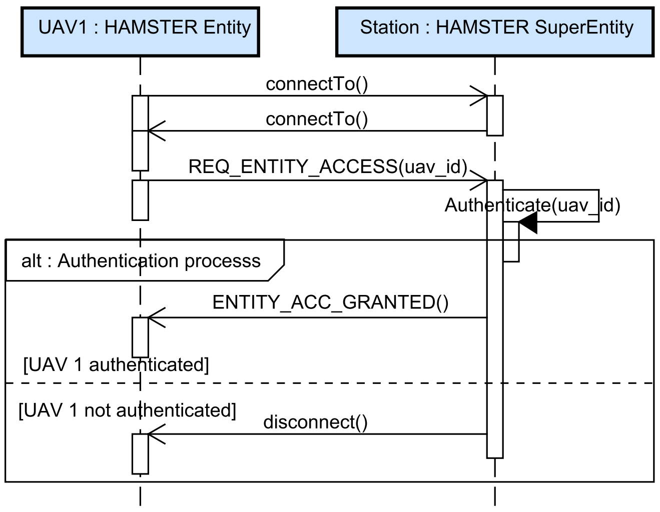
Upon connecting to the `Station`, the `UAV` sends its ID and public cryptographic key in HAMSTER_UNIT_ID (code `0x80000001`). The `Station` connects to the `UAV` the same way, creating a two way connection. After the connection is established, `UAV` sends message REQ_-ENTITY_ACCESS (code `0x40000001`) with its own authorization information, and the Super CSU module will then invoke the proper Authenticator. If the authentication is successful, the station answers with ENTITY_ACC_GRANTED message (code `0x40000003`). Otherwise, the station closes the connection with the UAV. Figure 32 presents this connection/authentication protocol used at the moment by HAMSTER UNITS.

After receiving `UAV` telemetry, the `Station` forwards the latitude, longitude and altitude data to the broker. On the other side, the MQTT client connects to the MQTT broker and subscribes to all topics of both `UAV`s, receiving the data once it is forwarded by the broker. Both MQTT client and `Station` have a username and password to access the MQTT broker.

6.3.3 Obtained Results

Test results shown in logs (some lines were suppressed for better readability) show a successful entity connection and authentication, with subsequent secure message passing between HAMSTER entities. The data received by the `Station` is published in the MQTT broker and forwarded to the subscribed client.

Figure 32 – UAV connection and authentication in Station in *Cloud*–SPHERE (repeated from page 124).



Source: Rodrigues, Pigatto and Branco (2018a).

It is possible to see all exchanged messages for connection and authentication in Figure 44 (Station) and Figure 45 (UAV1). First, the Station connects to the MQTT server. After, it receives two new connections and answers them with HAMSTER_UNIT_ID message. Authentication protocol for both UAVs is also present with messages REQ_ENTITY_ACCESS and its success response message ENTITY_ACC_GRANTED. Message USER_MESSAGE (code 0x00000002) is then used by the UAV to send telemetry data.

Figure 44 – Log for the Station.

```
0000002 [EVENT] Station is initializing.
0000003 [EVENT] Station connected to the server.
0000004 [EVENT] New connection, client port = 21679
0000005 [EVENT] Message HAMSTER_UNIT_ID sent.
0000006 [EVENT] New connection, client port = 22191
0000007 [EVENT] Message HAMSTER_UNIT_ID sent.
0000008 [EVENT] Message REQ_ENTITY_ACCESS received.
0000009 [EVENT] Message ENTITY_ACC_GRANTED sent.
0000010 [EVENT] Message REQ_ENTITY_ACCESS received.
0000011 [EVENT] Message ENTITY_ACC_GRANTED sent.
0000012 [EVENT] Message USER MESSAGE received.
0000013 [EVENT] Message USER MESSAGE received.
```

Source: Rodrigues and Branco (2020).

In Figure 47 is shown the log for the Mosquitto broker, in which is possible to verify MQTT client connection and topic subscription, as well as Station connection and topic publishing, with packages described at Section 6.3.1. Figure 46 shows the log for the MQTT client, which successfully receives published UAV telemetry data.

HAMSTER messages exchanged in localhost captured with Wireshark show a open

Figure 45 – Log for the `UAV1`.

```
0000001 [EVENT] Message HAMSTER_UNIT_ID sent.
0000002 [EVENT] New connection, client port = 21935
0000003 [EVENT] Message REQ_ENTITY_ACCESS sent.
0000004 [EVENT] Message ENTITY_ACC_GRANTED received.
0000005 [EVENT] UAV Initialized.
0000006 [EVENT] UAV is now sending telemetry data.
0000007 [EVENT] Message USER MESSAGE sent.
0000008 [EVENT] UAV is now sending telemetry data.
0000009 [EVENT] Message USER MESSAGE sent.
0000010 [EVENT] UAV is now sending telemetry data.
0000011 [EVENT] Message USER MESSAGE sent.
0000012 [EVENT] UAV is now sending telemetry data.
```

Source: Rodrigues and Branco (2020).

Figure 46 – Log for the MQTT Client.

```
0000001 [EVENT] uas/UAV1/latitude = 6445574336d
0000002 [EVENT] uas/UAV1/longitude = 6444612096
0000003 [EVENT] uas/UAV1/altitude = -2145566304
0000004 [EVENT] uas/UAV2/latitude = 6443780769P
0000005 [EVENT] uas/UAV2/longitude = 4294953568M
0000006 [EVENT] uas/UAV2/altitude = 0
0000007 [EVENT] uas/UAV1/latitude = 6445574336
0000008 [EVENT] uas/UAV1/longitude = 6444612096e
0000009 [EVENT] uas/UAV1/altitude = -2145566304
```

Source: Rodrigues and Branco (2020).

HAMSTER_UNIT_ID message (code `0x80000001`), but all subsequent messages (telemetry messages included) are encrypted and therefore protected (Figure 48 and Figure 49). On the MQTT side, however, the publish messages do not offer any kind of data protection, making data accessible to unauthorized parties (Figure 50). Because of that, more security measures are needed when dealing with MQTT protocol. This security can be a Transport layer SSL certificate as supported by Mosquitto, an Application layer encryption or both.

6.4 PoC 4: Context-Aware Security

This PoC was published in (RODRIGUES; PIGATTO; BRANCO, 2020) and demonstrates how contextual information can be translated into the Perceived Security Index (PSI) discussed in Section 4.4. In this Section, a demonstration on PSI estimation in HAMSTER is provided for an application with UAVs.

In this scenario, three small ($< 1m$ size) UAVs are used to capture images of a crop. They communicate in an ad hoc manner and each one has an IMU and a GPS to identify its location, a camera to capture the images, an autopilot that also activates the camera, a propeller and

Figure 47 – Log for the Mosquitto broker.

```
1538286850: mosquitto version 1.5.2 starting
1538286850: Using default config.
1538286850: Opening ipv4 listen socket on port 1883.
1538286850: Opening ipv6 listen socket on port 1883.
1538286861: New connection from 10.70.1.236 on port 1883.
1538286861: New client connected from 10.70.1.236 as client (c1, k20, u'client1').
1538286861: No will message specified.
1538286861: Sending CONNACK to client (0, 0)
1538286861: Received SUBSCRIBE from client
1538286861:         uas/UAV1/# (QoS 1)
1538286861: client 1 uas/UAV1/#
1538286861: Sending SUBACK to client
1538286861: Received SUBSCRIBE from client
1538286861:         uas/UAV2/# (QoS 1)
1538286861: client 1 uas/UAV2/#
1538286861: Sending SUBACK to client
1538286879: New connection from 10.70.1.20 on port 1883.
1538286879: New client connected from 10.70.1.20 as Station (c1, k60, u'station').
1538286879: No will message specified.
1538286879: Sending CONNACK to Station (0, 0)
1538286900: Received PUBLISH from Station (d0, q1, r1, m1, 'uas/UAV1/latitude', ... (10 bytes))
1538286900: Sending PUBACK to Station (Mid: 1)
1538286900: Sending PUBLISH to client (d0, q1, r0, m1, 'uas/UAV1/latitude', ... (10 bytes))
1538286900: Received PUBLISH from Station (d0, q1, r1, m2, 'uas/UAV1/longitude', ... (10 bytes))
1538286900: Sending PUBACK to Station (Mid: 2)
1538286900: Sending PUBLISH to client (d0, q1, r0, m2, 'uas/UAV1/longitude', ... (10 bytes))
1538286900: Received PUBLISH from Station (d0, q1, r1, m3, 'uas/UAV1/altitude', ... (11 bytes))
1538286900: Sending PUBACK to Station (Mid: 3)
1538286900: Sending PUBLISH to client (d0, q1, r0, m3, 'uas/UAV1/altitude', ... (11 bytes))
1538286900: Received PUBACK from client (Mid: 1)
1538286900: Received PUBACK from client (Mid: 2)
1538286900: Received PUBACK from client (Mid: 3)
1538286900: Received PUBLISH from Station (d0, q1, r1, m4, 'uas/UAV2/latitude', ... (10 bytes))
1538286900: Sending PUBACK to Station (Mid: 4)
1538286900: Sending PUBLISH to client (d0, q1, r0, m4, 'uas/UAV2/latitude', ... (10 bytes))
1538286900: Received PUBLISH from Station (d0, q1, r1, m5, 'uas/UAV2/longitude', ... (10 bytes))
1538286900: Sending PUBACK to Station (Mid: 5)
1538286900: Received PUBACK from client (Mid: 4)
```

Source: Rodrigues and Branco (2020).

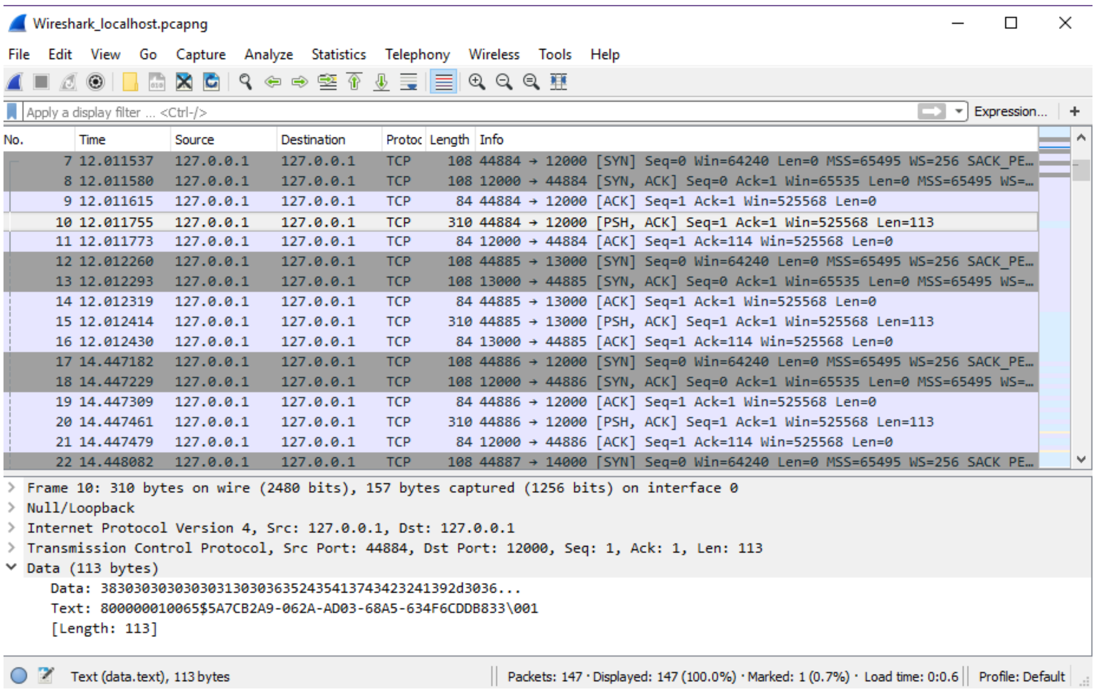
servomotors. All UAVs know the identity of their communication peers and perform (1) internal module authentication before (2) mutual UAV authentication so the mission can be started. The UAVs communicate with each other using asymmetric cryptography and take-off/landing and mission location is the same controlled area.

6.4.1 External Context Initial Evaluation

In order to evaluate the external context security index (ψ_{ex}), threats and available security measures need to be analyzed. The perceived security will be at its maximum when the risk coming from the external environment is minimum. In this case, the risk is analyzed based on attack difficulty, impact and

For determining attack difficulty, threats shown in Figure 6 (page 82) are classified based

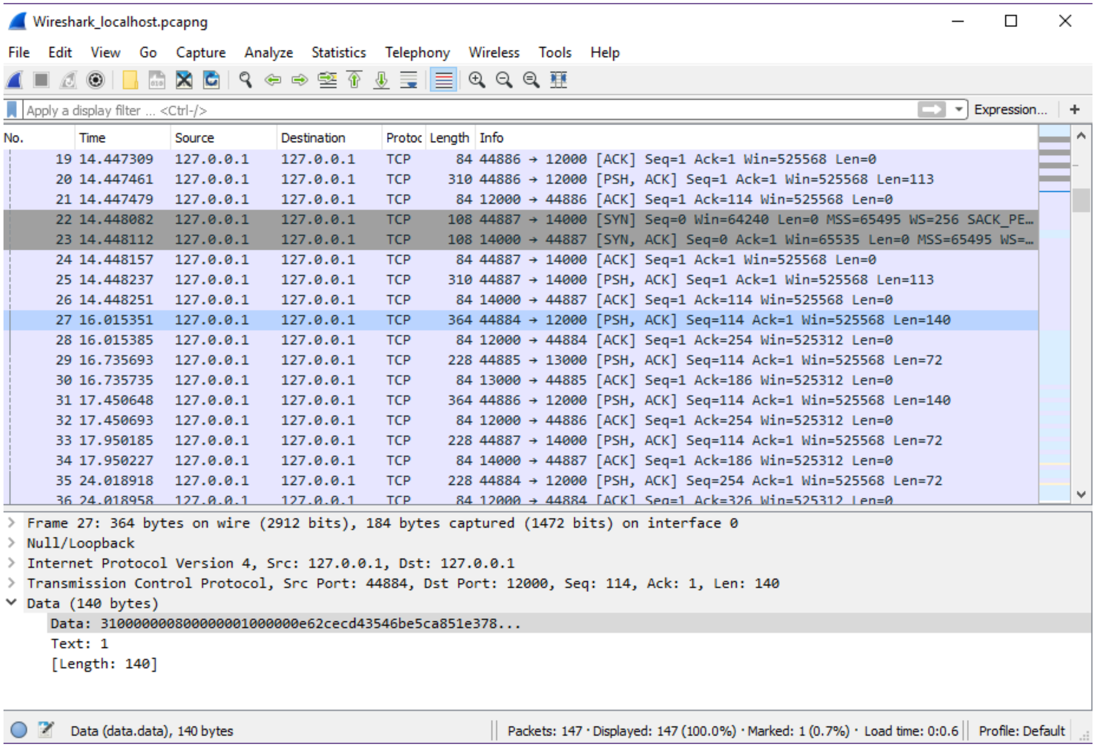
Figure 48 – Message HAMSTER_UNIT_ID (code 0x80000001) sent without encryption.


Wireshark_localhost.pcapng

```
No.  Time      Source     Destination  Protoc Length Info
   7 12.011537 127.0.0.1  127.0.0.1    TCP    108 44884 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PE…
   8 12.011580 127.0.0.1  127.0.0.1    TCP    108 12000 → 44884 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=…
   9 12.011615 127.0.0.1  127.0.0.1    TCP     84 44884 → 12000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
  10 12.011755 127.0.0.1  127.0.0.1    TCP    310 44884 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=113
  11 12.011773 127.0.0.1  127.0.0.1    TCP     84 12000 → 44884 [ACK] Seq=1 Ack=114 Win=525568 Len=0
  12 12.012260 127.0.0.1  127.0.0.1    TCP    108 44885 → 13000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PE…
  13 12.012293 127.0.0.1  127.0.0.1    TCP    108 13000 → 44885 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=…
  14 12.012319 127.0.0.1  127.0.0.1    TCP     84 44885 → 13000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
  15 12.012414 127.0.0.1  127.0.0.1    TCP    310 44885 → 13000 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=113
  16 12.012430 127.0.0.1  127.0.0.1    TCP     84 13000 → 44885 [ACK] Seq=1 Ack=114 Win=525568 Len=0
  17 14.447182 127.0.0.1  127.0.0.1    TCP    108 44886 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PE…
  18 14.447229 127.0.0.1  127.0.0.1    TCP    108 12000 → 44886 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=…
  19 14.447309 127.0.0.1  127.0.0.1    TCP     84 44886 → 12000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
  20 14.447461 127.0.0.1  127.0.0.1    TCP    310 44886 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=113
  21 14.447479 127.0.0.1  127.0.0.1    TCP     84 12000 → 44886 [ACK] Seq=1 Ack=114 Win=525568 Len=0
  22 14.448082 127.0.0.1  127.0.0.1    TCP    108 44887 → 14000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PE…

> Frame 10: 310 bytes on wire (2480 bits), 157 bytes captured (1256 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 44884, Dst Port: 12000, Seq: 1, Ack: 1, Len: 113
v Data (113 bytes)
    Data: 38303030303030313030363524354137434232241392d3036...
    Text: 800000010065$5A7CB2A9-062A-AD03-68A5-634F6CDDB833\001
    [Length: 113]

Text (data.text), 113 bytes    Packets: 147 · Displayed: 147 (100.0%) · Marked: 1 (0.7%) · Load time: 0:0.6    Profile: Default
```

Source: Rodrigues and Branco (2020).

Figure 49 – Message USER_MESSAGE (code 0x00000002) with telemetry data sent encrypted.


Wireshark_localhost.pcapng

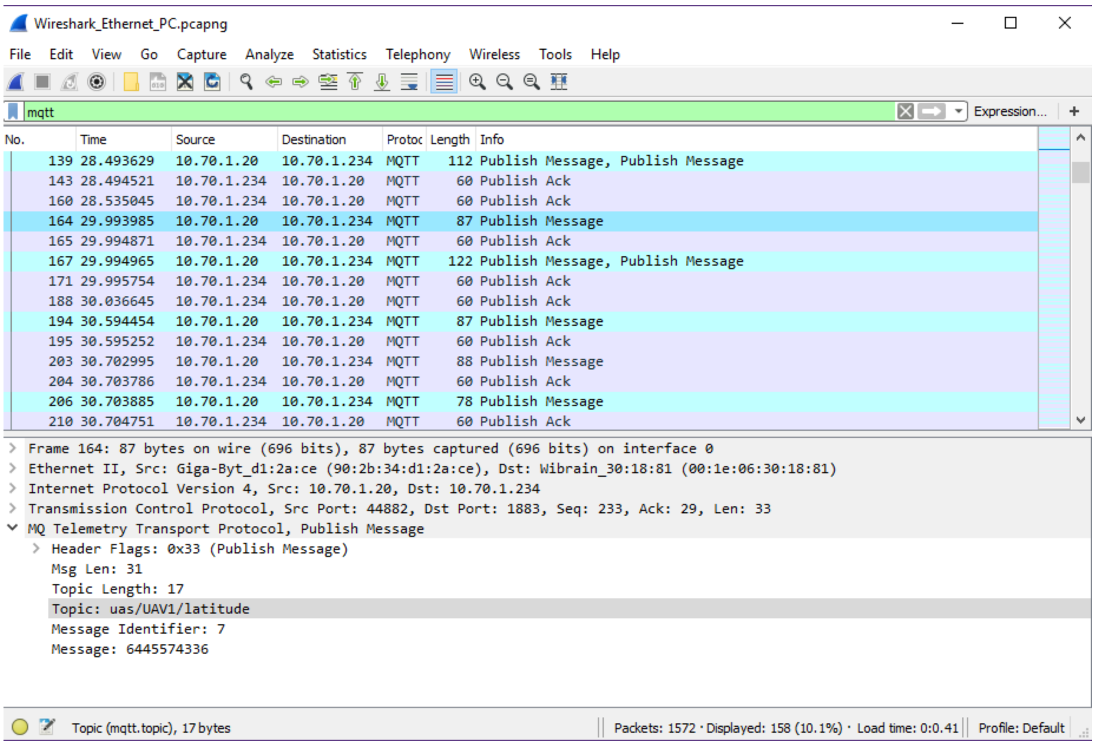
```
No.  Time      Source     Destination  Protoc Length Info
  19 14.447309 127.0.0.1  127.0.0.1    TCP     84 44886 → 12000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
  20 14.447461 127.0.0.1  127.0.0.1    TCP    310 44886 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=113
  21 14.447479 127.0.0.1  127.0.0.1    TCP     84 12000 → 44886 [ACK] Seq=1 Ack=114 Win=525568 Len=0
  22 14.448082 127.0.0.1  127.0.0.1    TCP    108 44887 → 14000 [SYN] Seq=0 Win=64240 Len=0 MSS=65495 WS=256 SACK_PE…
  23 14.448112 127.0.0.1  127.0.0.1    TCP    108 14000 → 44887 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=…
  24 14.448157 127.0.0.1  127.0.0.1    TCP     84 44887 → 14000 [ACK] Seq=1 Ack=1 Win=525568 Len=0
  25 14.448237 127.0.0.1  127.0.0.1    TCP    310 44887 → 14000 [PSH, ACK] Seq=1 Ack=1 Win=525568 Len=113
  26 14.448251 127.0.0.1  127.0.0.1    TCP     84 14000 → 44887 [ACK] Seq=1 Ack=114 Win=525568 Len=0
  27 16.015351 127.0.0.1  127.0.0.1    TCP    364 44884 → 12000 [PSH, ACK] Seq=114 Ack=1 Win=525568 Len=140
  28 16.015385 127.0.0.1  127.0.0.1    TCP     84 12000 → 44884 [ACK] Seq=1 Ack=254 Win=525312 Len=0
  29 16.735693 127.0.0.1  127.0.0.1    TCP    228 44885 → 13000 [PSH, ACK] Seq=114 Ack=1 Win=525568 Len=72
  30 16.735735 127.0.0.1  127.0.0.1    TCP     84 13000 → 44885 [ACK] Seq=1 Ack=186 Win=525312 Len=0
  31 17.450648 127.0.0.1  127.0.0.1    TCP    364 44886 → 12000 [PSH, ACK] Seq=114 Ack=1 Win=525568 Len=140
  32 17.450693 127.0.0.1  127.0.0.1    TCP     84 12000 → 44886 [ACK] Seq=1 Ack=254 Win=525312 Len=0
  33 17.950185 127.0.0.1  127.0.0.1    TCP    228 44887 → 14000 [PSH, ACK] Seq=114 Ack=1 Win=525568 Len=72
  34 17.950227 127.0.0.1  127.0.0.1    TCP     84 14000 → 44887 [ACK] Seq=1 Ack=186 Win=525312 Len=0
  35 24.018918 127.0.0.1  127.0.0.1    TCP    228 44884 → 12000 [PSH, ACK] Seq=254 Ack=1 Win=525568 Len=72
  36 24.018958 127.0.0.1  127.0.0.1    TCP     84 12000 → 44884 [ACK] Seq=1 Ack=326 Win=525312 Len=0

> Frame 27: 364 bytes on wire (2912 bits), 184 bytes captured (1472 bits) on interface 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 44884, Dst Port: 12000, Seq: 114, Ack: 1, Len: 140
v Data (140 bytes)
    Data: 3100000008000000001000000e62cecd43546be5ca851e378...
    Text: 1
    [Length: 140]

Data (data.data), 140 bytes    Packets: 147 · Displayed: 147 (100.0%) · Marked: 1 (0.7%) · Load time: 0:0.6    Profile: Default
```

Source: Rodrigues and Branco (2020).

on a criteria proposed by (SINGH; VERMA, 2017):

$$D = \frac{H + S + PA}{12}$$

Figure 50 – MQTT messages sent without any security.


Wireshark_Ethernet_PC.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
mqtt
No.  Time      Source       Destination  Protoc Length Info
 139 28.493629 10.70.1.20   10.70.1.234  MQTT   112 Publish Message, Publish Message
 143 28.494521 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 160 28.535045 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 164 29.993985 10.70.1.20   10.70.1.234  MQTT    87 Publish Message
 165 29.994871 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 167 29.994965 10.70.1.20   10.70.1.234  MQTT   122 Publish Message, Publish Message
 171 29.995754 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 188 30.036645 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 194 30.594454 10.70.1.20   10.70.1.234  MQTT    87 Publish Message
 195 30.595252 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 203 30.702995 10.70.1.20   10.70.1.234  MQTT    88 Publish Message
 204 30.703786 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack
 206 30.703885 10.70.1.20   10.70.1.234  MQTT    78 Publish Message
 210 30.704751 10.70.1.234  10.70.1.20   MQTT    60 Publish Ack

> Frame 164: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
> Ethernet II, Src: Giga-Byt_d1:2a:ce (90:2b:34:d1:2a:ce), Dst: Wibrain_30:18:81 (00:1e:06:30:18:81)
> Internet Protocol Version 4, Src: 10.70.1.20, Dst: 10.70.1.234
> Transmission Control Protocol, Src Port: 44882, Dst Port: 1883, Seq: 233, Ack: 29, Len: 33
v MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x33 (Publish Message)
    Msg Len: 31
    Topic Length: 17
    Topic: uas/UAV1/latitude
    Message Identifier: 7
    Message: 6445574336

Topic (mqtt.topic), 17 bytes    Packets: 1572 · Displayed: 158 (10.1%) · Load time: 0:0.41    Profile: Default
```

Source: Rodrigues and Branco (2020).

Table 8 – Difficulty criteria proposed by (SINGH; VERMA, 2017).

Required Hardware (H)	1	No extra hardware required
	2	Basic hardware other than PC/Laptop
	4	Advance hardware requirement
Required Skills (S)	1	No expert knowledge required
	2	Tools are available in the public domain
	4	Advanced skills needed
Physical Access (PA)	1	Physical access not required
	4	Physical access required

Source: Rodrigues, Pigatto and Branco (2020).

where H represents the type of hardware required for the attack, S the level of skill and PA if physical access is needed. The difficulty D results in a value from 0.25 (Easy) to 1 (Hard). The criteria proposed in (SINGH; VERMA, 2017) are reproduced in Table 8. The likelihood of an attack to happen will be inversely proportional to its difficulty.

The threat impact (TI) on the system is also classified according to the criteria used by (JAVAID *et al.*, 2012), with values Low (Very limited systems outages), Medium (Limited systems outages) and High (Long time systems outages).

Finally, the countermeasure impact (CI) for this study case (asymmetric cryptography and component/UAV mutual authentication) also needs to be assessed. Countermeasure impact can vary from 0 (none) to 3 (high).

Once those values are made, the threat risk r is calculated by

$$r = \frac{D*(1-TI)}{2.25*CI},$$

whose values are classified according to Table 9. The 2.25 value in denominator is used to normalize the risk values to the interval $[0,1]$.

Table 9 – Risk classification of threats.

$r < 0.20$	Very low risk
$0.2 \leq r < 0.4$	Low risk
$0.4 \leq r < 0.6$	Medium risk
$0.6 \leq r < 0.8$	High risk
$r > 0.8$	Very high risk

Source: Rodrigues, Pigatto and Branco (2020).

The results for this assessment are shown in Table 10, from which is noted that the most dangerous situations for these UAVs will be when a Jamming or External Spoofing attacks happens. In this case, security countermeasures have very low impact, and the UAVs need to take drastic measures if a critical sensor is being spoofed or if communication is jammed.

Table 10 – External context security index initial assessment.

Threat	Difficulty (D)	Threat Impact (TI)	Countermeasure Impact (CI)	Risk
Side-channel attack	Hard (0.83)	High (3)	None (0)	Low (0.22)
Code Modification	Hard (0.83)	High (3)	Low (1)	Low (0.22)
Code Injection	Hard (1)	High (3)	Low (1)	Very low (0)
Packet Sniffing	Easy (0.33)	Medium (2)	High (3)	Very low (0.19)
Packet Fuzzing	Medium (0.58)	High (3)	Medium (2)	Low (0.28)
Signal Spoofing	Easy (0.25)	High (3)	Low (1)	High (1)
Jamming	Easy (0.33)	High (3)	Low (1)	Very High (0.89)

Source: Rodrigues, Pigatto and Branco (2020).

Once the evaluation is made, it is possible to estimate the PSI external component ψ_{ex}, which is given by:

$$\psi_{ex} = 5*(1-\bar{r}),$$

which in this case will evaluate to $\psi_{ex} = 2.99$.

6.4.2 Mission Context Initial Evaluation

The mission context security index (ψ_{mi}) is evaluated analyzing the steps the UAV needs to perform in order to finish its mission. UAVs are specially critical since they present a risk of collision to other aircraft, manned or unmanned, called *air risk* and a risk of failure that can result in a free fall and bring injury to people of infrastructure, called *ground risk*. Therefore, mission location and the path planned for its execution can be quite critical.

There are many ground risk models for UAVs, as surveyed by (WASHINGTON; CLOTHIER; SILVA, 2017). Also, the Joint Authorities for Rulemaking of Unmanned Systems (JARUS)[1] is a worldwide organization that aims at providing a unique set of technical, safety and operational requirements for Remotely Piloted Aerial Systems (RPAS). They have produced a guideline for Specific Operations Risk Assessment (SORA) that has been used to assess risk in some works in the literature.

For this demonstration, the context security index is initially estimated by using a much simpler approach, which can be considered a simplification of SORA. UAV size, path, mission target and airspace operation are taken into account, as shown by Table 11.

Table 11 – Proposed ground and air risk assessment based on SORA.

LOCATION GROUND RISK

Operation Scenario	UAV size			
	$< 1m$	$1-3m$	$3-8m$	$> 8m$
Controlled area	Low	Low	Low	Medium
VLOS/underpopulated	Low	Low	Medium	Medium
BVLOS/underpopulated	Low	Medium	Medium	High
VLOS/populated	Medium	Medium	High	High
BVLOS/populated	Medium	High	High	High

AIR RISK FOR ATYPICAL AIRSPACE

Operation Scenario	Risk
Controlled area	Low (1)
VLOS, underpopulated	Low (1)
BVLOS, underpopulated	Medium (2)
VLOS, populated	Medium (2)
BVLOS, populated	High (3)

Source: Rodrigues, Pigatto and Branco (2020).

Similarly, the PSI mission component is given by the multiplicative inverse of the

[1] www.jarus-rpas.org/

normalized risk, proportionally to the maximum PSI level:

$$\psi_{mi} = 5 * (1 - \hat{r}),$$

which in this case will evaluate to $\psi_{mi} = 5 * (1 - (1/3)) = 3.33$.

6.4.3 Internal Context Initial Evaluation

As described, Internal Context is dealt by HAMSTER' NCI platform.

The scenario is analyzed by defining the each module's *NCIm* and then the *NCIe* for the UAV. According to the formulae and definitions of *NCIm* (PIGATTO, 2017), for each module in normal functioning are assumed as shown in Table 12.

Table 12 – *NCIm* for each module of a UAV.

Module	$NCIm^{sec}$			$NCIm^{saf}$			$NCIm$
	$storedData$	$temporaryData$	total	$health$	$priority$	total	
GPS	0	0.3	0.3	0	0.5	0.25	0.275
IMU	0	0.3	0.3	0	1	0.5	0.4
Camera	0.5	0	0.5	0	0	0	0.25
Autopilot	0.3	0.3	0.3	0	1	0.5	0.4
Motor	0	0	0	0	0.5	0.25	0.125
Servomotor1	0	0	0	0	1	0.5	0.25
Servomotor2	0	0	0	0	1	0.5	0.25
Radio transmitter/receiver	0	0.3	0.3	0	0.3	0.15	0.225

Source: Rodrigues, Pigatto and Branco (2020).

Considering that sensors (GPS and IMU), actuators (motor and servomotors) and the radio transmitter/receiver do not store data, *storedData* is set to 0. The camera stores images of the overflown region to identify assets and vulnerable areas, which leads to a score of 0.5 for *storedData*. The autopilot stores information about the positioning of the aircraft when pictures are taken. This module's *storedData* is set to 0.3 due to the importance of stored information. Although the GPS log is important for the mission, it is not as important as acquired images, which justifies the difference in scores between these modules.

GPS, IMU, autopilot and radio transmitter/receiver manipulate data related to the aircraft positioning, thus *temporaryData* is set to 0.3. Remaining modules deal with no data that could be considered risky for the UAV, being set to 0 on *temporaryData*. Regarding health, in a normal operation, all modules are properly working, thus *health* is set to 0.

The most critical modules for a proper functioning are IMU, autopilot and the servomotors. These modules are set to the highest value for *priority*, 1. GPS and motor's *priority* score are set to 0.5, because it is still possible to land the UAV even if any of these modules fails. The radio transmitter/receiver is not necessary to the accomplishment of the task. However, if the

UAV is forced to land, it is necessary to establish a communication via radio in order to locate and rescue the UAV, which justifies its value of 0.3 for *priority*. Finally, regarding camera's *priority*, it is set to 0 because if it fails, the UAV can safely go back home.

The definition of entities' *worth* measure is dependent on its cost. In this case, *worth* is considered as 1. The variable *field* is set to 0 due to the fact that the covered area is a crop and presents no risk to the environment or people in case of an accident. The *accomplishment* is set to 0 since the mission can be restarted at any time and a deadline was not specified. Indeed, the *NCIe* is 0.345.

Similarly, the PSI internal component is given by the multiplicative inverse of the normalized risk, proportionally to the maximum PSI level:

$$\psi_{in} = 5 * (1 - NCIe),$$

which in this case will evaluate to $\psi_{in} = 5 * (1 - 0.345) = 3.27$.

6.4.4 PSI Initial Evaluation

Given the three estimations, the initial Perceived Security Index value is given by

$$\Psi = \left\lfloor \sqrt{\frac{\psi_{ex}^2 + \psi_{mi}^2 + \psi_{in}^2}{3}} \right\rceil =$$

$$= \left\lfloor \sqrt{\frac{2.99^2 + 3.33^2 + 3.27^2}{3}} \right\rceil = \lfloor 3.2 \rceil$$

which is classified as 3 – Medium security risk according to Table 3.

6.4.5 Context changes impact on PSI

In this Section, it is discussed how HAMSTER deals with security context changes. For that, two situations are discussed: a GPS failure due to malfunction and the same failure caused by a spoofing attack.

First of all, HAMSTER SMU platform inside Cloud–SPHERE detects a non-expected change in GPS reading. This detection is informed to NCI, which updates the *health* component from GPS' $NCIm^{saf}$ index from 0 (normal) to 1 (experiencing issues). The GPS NCI then changes from 0.275 to 0.442. This results in a UAV NCI of 0.426.

As a consequence, the NCI modification impacts the PSI's internal context component, which evaluates to

$$\psi_{in}' = 5 * (1 - NCIe) = 2.79,$$

which in turn will impact global PSI with a new value of Ψ'=3.04, which is still evaluated as Medium Security.

The change in NCIm value results in the module being considered unreliable, as are all SEMU services related to it.If another module inside the UAV or an external client is using those services, they will have to query new providers.

Besides the GPS malfunction, if the SMU or another IDS-enabled component detects a spoofing attack, not only will the internal context change, but also the external context. In this case, the external risk is overruled to the normalized value of threat impact (TI in Table 9), if this value is higher than the current risk evaluation.

In this case, the TI normalized value for external spoofing attack is 1, and therefore the new value for PSI external component will be given by:

$$\psi'_{ex} = 5 * (1 - 1) = 0$$

The new PSI value in case of spoofing attack considers the value changes from both external and internal context, being evaluated to:

$$\Psi' = \left\lfloor \sqrt{\frac{\psi'^{\,2}_{ex} + \psi^{2}_{mi} + \psi'^{\,2}_{in}}{3}} \right\rceil =$$

$$= \left\lfloor \sqrt{\frac{0^2 + 3.33^2 + 2.79^2}{3}} \right\rceil = \lfloor 2.5 \rceil,$$

which is conservatively evaluated to 2 - Low Security.

In HAMSTER applications, as previously discussed on Section 4.4.4, there can be a minimum PSI level for the UAV to operate. In this case, if the PSI level was set to Medium Security, the detection of the spoofing attack would trigger the execution of an emergency protocol defined by the UAV application and not HAMSTER itself. In this situation, the UAV can be commanded to turn off all non-critical modules, interrupt services which have low security mechanisms applied to them, or return to base immediately.

It is important to point out that any component failure will increase the UAV NCI and therefore decrease ψ_{in}, making the UAV perceive a more problematic environment.Likewise, perceiving an attack indicator will decrease ψ_{ex}, signaling to the UV that the external environment is now more hostile.

6.5 PoC 5: UAV Delivery Service

This PoC was published in (RODRIGUES; BRANCO, 2021) and demonstrates how HAMSTER services managed by SEMU can be used in a hypothetical UAV-enabled delivery application that follow the steps:

1. The company receives a delivery request from a client;

2. The package is loaded in the UAV by a company employee;
3. The UAV navigates to the client house and hovers;
4. The client is informed the delivery has arrived and authorizes the UAV descent to the position;
5. The client retrieves the package and confirms the delivery to the company;
6. The company call the UAV back to its base.

Since the goal of this scenario is to demonstrate service exchange in HAMSTER, there are some limitations and assumptions:

- All elements in the system know the necessary network addresses beforehand;
- There is only one UAV available for delivery;
- The UAV navigates in a xyz space and in straight lines;
- The UAV stays in mid-height ($z = 5$) for package load and retrieval and navigates at a fixed height ($z = 10$);
- There are no checks or verification regarding security.

6.5.1 Environmental Setup

There are four components in this demonstration: (1) the client, (2) the company Ground Control Station (GCS), (3) the UAV and (4) the UAV navigation module, responsible for navigation. HAMSTER is deployed in the later three, with Unit Types and registered services shown by Table 13.

The client and the company have different communication messages through another connection outside HAMSTER, with the messages illustrated by Figure 51.

When the client requests a product delivery, the message REQUEST_DELIVERY is sent to the GCS, which in turn requests the UAV to ascend to a mid-height position for handling the package. Once the package is loaded, the GCS sends the message DELIVERY_STARTED to the client and commands the UAV to the client's house. Once the GCS is notified that the UAV has arrived at its destination, it sends the message PACKAGE_POSITION to the client, signalizing that the UAV has arrived and need authorization to descent so the package can be retrieved. The client sends the message AUTHORIZE_DESCENT to the GCS and the UAV descends to the mid-height position. Once the client retrieves the package, the message PACKAGE_RETRIEVED is sent to the GCS, which in turns calls the UAV home.

6.5.2 Obtained Results

Here, the logs from the UAV and the GCS are presented, showing the communication between them and between the GCS with the UAV. For readability reasons, some lines were suppressed.

Table 13 – Unit Types and services registered for each Component in the demonstration.

Component	Unit Type	Service Name	Service Type	Description
Client	Not deploying HAMSTER.			
GCS	SUPER ENTITY	Not providing services.		
UAV	ENTITY	`ProdHandle`	`COMMAND`	Request the UAV to go to Package Handling position (mid-height).
		`PkgReady`	`EVENT`	Notifies that the UAV is in Package Handling position.
		`NavigateTo`	`COMMAND`	Request the UAV to navigate to a specific waypoint.
		`InPosition`	`EVENT`	Notifies the UAV has reached the navigation point from previous `NavigateTo` request.
		`Land`	`COMMAND`	Request the UAV to land.
		`Landed`	`EVENT`	Notifies the UAV has landed.
Navigation	MODULE	`NavXPosition`	`DATA`	Current X position.
		`NavYPosition`	`DATA`	Current Y position.
		`NavZPosition`	`DATA`	Current Z position.
		`HoverCmd`	`COMMAND`	Request the navigation to Hover.
		`NavigateCmd`	`COMMAND`	Request the navigation to go to a specific waypoint.
		`Position Achieved`	`EVENT`	Notifies the navigation has reached the waypoint from previous `NavigateCmd` request.
		`StopCmd`	`COMMAND`	Tells the navigation to stop.

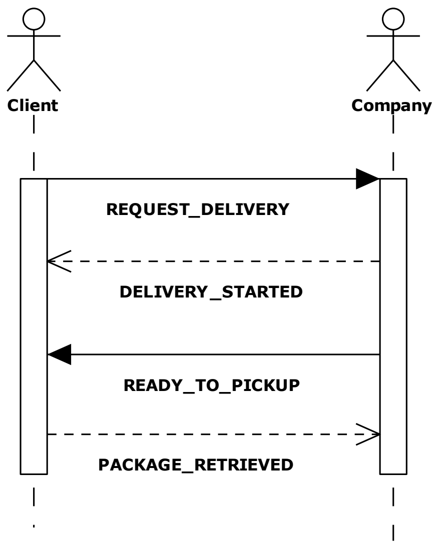
Source: Elaborated by the author.

In client log (Figure 52), it is possible to see the order in which the client messages (without tags) and the internal messages (with `[App]` tag) are exchanged. The communication illustrates the application client side and demonstrates the protocol described in Figure 51.

The GCS log (Figure 53) illustrates the server side of the application and presents the SEO-related messages exchanged with the UAV (Peer ID = 1) for service consumption. In the log, the GCS successfully requests three event services from the UAV (Lines 14-16): `PkgReady`, `InPosition` and `Landed`.

After receiving the delivery request (Line 17), the GCS commands the UAV to a mid-height (command service `ProdHandle`, line 18). When the position is achieved and the package is loaded (Lines 19-20), the GCS commands the UAV to its destination (command service

Figure 51 – Messages exchanged between the client and the company.



Source: Rodrigues and Branco (2021).

Figure 52 – Client log messages

```
01 Welcome to HAMSTER Delivery!
02 Press any key to request a delivery...
03 [  App  ] Message REQUEST_DELIVERY sent
04 Deliver request Successful. Wait for confirmation.
05 [  App  ] Received DELIVERY_STARTED message from server.
06 Yay! Your package is on your way!
07 [  App  ] Received PACKAGE_POSITION message from server.
08 Your package arrived! Press any key to authorize descent.
09 [  App  ] Message AUTHORIZE_DESCENT sent
10 [  App  ] Received PICKUP_READY message from server.
11 Your package is in position!
12 Press any key after picking it up.
13 [  App  ] Message PACKAGE_RETRIEVED sent
14 Thank you for using our services!
```

Source: Rodrigues and Branco (2021).

`NavigateTo`, line 21) and later receives the UAV confirmation that the position was achieved (Line 22). After the client authorizes the UAV descent (Lines 23-24), the GCS again commands the UAV to a mid-height so the client can pick up the package (Lines 25-26). Once this happens (Line 27), the GCS commands the UAV to its home coordinates (Lines 28-29) and then to land (Lines 30-31), which finalizes the delivery application.

UAV log (Figure 54) illustrates the service requests received by the UAV from its CGS (Peer ID = 5, lines 38-40). Upon client delivery request, the CGS requests the UAV to receive the package (lines 41-42). The UAV redirects the command to its navigation unit (Peer ID = 3, line 43) and notifies the GCS when it is ready to receive the package (lines 50-51). At this

Figure 53 – GCS log messages.

```
(...)
14 [Event  ] Message SERVICE_REQUEST_OK for Service ID PkgReady received in SEO from peer 1.
15 [Event  ] Message SERVICE_REQUEST_OK for Service ID InPosition received in SEO from peer 1.
16 [Event  ] Message SERVICE_REQUEST_OK for Service ID Landed received in SEO from peer 1.
17 [  App  ] Received REQUEST_DELIVERY message from client.
18 [Event  ] Message SERVICE_REQUEST_OK for Service ID ProdHandle received in SEO from peer 1.
19 [Event  ] Message SERVICE_NOTIFICATION for Service ID PkgReady received in SEO from peer 1.
20 [  App  ] Drone ready for package. Enter any key when loading is done...
21 [Event  ] Message SERVICE_REQUEST_OK for Service ID NavigateTo received in SEO from peer 1.
22 [Event  ] Message SERVICE_NOTIFICATION for Service ID InPosition received in SEO from peer 1.
23 [  App  ] Drone arrived at client. Wait for consumer authorization for descent.
24 [  App  ] Received AUTHORIZE_DESCENT message from client.
25 [Event  ] Message SERVICE_REQUEST_OK for Service ID ProdHandle received in SEO from peer 1.
26 [Event  ] Message SERVICE_NOTIFICATION for Service ID PkgReady received in SEO from peer 1.
27 [  App  ] Received PACKAGE_RETRIEVED message from client.
28 [Event  ] Message SERVICE_REQUEST_OK for Service ID NavigateTo received in SEO from peer 1.
29 [Event  ] Message SERVICE_NOTIFICATION for Service ID InPosition received in SEO from peer 1.
30 [Event  ] Message SERVICE_REQUEST_OK for Service ID Land received in SEO from peer 1.
31 [Event  ] Message SERVICE_NOTIFICATION for Service ID Landed received in SEO from peer 1.
```

Source: Rodrigues and Branco (2021).

point, the UAV receives the `NavigateTo` service request (lines 52-54) and starts navigating to the client location. Upon arrival in the client's house and with their authorization to descent with the package, the UAV receives another `ProdHandle` request (lines 63-66). After the product is picked up, the UAV is commanded to return home (lines 73-77) and land (lines 85-95), finishing its delivery mission.

6.6 PoC 6: UAV Data Gathering Service

This PoC is present in (RODRIGUES; BRANCO, 2022) and demonstrates a UAV-enabled data collection service. Here, the UAV mission is to collect data stored in HAMSTER-enabled, standalone sensors which are located in a remote area. The SEMU Platform is used to control the UAV and communicate with the Cloud and the Sensors through services; the NP Platform is used to manage the UAV task and the ON/OFF state of its modules; finally, the *Cloud*–SPHERE Platform provides authentication and cryptography.

In this UAV-enabled service, the Cloud Server requests the UAV to start the collection task; the UAV in turn flights to three different sensors, connects and authenticates with them through a special internal component and gather the stored sensor data.

It is not the objective of this demonstration to perform any security or performance test on the demonstration, since both of them are greatly impacted by the application aspects such as the security algorithms used, service complexity and communication technology latency. The objective of this scenario is to provide a working example to demonstrate how HAMSTER can be used to enable secure services to the IoT environment with a UAV-enabled service scenario.

Figure 54 – UAV log messages.

```
(...)
38 [Event  ] Message SERVICE_REQUEST for Service ID PkgReady received in SEO from peer 5.
39 [Event  ] Message SERVICE_REQUEST for Service ID InPosition received in SEO from peer 5.
40 [Event  ] Message SERVICE_REQUEST for Service ID Landed received in SEO from peer 5.
41 [Event  ] Message SERVICE_REQUEST for Service ID ProdHandle received in SEO from peer 5.
42 [  App  ] Package Handling requested.
43 [Event  ] Message SERVICE_REQUEST sent to Peer ID 3
(...)
50 [Event  ] Message SERVICE_NOTIFICATION sent to Peer ID 5
51 [  App  ] UAV ready for package.
52 [Event  ] Message SERVICE_REQUEST for Service ID NavigateTo received in SEO from peer 5.
53 [Event  ] Message SERVICE_REQUEST sent to Peer ID 3
54 [  App  ] UAV navigating...
(...)
63 [  App  ] UAV hovering.
64 [Event  ] Message SERVICE_NOTIFICATION sent to Peer ID 5
65 [Event  ] Message SERVICE_REQUEST for Service ID ProdHandle received in SEO from peer 5.
66 [  App  ] Package Handling requested.
(...)
73 [Event  ] Message SERVICE_NOTIFICATION sent to Peer ID 5
74 [  App  ] UAV ready for package.
75 [Event  ] Message SERVICE_REQUEST for Service ID NavigateTo received in SEO from peer 5.
76 [Event  ] Message SERVICE_REQUEST sent to Peer ID 3
77 [  App  ] UAV navigating...
(...)
85 [  App  ] UAV hovering.
86 [Event  ] Message SERVICE_NOTIFICATION sent to Peer ID 5
87 [Event  ] Message SERVICE_REQUEST for Service ID Land received in SEO from peer 5.
88 [Event  ] Message SERVICE_REQUEST sent to Peer ID 3
89 [  App  ] UAV descending...
(...)
95 [Event  ] Message SERVICE_NOTIFICATION sent to Peer ID 5
96 [  App  ] UAV has landed.
```

Source: Rodrigues and Branco (2021).

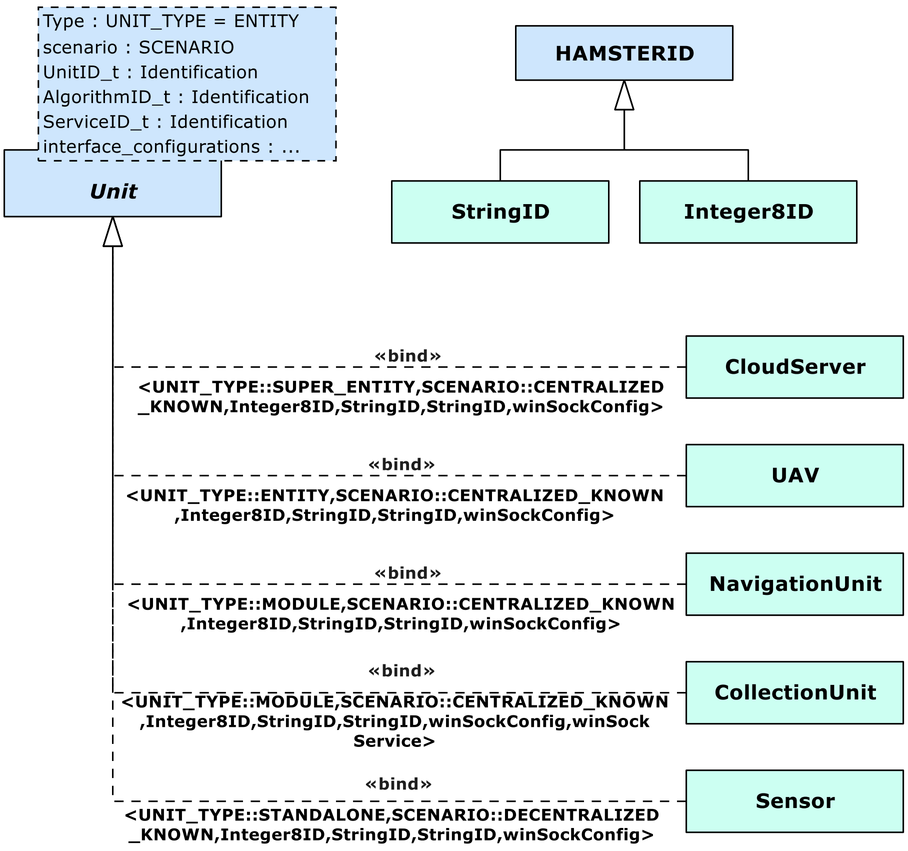
6.6.1 Environment Setup

In this demonstration, there are different HAMSTER Units: (1) a Cloud Server that can either manage UAVs autonomously or accept external client requests; (2) a UAV that will perform the task; (3) a UAV-internal Navigation Unit to control the vehicle's navigation; (4) a UAV-internal Collection Unit which connects to external sensors and collect their data; (5) three standalone Sensor objects that gathers data which is later collected by the UAV.

6.6.1.1 HAMSTER Units

Figure 55 brings the modeling used for the Units present in the demonstration. Based on the template parameters from **`Unit`** class (Figure 11), all Units are in a *Known formation*, with class **`Sensor`** being the only one in a decentralized scenario and all other Units form a centralized scenario: the **`UAV`** is the central Unit of both **`Navigation Unit`** and **`Collection Unit`**, while the **`CloudServer`** is the central Unit of the UAV. The Units are identified by class **`Unit8ID`** (an 8-bit integer) and services and algorithms are identified by strings represented

Figure 55 – Units present in the demonstration.


Type : UNIT_TYPE = ENTITY
scenario : SCENARIO
UnitID_t : Identification
AlgorithmID_t : Identification
ServiceID_t : Identification
interface_configurations : ...
Unit
HAMSTERID
StringID
Integer8ID
«bind»
<UNIT_TYPE::SUPER_ENTITY,SCENARIO::CENTRALIZED_KNOWN,Integer8ID,StringID,StringID,winSockConfig>
CloudServer
«bind»
<UNIT_TYPE::ENTITY,SCENARIO::CENTRALIZED_KNOWN,Integer8ID,StringID,StringID,winSockConfig>
UAV
«bind»
<UNIT_TYPE::MODULE,SCENARIO::CENTRALIZED_KNOWN,Integer8ID,StringID,StringID,winSockConfig>
NavigationUnit
«bind»
<UNIT_TYPE::MODULE,SCENARIO::CENTRALIZED_KNOWN,Integer8ID,StringID,StringID,winSockConfig,winSockService>
CollectionUnit
«bind»
<UNIT_TYPE::STANDALONE,SCENARIO::DECENTRALIZED_KNOWN,Integer8ID,StringID,StringID,winSockConfig>
Sensor

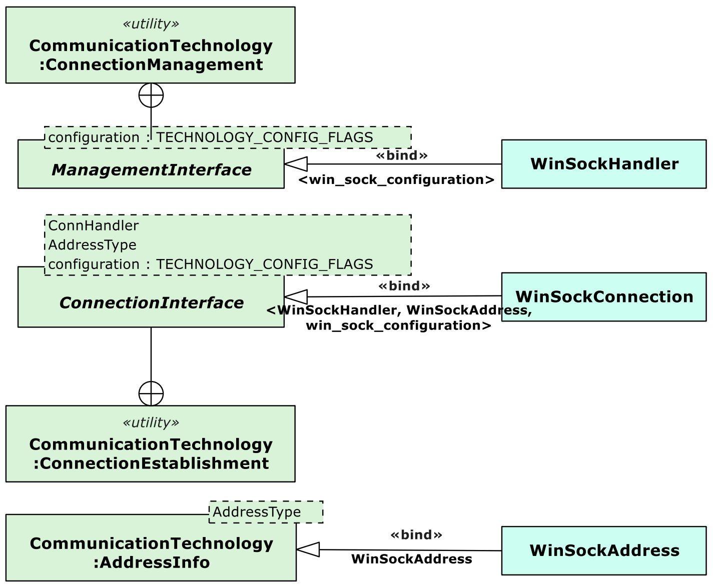
Source: Rodrigues and Branco (2022).

by class `StringID`. Finally, all Units communicate through the TCP/IP protocol, which is an Operation System specific implementation. In this demonstration, the Windows implementation of sockets is used. It is notable that the `CollectionUnit` class has two different configurations, meaning the Unit has two different communication interfaces: one for internal communication and other exclusively for external communication.

6.6.1.2 Unit Communication

In this scenario, all Units communicate through Windows sockets and use only the Service Gate (no Application messages are exchanged). The communication technology implementation is done through three different classes, as shown by Figure 56. The technology is configured so (1) the Acceptor (the server) has an address and (2) the Acceptor executes a loop (so the server is asynchronous). The Connector (the client) do not need an address or a loop not before or after the connection is established.

Figure 56 – Classes implementing Windows sockets in the demonstration.


«utility»
CommunicationTechnology :ConnectionManagement
configuration : TECHNOLOGY_CONFIG_FLAGS
ManagementInterface
«bind»
<win_sock_configuration>
WinSockHandler
ConnHandler
AddressType
configuration : TECHNOLOGY_CONFIG_FLAGS
ConnectionInterface
«bind»
<WinSockHandler, WinSockAddress, win_sock_configuration>
WinSockConnection
«utility»
CommunicationTechnology :ConnectionEstablishment
AddressType
CommunicationTechnology :AddressInfo
«bind»
WinSockAddress
WinSockAddress

Source: Rodrigues and Branco (2022).

6.6.1.3 Services

The entire scenario is controlled by the requisition and consumption of services between the Units. In Table 14 the envisioned services for the Units are presented. These services are created in the Application Layer, i.e., they are specific for this demonstration. All applications can create the services deemed necessary.

Table 14 – Services present in the demonstration.

Unit	Service	Service Type
UAV	START_MISSION	Command
UAV	MISSION_FINISHED	Event
NavigationUnit	X_POSITION	Data
NavigationUnit	Y_POSITION	Data
NavigationUnit	Z_POSITION	Data
NavigationUnit	NAVIGATE	Command
NavigationUnit	POSITION_ACHIEVED	Event
CollectionUnit	START_COLLECTION	Command
CollectionUnit	COLLECTION_RESULT	Event
Sensor	COLLECT_DATA	Command
Sensor	COLLECTION_FINISHED	Event

Source: Rodrigues and Branco (2022).

6.6.1.4 Authentication & Cryptography

HAMSTER can be used with any authentication and cryptography algorithms that follow the necessary interface presented in Section 5.4. In this scenario, it was used the same authentication algorithm used in previous work presented in (RODRIGUES *et al.*, 2019). For cryptography, a simple Caesar Cipher was used.

6.6.1.5 Task Management

The NP Platform is used to manage the UAV collection task. In this demonstration, the task states shown in Table 15 are hard-coded in the program, but could be dynamically configured through application messages or services, for instance.

Table 15 – Task states in the demonstration.

Number	Description	NavigationUnit state	CollectionUnit state
1	Flight until Sensor 1	ON	OFF
2	Collect from Sensor 1	ON	ON
3	Flight until Sensor 2	ON	OFF
4	Collect from Sensor 2	ON	ON
5	Flight until Sensor 3	ON	OFF
6	Collect from Sensor 3	ON	ON
7	Return to base	ON	OFF

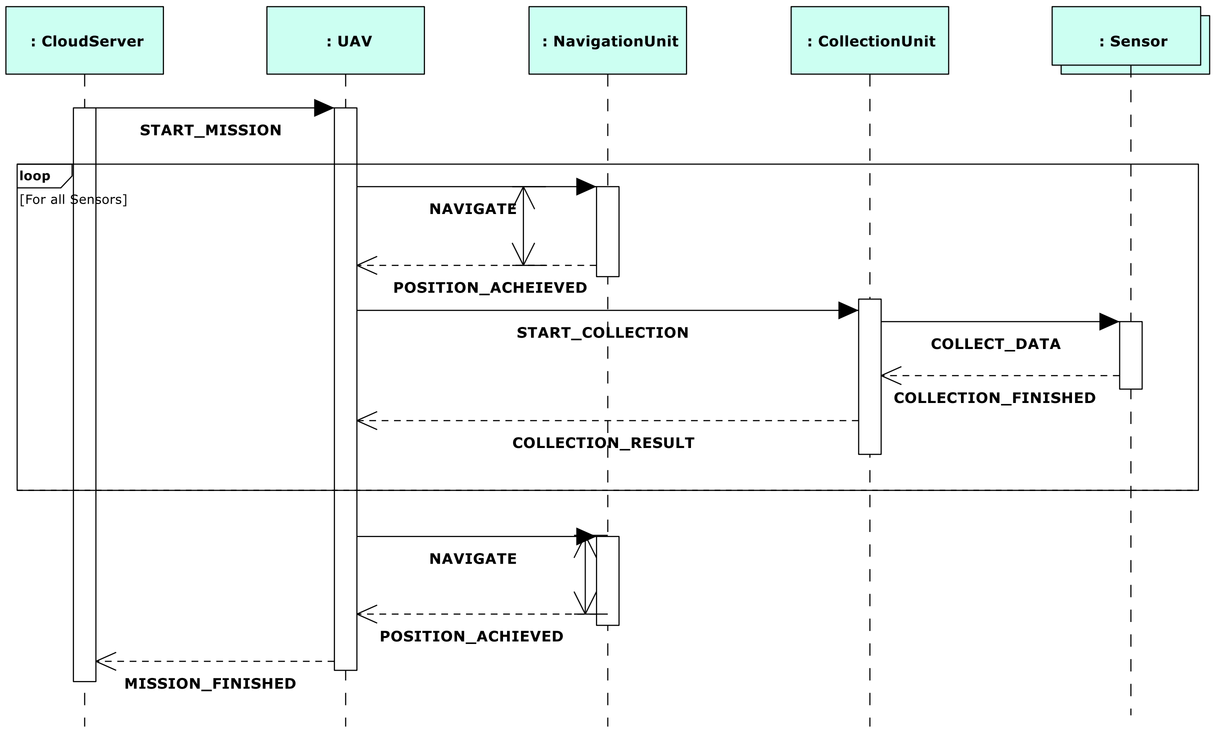
Source: Rodrigues and Branco (2022).

6.6.2 Demonstration Details

As said before, this demonstration use HAMSTER-enabled services to perform a UAV-enabled data collection service. Figure 57 brings a high-level view of the services used (continuous lines are command services and dashed lines are event services) to provide the demonstration, which can be summarized in the following steps:

1. the UAV waits for its internal components to authenticate;
2. the UAV authenticates itself with the Cloud Server;
3. the Cloud Server requests the UAV to start the mission;
4. the UAV travels to sensor 1 (12,5);
5. the UAV authenticates to sensor 1 and collects its data;
6. the UAV travels to sensor 2 (15,7);
7. the UAV authenticates to sensor 2 and collects its data;
8. the UAV travels to sensor 3 (20,12);
9. the UAV authenticates to sensor 3 and collects its data;

Figure 57 – Demonstration scenario operation based on registered services.



Source: Elaborated by the author.

10. the UAV goes back to base (0,0) and finishes its task.

6.6.3 Obtained Results

This demonstration was executed in a i5-10210U CPU laptop with 8GB RAM running a 64-bit Windows 10, with all nodes communicating through the localhost at different IP addresses. All logs are presented at the Appendix A.1 and are edited for better readability. S_n log represents the standalone sensor log, which is equivalent for all sensors.

6.6.3.1 Initialization and Service Registration

All Units must register any cryptography/authentication algorithm and services in their custom initialization. In the application, all Units register the authentication and cryptography algorithms and initialize all Peer-related data (authentication secrets and cryptography keys) and services. (lines 2-3 in UAV Log, Collection Unit Log and S_n log; lines 2-6 in Navigation Unit Log).

6.6.3.2 Connection and Authentication

In a centralized architecture, the Unit connects and authenticates itself to its central Unit. UAV Log (lines 4-12) and Cloud Server (lines 3-8) show the authentication process between those two Units, which is similar to the other Units authenticating themselves.

6.6.3.3 Service requests

Next in the demonstration, the Units request each other's services. Data and event services can be requested at the beginning of the execution, but command services must be requested at the time the application wishes to run the command supported by the service. The UAV requests the services of its modules at lines 14-18.

6.6.3.4 Task execution

The task initiates by the request of START_MISSION command service from the Cloud Server (log line 11) to the UAV (log line 19). The UAV starts the new Task State (1) and sends the CURRENT_STATE for both modules (lines 20-21), which turns on (Navigation Unit, line 13) and off (Collection Unit, line 8). After that, the UAV requests the Navigation Unit to NAVIGATE to the next sensor (line 24).

6.6.3.5 Navigation

The Navigation Unit will go from the current UAV position to the next position by moving in the Z plane until it reaches a flight height (10) and then moving the vehicle in the XY plane. When the XY position of a sensor is achieved, the UAV receives the POSITION_ACHIEVED notification and change the Task to the next State, which turns on the Collection Unit (lines 9, 18 and 27). The UAV then requests the Navigation Unit to descend until a Collection Height (z = 5,lines 41, 53 and 60) and, when the position is achieved (lines 42, 54 and 61), request the data collection to the Collection Unit (lines 43, 55 and 62).

6.6.3.6 Connection to Sensors

For every sensor, the Collection Unit authenticate itself and requests the COLLECT_DATA service (lines 14, 23 and 32). The answer from sensors comes with COLLECTION_FINISHED (lines 15, 24 and 33). Once this is received, the Collection Unit notifies the UAV the collection is complete through COLLECTION_RESULT notification.

6.6.3.7 Task finalization

When data from all services are collected, the UAV goes to the last Task State (7) and requests the Navigation Unit to go to its base. Once the (0,0,0) position is achieved, the UAV finished the task by notifying its modules (lines 65-66) and notifies the Cloud Service the task is done through event MISSION_FINISHED notification (line 13 in Cloud Server log).

6.7 Final Remarks

In this Chapter, different proof of concepts the demonstrate the functioning of HAMSTER as a whole and specially the platforms expanded in this PhD so Service Management for UAVs became possible.

As demonstrated, different real-life scenarios can benefit from this work. It is possible to make an autonomous application with task management in NP, in which services can be managed and exchanged by SEMUusing security mechanisms provided by *Cloud*–SPHERE. Not only that, already established protocols can also use *Cloud*–SPHERE to have its connection and message exchange security enhanced. Finally, the use of Standalone nodes increases the impact of this work, since it can also be applied in low-cost hardware to enable communication security and service exchange for these devices as well, making it possible to build an IoT-ready application with all nodes compatible between them and platform/protocol agnostic.

CHAPTER

7

CONCLUSION

Following the technological development and recent trends, UAVs are now being connected to other devices and systems such as the Internet of Things. One promising opportunity is the use of vehicles and sensors as services allowing a better integration of all things, leading to an environment composed by everything, updated every-time and available everywhere.

There are many challenges to be tackled due to inherent characteristics of the UAV-IoT ecosystem such as resource constrains, heterogeneity and lack of standardization.

For battery-powered devices such as UAVs and IoT sensors, energy efficiency is a key challenge. For UAVs, energy efficiency increases the vehicle autonomy, and also potentially increases the battery lifetime, reducing waste. For IoT sensors, energy efficiency might be the key requirement when deploying a system in an area with difficult human access — the least you need human intervention, the better.

How to deploy services is also a challenge. There are many different situations in which services can be provided. Applications can vary from a closed network with only known, trusted clients, to a open-for-all, IoT application that offer different services to the general public. The service exchange will have different types of data, networks and access controls.

Even though Security is a critical problem in all elements of this ecosystem, the development of security measures and solutions are being left behind when comparing to other aspects of the UAV-IoT integration. The high heterogeneity allied with the resource limitation of most of devices connected to it makes it impossible to achieve a ‘one-size-fits-all’ solution. Any efforts towards security have to take into account many different aspects and be prepared to provide a flexible and configurable solution to support many different devices.

HAMSTER was designed to be flexible, protocol-, data- and platform-agnostic to be better suited to a broader number of applications. It deals with the system heterogeneity by providing generic APIs and concrete functionality through abstractions that can be specialized according to each systems necessities. Also, four operation scenarios for HAMSTER are envisioned, which is likely to cover a broad number of IoT-enabled UAVs applications.

HAMSTER provides communication security through *Cloud*–SPHERE, service exchange support and management through SEMU and task management with NP platform, in a functional prototype to be freely available for further development and research support. With that, HAMSTER has the potential to be a great ally in securely integrating UAVs into the IoT.

Despite the challenges, the integration of UAVs into IoT is a promising research topic with high chances of applicability. The continuous development of the technology to tackle the necessary challenges such as the ones presented in this work will make possible to broaden the use of UAVs and contribute to economic development and quality of life.

7.1 Contributions

The main contribution if this work is *the extension of HAMSTER architecture so it can support **service operations** with **secure communication** so UAVs can be integrated into the Internet of Things*. To accomplish this goal, other contributions were derived:

- **The Security and safety Platform for HEteRogeneous systEms connected to the Cloud (RODRIGUES; PIGATTO; BRANCO, 2018a).** *Cloud*–SPHERE is an expansion of the original SPHERE platform that aims to provide security not only for in-system communication, but also with external system like the Cloud. The platform was expanded to include security mechanisms abstractions and security configuration APIs to be used by application designers.
- **HAMSTER Operation Scenarios (RODRIGUES; PIGATTO; BRANCO, 2018a; RODRIGUES; BRANCO, 2022).** Possible operation scenarios of Unmanned Vehicles in the IoT ecosystem were proposed and analyzed regarding security and service-provisioning aspects.
- **Task management functionality in HAMSTER (RODRIGUES; PIGATTO; BRANCO, 2018b).** The original Navigation Phases (NP) platform was expanded so it is able to not only manage the state of an Unmanned Vehicle components, but also to support autonomous task management based on different phases and its change triggers.
- **Reference Model for Service Management (RODRIGUES; BRANCO, 2019).** The reference model for Unmanned Vehicles services is protocol- and data-agnostic. It envisioned different types of service with possible distinct distribution.
- **Context-aware security (RODRIGUES; PIGATTO; BRANCO, 2020).** A context-aware security evaluation was proposed to encompass different types of contextual evaluation, effectively expanding the original NCI platform that considered only internal context information to evaluate risk. Contextual information is translated to a Perceived Security Index that can be used to take decisions on which security measures to use and to which clients to provide a service.

- **HAMSTER layered model (RODRIGUES; BRANCO, 2020).** This work modeled and organized HAMSTER in a model applicable to real applications, evolving the UML reference model from Pigatto (2017). The model was design to be platform-agnostic and to allow different types of data and algorithms to be used. Designers are able to build HAMSTER-enabled applications by using the functionality API from the Core Layer and the interfaces from the Abstraction Layer.
- **Service Management Unit (SEMU) Platform (RODRIGUES; BRANCO, 2021).** Models and supports different services operations such as registration, discovery, request and exchange so HAMSTER-enable devices can offer those services to clients.
- **The HAMSTER Prototype (RODRIGUES; BRANCO, 2022).** a functioning prototype of the entire HAMSTER architecture supporting one of the operation scenarios envisioned. The prototype is going to be released MIT-licensed, openly available for further research and development.

7.2 Limitations and Difficulties

The initial project proposed to focus on *Cloud*–SPHERE and the development of security mechanisms for Unmanned Vehicles connected to the Cloud. During the initial research, however, it became evident that the system heterogeneity would make the development of a comprehensive solution unfeasible. Therefore, a more agnostic, abstraction-based approach was necessary.

Initially, it was envisioned that SEMU would be a part of *Cloud*–SPHERE to deal with security aspects of service exchange. However, the lack of a service reference model resulted in making SEMU a separate platform that deal exclusively with service exchange. Because of that, the integration of the service management functionality with security perception is not yet made. The same happened with the envisioned scenarios, with only one of them being implemented.

An important limitation of HAMSTER model is its application to other types of vehicles. Even though the architecture was envisioned to attend other types of Unmanned Vehicles, the design was mainly modeled after Aerial Vehicles specifically, so its application on other vehicle types might not be smooth.

In this same line, the modeled abstractions were made with common UAV technologies in mind, such as Ethernet, Zigbee and known cryptography and authentication algorithms. There is no guarantee that the abstractions are going to be enough for other security algorithms or communicating technologies with different paradigms, like cellular networks (3G/4G/5G).

HAMSTER and its platforms developed in this PhD (*Cloud*–SPHERE, SEMUand NP) were validated functionally, but further tests and investigations are needed. The PSI was only conceived theoretically and had no real-life scenario validation. Communication abstraction was only implemented for Ethernet sockets, and tests were made only in Desktop environments. UAV mobility greatly impacts communication, and further studies in conditions/hardware closer to an actual UAV application are necessary. That being said, no studies on performance were made. Despite HAMSTER MESSAGES having a small payload, the impact on network/services availability, performance and QoS needs to be studied.

The prototype presented a great difficulty in the middle of the research. Due to unclear requirements because of the ongoing research, the prototype became unfeasible for resource-constrained devices, with the executable application reaching more than 3MB in size. Because of that, the entire prototype was refactored using C++ optimizations and modern functionality. Even though now the executable applications are now about 500Kb in size, the refactoring took a lot of time and work.

7.3 Future Work

Even though HAMSTER is already a very complex architecture, there are still many opportunities ahead.

When considering high-level opportunities, there are many points to be explored. Currently, only one scenario (centralized architecture with known formation) is totally implemented in the prototype. Extending HAMSTER to other operating scenarios as described in Section 4.7 implies in tackling many challenges.

Unknown formation scenarios will demand a negotiation on which algorithms to use for Cryptography and other security mechanisms. In this case, an adequate model would provide a good abstraction so different strategies may be deployed (power-aware, security-aware, etc) depending on the application. Services and available security mechanisms from a Unit can be discovered through the envisioned Broadcast Gate, but both the Gate and a protocol for this discovery needs to be implemented. Also, key management and how to certify identities are also challenges in decentralized scenarios and, as with IoT, demand constraint-aware solutions.

Centralized scenarios, specially in UAV swarms in which there is a ‘leader UAV’, have to deal with the possibility of a leadership handover. The swarm leader would probably consume more power due to extra responsibilities, and with the use of smaller UAVs to compose a swarm a leadership handover would be necessary. In this case, an adequate abstraction is also necessary so diverse techniques could be applied depending on the application context.

When considering UAV-as-a-Service (UaaS) scenarios, there are many aspects on UAV selection for a specific service and UAV substitution in case the selected vehicle is unable to finish the assigned task.

Emergency situations have also not been dealt with and need to be addressed. How HAMSTER could support a situation in which an attack or malfunction is detected is an open question.

The prototype future development also present interesting opportunities. First, adding a system in which network configuration parameters (such as network addresses) and task stages can be configured dynamically would allow the application of HAMSTER in UAV-enabled service hubs, in which the swarm that will provide the service is dynamically chosen depending on onboard sensors, battery level, etc.

Regarding security, the inclusion of other security mechanisms such as Access Control and integrity checking in *Cloud*–SPHEREand the integration with context-aware evaluation and NCI will increase HAMSTER security support.

The use of context-aware information is not yet included in the prototype. This brings challenges like (1) how is the initial PSI evaluation would be shared with the UAV, (2) develop policies for decisions to take when the PSI changes, (3) how to take into account the available security mechanisms in PSI evaluation, (4) how to switch between different security mechanisms (i.e., change the cryptography algorithm) seamlessly, (5) how to evaluate the impact of security mechanisms available on the environment, the communication links and UV survivability, among others.

The SMU Platform is still to be proper modeled and implemented in the prototype and is a great opportunity to increasing the safety of HAMSTER UNITS. There are many possibilities that can be used in this context, such as the use of Artificial Intelligence to detect a malfunction.

Al this security evaluation also need to be integrated with service management and provision. There is still no collaboration between SEMU and the security platforms so services can be conditionally provided depending on security aspects such as data sensitiveness and how much the client is trusted. Other support operations such as service discovery mechanisms on the broadcast gate are not yet modeled and implemented.

The NP platform can be expanded with other control parameters besides the ON/OFF state, such as bandwidth and radio power, for example. A feedback on the commands would also improve the system's safety and point to potential problems or attacks — if, for instance, a HAMSTER OBJECT is ordered to be turned OFF and stays ON.

Another possibility is the collaboration between the task management functionality in NP and the SEMU platform so services can be used as triggers for phase changes.

7.4 Declaration of Original Authorship

I confirm that this doctoral thesis has not been submitted in support of an application for another degree at this or any other teaching or research institution. It is the result of my own work and the use of all material from other sources has been properly and fully acknowledged. Research done in collaboration is also clearly indicated. Excerpts of this thesis have been either published or submitted for the appreciation of editorial boards of journals, conferences and workshops, according to the list of publications presented below. My contributions to each publication are listed as well.

7.5 Publication List

7.5.1 Published Papers

- MARCONATO, E. A.; **RODRIGUES, M.**; PIRES, R. D. M.; PIGATTO, D. F.; PINTO, A. R.; & BRANCO, K. R. AVENS-A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System. In: **Proceedings of the 50th Hawaii International Conference on System Sciences**. 2017.
  CONTRIBUTION LEVEL: Medium — the author contributed with the design and development of AVENS plug-in.
- **RODRIGUES, M.**; PIGATTO, D. F.; FONTES, J. V. C.; PINTO, A. S. R.; DIGUET, J.-P.; BRANCO, K. R. L. J. C. UAV Integration Into IoIT : Opportunities and Challenges. **International Conference on Autonomic and Autonomous Systems (ICAS)**, IARIA, p. 6, 2017.
  CONTRIBUTION LEVEL: Medium — the author was responsible for IoT-related research and integrations challenges.
- VANELLI, B.; **RODRIGUES, M.**; SILVA, M. P. D.; PINTO, A.; & DANTAS, M. A. R.. A Proposed Architecture for Robots as a Service. In: **Communication in Critical Embedded Systems.** Springer, Cham, 2014. p. 117-130.
  CONTRIBUTION LEVEL: Low — the author contributed with MQTT research.
- **RODRIGUES, M.**; PIGATTO, D. F.; BRANCO, K. R. L. J. C. Cloud-SPHERE: A Security Approach for Connected Unmanned Aerial Vehicles. In: **International Conference on Unmanned Aircraft Systems (ICUAS)**. Dallas, EUA: IEEE, 2018. p. 769–778. ISBN 978-1-5386-1354-2.
  CONTRIBUTION LEVEL: High — the author was the main investigator of this paper.
- PIGATTO, D. F.; FONTES, J. V. C.; **RODRIGUES, M.**; PINTO, A. R. S.; SMITH, J. BRANCO, K. R. L. J. C. **The Internet of Flying Things.** (Book Chapter). In: Internet of Things – Applications and Implementations, CRC Press.
  CONTRIBUTION LEVEL: Low — the author contributed with IoT-related research.
- **RODRIGUES, M.**; PIGATTO, D. F.; BRANCO, K. R. L. J. C. Navigation phases platform: Towards green computing for UAVs. In: **2018 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2018. p. 01171–01176.
  CONTRIBUTION LEVEL: High — the author was the main investigator of this paper.
- **RODRIGUES, M.**; BRANCO, K. R. L. J. C. UAVs at your service: Towards IoT integration with HAMSTER. In: **2019 International Conference on Unmanned Aircraft Systems (ICUAS).** [s.n.], 2019. p. 856–865.
  CONTRIBUTION LEVEL: High — the author was the main investigator of this paper.

- **RODRIGUES, M.**; AMARO, J.; OSÓRIO, F. S.; & RLJC, B. K. Authentication methods for UAV communication. In: **2019 IEEE Symposium on Computers and Communications (ISCC).** IEEE, 2019. p. 1210-1215.
  Contribution Level: High — the author was the main investigator of this paper.
- LA SCALEA, R.; **RODRIGUES, M.**; OSORIO, D. P. M.; LIMA, C. H.; SOUZA, R. D.; ALVES, H.; & BRANCO, K. C. Opportunities for autonomous UAV in harsh environments. In: **2019 16th International Symposium on Wireless Communication Systems (ISWCS).** IEEE, 2019. p. 227-232.
  Contribution Level: Low — the author contributed with theoretical research.
- **RODRIGUES, M.**; BRANCO, K. R. L. J. C. Cloud–SPHERE: Towards Secure UAV Service Provision. **Journal of Intelligent & Robotic Systems**, v. 97, p. 249–268, 2020. ISSN 1573-0409.
  Contribution Level: High — the author was the main investigator of this paper.
- **RODRIGUES, M.**; PIGATTO, D. F.; BRANCO, K. R. L. J. C. Context-aware operation for unmanned systems with hamster. In: **2020 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2020. p. 1–6.
  Contribution Level: High — the author was the main investigator of this paper.
- **RODRIGUES, M.**; BRANCO, K. R. L. J. C. Enabling UAV services in the IoT with HAMSTER. In: **2021 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2021. p. 1–6.
  Contribution Level: High — the author was the main investigator of this paper.

7.5.2 Submitted Papers

- **RODRIGUES, M.**; BRANCO, K. R. L. J. C. Providing UAV Secure Services in the Internet of Things with HAMSTER: a Tutorial. 2022.
  Contribution Level: High — the author was the main investigator of this paper.

BIBLIOGRAPHY

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: GELLERSEN, H.-W. (Ed.). **Handheld and Ubiquitous Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 304–307. ISBN 978-3-540-48157-7. Available: <https://doi.org/10.1007/3-540-48157-5_29>. Citation on page 71.

ABRAR, M.; AJMAL, U.; ALMOHAIMEED, Z. M.; GUI, X.; AKRAM, R.; MASROOR, R. Energy efficient uav-enabled mobile edge computing for iot devices: A review. **IEEE Access**, v. 9, p. 127779–127798, 2021. Available: <https://doi.org/10.1109/ACCESS.2021.3112104>. Citation on page 62.

ACHIR, M.; ABDELLI, A.; MOKDAD, L. A taxonomy of service discovery approaches in iot. In: **2020 8th International Conference on Wireless Networks and Mobile Communications (WINCOM)**. [s.n.], 2020. p. 6. Available: <https://doi.org/10.1109/WINCOM50532.2020.9272512>. Citations on pages 43 and 51.

AHMED, A. I. A.; GANI, A.; HAMID, S. H. A.; ABDELMABOUD, A.; SYED, H. J.; Habeeb Mohamed, R. A. A.; ALI, I. Service Management for IoT: Requirements, Taxonomy, Recent Advances and Open Research Challenges. **IEEE Access**, IEEE, v. 7, p. 155472–155488, 2019. ISSN 2169-3536. Available: <https://doi.org/10.1109/ACCESS.2019.2948027>. Citations on pages 42 and 43.

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. **IEEE Communications Surveys & Tutorials**, v. 17, n. 4, p. 2347–2376, jan 2015. ISSN 1553-877X. Available: <https://doi.org/10.1109/COMST.2015.2444095>. Citations on pages 25, 26, 39, 40, 42, 46, 47, 49, 50, 51, 57, 68, 88, and 132.

AL-KAFF, A.; MORENO, F. M.; ESCALERA, A. de la; ARMINGOL, J. M. Intelligent vehicle for search, rescue and transportation purposes. In: **IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)**. Shanghai, China: IEEE, 2017. p. 110–115. ISBN 978-1-5386-3923-8. Available: <https://doi.org/10.1109/SSRR.2017.8088148>. Citation on page 30.

ALI, M.; KHAN, S. U.; VASILAKOS, A. V. Security in cloud computing: Opportunities and challenges. **Information Sciences**, Elsevier Inc., v. 305, p. 357–383, 2015. ISSN 00200255. Available: <https://doi.org/10.1016/j.ins.2015.01.025>. Citations on pages 69 and 70.

ALMUHAYA, M. A. M.; JABBAR, W. A.; SULAIMAN, N.; ABDULMALEK, S. A Survey on LoRaWAN Technology: Recent Trends, Opportunities, Simulation Tools and Future Directions. **Electronics**, v. 11, n. 1, 2022. ISSN 2079-9292. Available: <https://doi.org/10.3390/electronics11010164>. Citation on page 53.

ALTAWY, R.; YOUSSEF, A. M. Security, Privacy, and Safety Aspects of Civilian Drones. **ACM Transactions on Cyber-Physical Systems**, ACM, v. 1, n. 2, p. 1–25, nov 2016. ISSN

2378-962X. Available: <https://doi.org/10.1145/3001836>. Citations on pages 31, 35, 36, 68, 69, and 70.

ALWATEER, M.; LOKE, S. W.; RAHAYU, W. Drone services: An investigation via prototyping and simulation. In: **IEEE 4th World Forum on Internet of Things (WF-IoT)**. Singapore, Singapore: IEEE, 2018. p. 367–370. ISBN 978-1-4673-9944-9. Available: <https://doi.org/10.1109/WF-IoT.2018.8355153>. Citations on pages 26, 62, 65, and 67.

AMJAD, A.; AZAM, F.; ANWAR, M. W.; BUTT, W. H. A Systematic Review on the Data Interoperability of Application Layer Protocols in Industrial IoT. **IEEE Access**, v. 9, p. 96528–96545, 2021. Available: <https://doi.org/10.1109/ACCESS.2021.3094763>. Citation on page 45.

ANTON, S. R.; INMAN, D. J. Vibration energy harvesting for unmanned aerial vehicles. In: **SPIE 6928, Active and Passive Smart Structures and Integrated Systems 2008**. SPIE, 2008. v. 692824. Available: <https://doi.org/10.1117/12.774990>. Citation on page 64.

ARCHIBALD, C.; SCHWALM, L.; BALL, J. E. A Survey of Security in Robotic Systems: Vulnerabilities, Attacks, and Solutions. **International Journal of Robotics and Automation**, v. 32, n. 2, p. 151–157, 2017. ISSN 1925-7090. Available: <https://doi.org/10.2316/Journal.206.2017.2.206-4705>. Citation on page 88.

ARELLANES, D.; LAU, K.-K. Evaluating IoT service composition mechanisms for the scalability of IoT systems. **Future Generation Computer Systems**, Elsevier B.V., v. 108, p. 827–848, jul 2020. ISSN 0167-739X. Available: <https://doi.org/10.1016/j.future.2020.02.073>. Citations on pages 42 and 43.

ARUMUGAM, R.; ENTI, V. R.; Liu Bingbing; Wu Xiaojun; BASKARAN, K.; Foong Foo Kong; KUMAR, A. S.; Kang Dee Meng; Goh Wai Kit. DAvinCi: A Cloud Computing framework for service robots. In: **IEEE International Conference on Robotics and Automation**. Anchorage, USA: IEEE, 2010. p. 3084–3089. ISBN 978-1-4244-5038-1. ISSN 1050-4729. Available: <https://doi.org/10.1109/ROBOT.2010.5509469>. Citation on page 66.

ASGHARI, P.; RAHMANI, A. M.; JAVADI, H. H. S. Service composition approaches in IoT: A systematic review. **Journal of Network and Computer Applications**, Elsevier Ltd, v. 120, n. January, p. 61–77, 2018. ISSN 10958592. Available: <https://doi.org/10.1016/j.jnca.2018.07.013>. Citation on page 43.

____. Internet of things applications: A systematic review. **Computer Networks**, v. 148, p. 241–261, 2019. ISSN 1389-1286. Available: <https://doi.org/10.1016/j.comnet.2018.12.008>. Citation on page 39.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. **Computer Networks**, Elsevier B.V., v. 54, n. 15, p. 2787–2805, oct 2010. ISSN 1389-1286. Available: <https://doi.org/10.1016/j.comnet.2010.05.010>. Citation on page 25.

AZIEZ, M.; BENHARZALLAH, S.; BENNOUI, H. Service discovery for the Internet of Things: Comparison study of the approaches. In: **4th International Conference on Control, Decision and Information Technologies (CoDIT)**. Barcelona, Spain: IEEE, 2017. p. 0599–0604. ISBN 978-1-5090-6465-6. Available: <https://doi.org/10.1109/CoDIT.2017.8102660>. Citation on page 43.

BANSAL, S.; KUMAR, D. IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication. **International Journal of Wireless Information Networks**, Springer US, feb 2020. ISSN 1068-9605. Available: <https://doi.org/10.1007/s10776-020-00483-7>. Citations on pages 39, 41, and 53.

Beecham Research Limited. **IoT Security Threat Map**. Beecham Research Limited, 2015. 4 p. Available: <http://www.beechamresearch.com/downloadFile.aspx?name=BRLIoTSecurityThreatMap2015explained.pdf>. Citation on page 58.

BEKMEZCI, I.; SAHINGOZ, O. K.; TEMEL, S. Flying Ad-Hoc Networks (FANETs): A survey. **Ad Hoc Networks**, Elsevier B.V., v. 11, n. 3, p. 1254–1270, may 2013. ISSN 1570-8705. Available: <https://doi.org/10.1016/j.adhoc.2012.12.004>. Citations on pages 32, 33, 34, and 35.

BEKMEZCI, I.; ŞENTÜRK, E.; TÜRKER, T. Security issues in flying ad-hoc networks (fanets). **Journal of Aeronautics and Space Technologies**, v. 9, n. 2, p. 13–21, Jul. 2016. Available: <https://193.255.215.140/index.php/JAST/article/view/32>. Citations on pages 35 and 38.

BIANZINO, A. P.; CHAUDET, C.; ROSSI, D.; ROUGIER, J.-L. A Survey of Green Networking Research. **IEEE Communications Surveys & Tutorials**, IEEE, v. 14, n. 1, p. 3–20, 2012. ISSN 1553-877X. Available: <https://doi.org/10.1109/SURV.2011.113010.00106>. Citation on page 65.

BILAL, R.; KHAN, B. M. Analysis of Mobility Models and Routing Schemes for Flying Ad-Hoc Networks (FANETS). **International Journal of Applied Engineering Research**, Research India Publications, v. 12, n. 12, p. 3263–3269, 2017. ISSN 0973-4562. Available: <https://www.ripublication.com/ijaer17/ijaerv12n12_37.pdf>. Citation on page 35.

BLOISI, D. D.; NARDI, D.; RICCIO, F.; TRAPANI, F. **Context-Enhanced Information Fusion**. Cham: Springer International Publishing, 2016. 675–699 p. (Advances in Computer Vision and Pattern Recognition). ISBN 978-3-319-28969-4. Available: <https://doi.org/10.1007/978-3-319-28971-7>. Citations on pages 71 and 81.

BLUETOOTH SPECIAL INTEREST GROUP. **Bluetooth Core Specification Version 5.2**. [S.l.], 2019. 3256 p. Available: <https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726>. Accessed: 2020-03-14. Citations on pages 49 and 52.

BOLLA, G. M.; CASAGRANDE, M.; COMAZZETTO, A.; Dal Moro, R.; DESTRO, M.; FANTIN, E.; COLOMBATTI, G.; ABOUDAN, A.; LORENZINI, E. C. ARIA: Air Pollutants Monitoring Using UAVs. In: **IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)**. Rome, Italy: IEEE, 2018. p. 225–229. ISBN 978-1-5386-2474-6. Available: <https://doi.org/10.1109/MetroAeroSpace.2018.8453584>. Citation on page 30.

BOLLA, R.; BRUSCHI, R.; DAVOLI, F.; CUCCHIETTI, F. Energy Efficiency in the Future Internet: A Survey of Existing Approaches and Trends in Energy-Aware Fixed Network Infrastructures. **IEEE Communications Surveys & Tutorials**, IEEE, v. 13, n. 2, p. 223–244, 2011. ISSN 1553-877X. Available: <https://doi.org/10.1109/SURV.2011.071410.00073>. Citation on page 65.

BORMANN, C.; ERSUE, M.; KERANEN, A. **RFC 7228: Terminology for Constrained-Node Networks**. Fermont, 2014. 17 p. Available: <https://doi.org/10.17487/rfc7228>. Citation on page 40.

BOTTA, A.; DONATO, W. de; PERSICO, V.; PESCAPÉ, A. Integration of Cloud computing and Internet of Things: A survey. **Future Generation Computer Systems**, Elsevier, v. 56, p. 684–700, mar 2016. ISSN 0167-739X. Available: <https://doi.org/10.1016/j.future.2015.09.021>. Citations on pages 26, 42, 54, and 55.

BUJARI, A.; PALAZZI, C. E.; RONZANI, D. FANET Application Scenarios and Mobility Models. In: **Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet)**. New York, New York, USA: ACM Press, 2017. p. 43–46. ISBN 9781450349604. Available: <https://doi.org/10.1145/3086439.3086440>. Citation on page 34.

BUTTERFIELD, A.; NGONDI, G.; KERR, A. (Ed.). **A Dictionary of Computer Science**. Oxford University Press, 2016. (Oxford Quick Reference). ISBN 9780199688975. Available: <https://doi.org/10.1093/acref/9780199688975.001.0001>. Citations on pages 41 and 44.

BUTUN, I.; ÖSTERBERG, P.; SONG, H. Security of the internet of things: Vulnerabilities, attacks, and countermeasures. **IEEE Communications Surveys Tutorials**, v. 22, n. 1, p. 616–644, 2020. Available: <https://doi.org/10.1109/COMST.2019.2953364>. Citation on page 58.

CABRERA, C.; PALADE, A.; CLARKE, S. An evaluation of service discovery protocols in the internet of things. In: **Proceedings of the Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2017. p. 469–476. ISBN 9781450344869. Available: <https://doi.org/10.1145/3019612.3019698>. Citation on page 51.

CAVALCANTE, E.; PEREIRA, J.; ALVES, M. P.; MAIA, P.; MOURA, R.; BATISTA, T.; DELICATO, F. C.; PIRES, P. F. On the Interplay of Internet of Things and Cloud Computing: A Systematic Mapping Study. **Computer Communications**, Elsevier, v. 89-90, p. 17–33, 2016. ISSN 0140-3664. Available: <https://doi.org/10.1016/j.comcom.2016.03.012>. Citations on pages 26, 42, 54, and 55.

CHABAREK, J.; SOMMERS, J.; BARFORD, P.; ESTAN, C.; TSIANG, D.; WRIGHT, S. Power Awareness in Network Design and Routing. In: **IEEE Conference on Computer Communications (INFOCOM)**. Phoenix, USA: IEEE, 2008. p. 457–465. ISBN 978-1-4244-2026-1. ISSN 0743-166X. Available: <https://doi.org/10.1109/INFOCOM.2008.93>. Citation on page 65.

CHEBROLU, N.; LABE, T.; STACHNISS, C. Robust Long-Term Registration of UAV Images of Crop Fields for Precision Agriculture. **IEEE Robotics and Automation Letters**, IEEE, v. 3, n. 4, p. 3097–3104, oct 2018. ISSN 2377-3766. Available: <https://doi.org/10.1109/LRA.2018.2849603>. Citation on page 30.

CHETTRI, L.; BERA, R. A comprehensive survey on internet of things (iot) toward 5g wireless systems. **IEEE Internet of Things Journal**, v. 7, n. 1, p. 16–32, 2020. Available: <https://doi.org/10.1109/JIOT.2019.2948888>. Citation on page 60.

CHIGAN, C.; LI, L.; YE, Y. Resource-aware Self-Adaptive Security Provisioning in Mobile Ad Hoc Networks. In: **IEEE Wireless Communications and Networking Conference**. New Orleans, USA: IEEE, 2005. v. 4, p. 2118–2124. ISBN 0-7803-8966-2. ISSN 1525-3511. Available: <https://doi.org/10.1109/WCNC.2005.1424845>. Citations on pages 72 and 81.

CHRIKI, A.; TOUATI, H.; SNOUSSI, H.; KAMOUN, F. FANET: Communication, mobility models and security issues. **Computer Networks**, Elsevier B.V., v. 163, nov 2019. ISSN 1389-1286. Available: <https://doi.org/10.1016/j.comnet.2019.106877>. Citations on pages 33, 34, and 35.

Cloud Security Alliance. **The Egregious 11: Cloud Computing Top Threats in 2019**. [S.l.], 2016. 43 p. Available: <https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/>. Citation on page 58.

CoAP — Constrained Application Protocol | Overview. 2022. Https://coap.technology. Accessed in 2022-04-20. Citation on page 49.

ČOLAKOVIĆ, A.; HADŽIALIĆ, M. Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues. **Computer Networks**, Elsevier B.V., v. 144, p. 17–39, oct 2018. ISSN 1389-1286. Available: <https://doi.org/10.1016/j.comnet.2018.07.017>. Citations on pages 39, 40, and 53.

CONTIKI: The Open Source Operating System for the Internet of Things. 2021. Http://www.contiki-os.org. Accessed in 2021-12-20. Citation on page 41.

COSKUN, V.; OZDENIZCI, B.; OK, K. A Survey on Near Field Communication (NFC) Technology. **Wireless Personal Communications**, Springer, v. 71, n. 3, p. 2259–2294, aug 2013. ISSN 2259–2294. Available: <https://doi.org/10.1007/s11277-012-0935-5>. Citations on pages 49 and 52.

COSTA, L. F.; GONCALVES, L. M. RoboServ: A ROS Based Approach towards Providing Heterogeneous Robots as a Service. In: **2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)**. Recife, Brazil: IEEE, 2016. p. 169–174. ISBN 978-1-5090-3656-1. Available: <https://doi.org/10.1109/LARS-SBR.2016.35>. Citation on page 66.

DALAMAGKIDIS, K. Classification of UAVs. In: VALAVANIS, K. P.; VACHTSEVANOS, G. J. (Ed.). **Handbook of Unmanned Aerial Vehicles**. Dordrecht: Springer Netherlands, 2015. chap. 5, p. 83–91. ISBN 978-90-481-9707-1. Available: <https://dix.doi.org/10.1007/978-90-481-9707-1{_}94>. Citation on page 29.

DANG, D.-K.; MIFDAOUI, A.; GAYRAUD, T. Fly-By-Wireless for next generation aircraft: Challenges and potential solutions. In: **IFIP Wireless Days**. Dublin, Ireland: IEEE, 2012. p. 1–8. ISBN 978-1-4673-4404-3. ISSN 2156-9711. Available: <https://doi.org/10.1109/WD.2012.6402820>. Citation on page 65.

DARROUDI, S. M.; GOMEZ, C. Bluetooth Low Energy Mesh Networks: A Survey. **Sensors**, v. 17, n. 7, p. 1467, jun 2017. ISSN 1424-8220. Available: <https://doi.org/10.3390/s17071467>. Citation on page 53.

DAVIES, L.; VAGAPOV, Y.; GROUT, V.; CUNNINGHAM, S.; ANUCHIN, A. Review of air traffic management systems for uav integration into urban airspace. In: **2021 28th International Workshop on Electric Drives: Improving Reliability of Electric Drives (IWED)**. [S.l.: s.n.], 2021. p. 1–6. Citations on pages 31 and 32.

DHUMANE, A.; PRASAD, R.; PRASAD, J. Routing Issues in Internet of Things : A Survey. In: AO, S. I.; CASTILLO, O.; DOUGLAS, C.; FENG, D. D.; KORSUNSKY, A. M. (Ed.). **International MultiConference of Engineers and Computer Scientists (IMECS)**. Hong Kong, China: Newswood Limited, 2016. I, p. 404–412. ISBN 978-988-19253-8-1. ISSN 2078-0966. Available: <http://www.iaeng.org/publication/IMECS2016/IMECS2016_pp404-412.pdf>. Citation on page 51.

DING, J.; NEMATI, M.; RANAWEERA, C.; CHOI, J. Iot connectivity technologies and applications: A survey. **IEEE Access**, v. 8, p. 67646–67673, 2020. Available: <https://doi.org/10.1109/ACCESS.2020.2985932>. Citations on pages 48, 52, and 53.

DORIYA, R.; CHAKRABORTY, P.; NANDI, G. Robotic Services in Cloud Computing Paradigm. In: **International Symposium on Cloud and Services Computing**. Mangalore, India: IEEE, 2012. p. 80–83. ISBN 978-1-4673-4854-6. Available: <https://doi.org/10.1109/ISCOS.2012.24>. Citation on page 66.

DUSTDAR, S.; AVASALCAI, C.; MURTURI, I. Invited Paper: Edge and Fog Computing: Vision and Research Challenges. In: **2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)**. IEEE, 2019. p. 96–9609. ISBN 978-1-7281-1442-2. Available: <https://doi.org/10.1109/SOSE.2019.00023>. Citation on page 55.

ECLIPSE. **Eclipse Mosquitto®- An open source MQTT broker**. 2018. Available: <https://mosquitto.org>. Citation on page 133.

____. **Eclipse Paho**. 2018. Available: <https://www.eclipse.org/paho/>. Citation on page 133.

EL-HAJJ, M.; FADLALLAH, A.; CHAMOUN, M.; SERHROUCHNI, A. A Survey of Internet of Things (IoT) Authentication Schemes. **Sensors**, v. 19, n. 5, p. 1141, mar 2019. ISSN 1424-8220. Available: <https://doi.org/10.3390/s19051141>. Citation on page 58.

Elgezabal Gomez, O. Fly-by-wireless: Benefits, risks and technical challenges. In: **CANEUS Fly by Wireless Workshop**. Orono, USA: IEEE, 2010. p. 14–15. ISBN 978-1-4244-9255-8. Available: <https://doi.org/10.1109/FBW.2010.5613788>. Citation on page 65.

ELKHODR, M.; SHAHRESTANI, S.; CHEUNG, H. Emerging Wireless Technologies in the Internet of Things : A Comparative Study. **International Journal of Wireless & Mobile Networks**, AIRCC, v. 8, n. 5, p. 67–82, oct 2016. ISSN 0975-4679. Available: <https://doi.org/10.5121/ijwmn.2016.8505>. Citation on page 41.

ELRAWY, M. F.; AWAD, A. I.; HAMED, H. F. A. Intrusion detection systems for IoT-based smart environments: a survey. **Journal of Cloud Computing**, v. 7, 2018. ISSN 2192-113X. Available: <https://doi.org/10.1186/s13677-018-0123-6>. Citation on page 60.

EOM, J. ho. Security Threats Recognition and Countermeasures on Smart Battlefield Environment based on IoT. **International Journal of Security and Its Applications**, v. 9, n. 7, p. 347–356, jul 2015. ISSN 1738-9976. Available: <https://doi.org/10.14257/ijsia.2015.9.7.32>. Citations on pages 56 and 58.

ERMACORA, G.; TOMA, A.; ROSA, S.; BONA, B.; CHIABERGE, M.; SILVAGNI, M.; GASPARDONE, M.; ANTONINI, R. A Cloud Based Service for Management and Planning of Autonomous UAV Missions in Smart City Scenarios. In: Jan Hodicky (Ed.). **Modelling and Simulation for Autonomous Systems**. Springer International Publishing, 2014, (Lecture Notes in Computer Science, v. 8906). chap. 3, p. 20–26. ISBN 978-3-319-13823-7. Available: <https://doi.org/10.1007/978-3-319-13823-7_3>. Citation on page 62.

ERTURK, A.; RENNO, J. M.; INMAN, D. J. Modeling of Piezoelectric Energy Harvesting from an L-shaped Beam-mass Structure with an Application to UAVs. **Journal of Intelligent Material Systems and Structures**, SAGE, v. 20, n. 5, p. 529–544, mar 2009. ISSN 1045-389X. Available: <https://doi.org/10.1177/1045389X08098096>. Citation on page 64.

EUROPEAN AVIATION SAFETY AGENCY. **‘Prototype’ Commission Regulation on Unmanned Aircraft Operations**. [S.l.], 2016. 72 p. Available: <https://www.easa.europa.eu/system/files/dfu/UASPrototypeRegulationfinal.pdf>. Citation on page 32.

FAN, X.; CAI, W.; LIN, J. A survey of routing protocols for highly dynamic mobile ad hoc networks. In: **IEEE 17th International Conference on Communication Technology (ICCT)**. [s.n.], 2017. p. 1412–1417. ISSN 2576-7828. Available: <https://doi.org/10.1109/ICCT.2017.8359865>. Citation on page 35.

FEDERAL AVIATION ADMINISTRATION. **14 CFR part 107**. 2016. Available: <https://www.ecfr.gov/cgi-bin/text-idx?&node=pt14.2.107>. Citation on page 32.

____. **Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS) Roadmap**. [S.l.], 2018. 74 p. 2nd. Edition. Available: <https://www.faa.gov/uas/resources/policy_library/media/Second_Edition_Integration_of_Civil_UAS_NAS_Roadmap_July%202018.pdf>. Citations on pages 31 and 32.

____. **Unmanned Aircraft System (UAS) Traffic Management (UTM) Concept of Operations**. 2018. Available: <https://utm.arc.nasa.gov/docs/2018-UTM-ConOps-v1.0.pdf>. Citation on page 32.

FERRAG, M. A.; MAGLARAS, L. A.; JANICKE, H.; JIANG, J.; SHU, L. Authentication Protocols for Internet of Things: A Comprehensive Survey. **Security and Communication Networks**, v. 2017, p. 1–41, 2017. ISSN 1939-0114. Available: <https://doi.org/10.1155/2017/6562953>. Citation on page 58.

FreeRTOS — Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. 2021. Https://www.freertos.org. Accessed in 2021-12-20. Citation on page 42.

FURRER, J.; KAMEI, K.; SHARMA, C.; MIYASHITA, T.; HAGITA, N. UNR-PF: An open-source platform for cloud networked robotic services. In: **IEEE/SICE International Symposium on System Integration (SII)**. Fukuoka, Japan: IEEE, 2012. p. 945–950. ISBN 978-1-4673-1497-8. Available: <https://doi.org/10.1109/SII.2012.6427281>. Citation on page 66.

GABRIELSSON, J.; BUGEJA, J.; VOGEL, B. Hacking a Commercial Drone with Open-Source Software: Exploring Data Privacy Violations. In: **10th Mediterranean Conference on Embedded Computing (MECO)**. IEEE, 2021. p. 1–5. ISBN 978-1-6654-3912-1. Available: <https://doi.org/10.1109/MECO52532.2021.9460295>. Citation on page 32.

GHAZZAI, H.; Ben Ghorbel, M.; KADRI, A.; HOSSAIN, M. J.; MENOUAR, H. Energy-Efficient Management of Unmanned Aerial Vehicles for Underlay Cognitive Radio Systems. **IEEE Transactions on Green Communications and Networking**, IEEE, v. 1, n. 4, p. 434–443, dec 2017. ISSN 2473-2400. Available: <https://doi.org/10.1109/TGCN.2017.2750721>. Citation on page 65.

GIRI, A.; DUTTA, S.; NEOGY, S.; DAHAL, K.; PERVEZ, Z. Internet of things (IoT): a survey on architecture, enabling technologies, applications and challenges. In: **Proceedings of the 1st International Conference on Internet of Things and Machine Learning - IML ’17**. Liverpool, UK: ACM Press, 2017. ISBN 9781450352437. Available: <https://doi.org/10.1145/3109761.3109768>. Citations on pages 46 and 47.

GOMEZ, C.; PURDIE, H. UAV- based Photogrammetry and Geocomputing for Hazards and Disaster Risk Monitoring – A Review. **Geoenvironmental Disasters**, v. 3, n. 1, p. 23, nov 2016. ISSN 2197-8670. Available: <https://doi.org/10.1186/s40677-016-0060-y>. Citation on page 30.

GUPTA, L.; JAIN, R.; VASZKUN, G. Survey of Important Issues in UAV Communication Networks. **IEEE Communications Surveys & Tutorials**, v. 18, n. 2, p. 1123–1152, 2016. ISSN 1553-877X. Available: <https://doi.org/10.1109/COMST.2015.2495297>. Citations on pages 32, 33, 34, 35, and 39.

GYRARD, A.; DATTA, S. K.; BONNET, C. A survey and analysis of ontology-based software tools for semantic interoperability in iot and wot landscapes. In: **IEEE 4th World Forum on Internet of Things (WF-IoT)**. IEEE, 2018. p. 86–91. ISBN 978-1-4673-9944-9. Available: <https://doi.org/10.1109/WF-IoT.2018.8355091>. Citation on page 45.

HAHM, O.; BACCELLI, E.; PETERSEN, H.; TSIFTES, N. Operating systems for low-end devices in the internet of things: A survey. **IEEE Internet of Things Journal**, v. 3, n. 5, p. 720–734, Oct 2016. ISSN 2372-2541. Available: <https://doi.org/10.1109/JIOT.2015.2505901>. Citations on pages 41 and 42.

HAMZEI, M.; Jafari Navimipour, N. Toward Efficient Service Composition Techniques in the Internet of Things. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 5, p. 3774–3787, oct 2018. ISSN 2327-4662. Available: <https://doi.org/10.1109/JIOT.2018.2861742>. Citation on page 43.

HANNAN, A.; HUSSAIN, F.; ALI, N.; EHATISHAM-UL-HAQ, M.; ASHRAF, M. U.; ALGHAMDI, A. M.; ALFAKEEH, A. S. A decentralized hybrid computing consumer authentication framework for a reliable drone delivery as a service. **PLoS One**, v. 16, n. 4, 2021. Available: <https://doi.org/10.1371/journal.pone.0250737>. Citation on page 70.

HARDKERNEL. **ODROID-XU4**. 2018. Available: <https://www.hardkernel.com/main/products/prdt_info.php>. Citation on page 133.

HARTMANN, K.; STEUP, C. The vulnerability of UAVs to cyber attacks - An approach to the risk assessment. In: **International Conference on Cyber Conflict (CyCon)**. Tallinn, Estonia: IEEE, 2013. p. 1–23. ISBN 978-1-4799-0450-1. ISSN 2325-5366. Available: <https://ieeexplore.ieee.org/document/6568373>. Citations on pages 36, 62, and 70.

HASHIZUME, K.; ROSADO, D. G.; FERNÁNDEZ-MEDINA, E.; FERNANDEZ, E. B. An analysis of security issues for cloud computing. **Journal of Internet Services and Applications**, v. 4, n. 1, p. 5, 2013. ISSN 1869-0238. Available: <https://doi.org/10.1186/1869-0238-4-5>. Citations on pages 58, 68, 70, and 88.

HASSAN, R.; QAMAR, F.; HASAN, M. K.; AMAN, A. H. M.; AHMED, A. S. Internet of things and its applications: A comprehensive survey. **Symmetry**, v. 12, n. 10, 2020. ISSN 2073-8994. Available: <https://doi.org/10.3390/sym12101674>. Citation on page 39.

HAYAT, S.; YANMAZ, E.; MUZAFFAR, R. Survey on Unmanned Aerial Vehicle Networks for Civil Applications: A Communications Viewpoint. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 4, p. 2624–2661, 2016. ISSN 1553-877X. Available: <https://doi.org/10.1109/COMST.2016.2560343>. Citations on pages 30 and 69.

HE, D.; CHAN, S.; GUIZANI, M. Communication Security of Unmanned Aerial Vehicles. **IEEE Wireless Communications**, IEEE, v. 24, n. 4, p. 134–139, 2017. ISSN 1536-1284. Available: <https://doi.org/10.1109/MWC.2016.1600073WC>. Citations on pages 32, 35, 68, 69, and 88.

HOOK, B. **Write portable code: an introduction to developing software for multiple platforms**. 1. ed. San Francisco, CA, USA: No Starch Press, 2005. 272 p. ISBN 9781593270568. Citation on page 108.

HUANG, J.; LI, R.; AN, J.; NTALASHA, D.; YANG, F.; LI, K. Energy-Efficient Resource Utilization for Heterogeneous Embedded Computing Systems. **IEEE Transactions on Computers**, IEEE, v. 66, n. 9, p. 1518–1531, sep 2017. ISSN 0018-9340. Available: <https://doi.org/10.1109/TC.2017.2693186>. Citation on page 64.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. **2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)**, p. 791–798, 2008. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4554519>. Citation on page 132.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. Ieee standard glossary of software engineering terminology. p. 84, 1990. Available: <https://doi.org/10.1109/IEEESTD.1990.101064>. Citation on page 45.

____. **IEEE Standard for Low-Rate Wireless Networks**. [S.l.], 2016. 709 p. Available: <https://doi.org/10.1109/IEEESTD.2016.7460875>. Citations on pages 49 and 53.

Institute of Electrical and Electronics Engineers. **IEEE Std 1934-2018: IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing**. [S.l.], 2018. 1-176 p. Available: <https://doi.org/10.1109/IEEESTD.2018.8423800>. Citations on pages 54 and 55.

INTERNATIONAL BUSINESS MACHINES CORPORATION (IBM). **MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2**. [S.l.], 2014. 81 p. Available: <http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf>. Citation on page 50.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 17788:2014**: Information technology – Cloud computing – Overview and vocabulary. Geneva, 2014. 16 p. Available: <https://www.iso.org/standard/60544.html>. Citation on page 54.

INTERNATIONAL TELECOMMUNICATION UNION. **Overview of the Internet of things**. Geneva, 2012. 22 p. Available: <https://handle.itu.int/11.1002/1000/11559>. Citations on pages 25, 39, 45, 47, 57, and 69.

INTERNET ENGINEERING TASK FORCE. **RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks**. Fermont, 2012. 157 p. Available: <https://doi.org/10.17487/RFC6550>. Citations on pages 49 and 52.

INTERNET ENGINEERING TASK FORCE (IETF). **Extensible Messaging and Presence Protocol (XMPP): Core**. Fremont, 2011. 211 p. Available: <https://doi.org/10.17487/RFC6120>. Citations on pages 49 and 50.

____. **DNS-Based Service Discovery**. Fermont, 2013. 1–49 p. Available: <https://doi.org/10.17487/rfc6763>. Citations on pages 49 and 51.

____. **Multicast DNS**. Fermont, 2013. 70 p. Available: <https://doi.org/10.17487/rfc6762>. Citations on pages 49 and 51.

____. **The Constrained Application Protocol (CoAP)**. Fermont, 2014. 112 p. Available: <https://doi.org/10.17487/rfc7252>. Citation on page 49.

JAVAID, A. Y.; SUN, W.; DEVABHAKTUNI, V. K.; ALAM, M. Cyber Security Threat Analysis and Modeling of an Unmanned Aerial Vehicle System. In: **IEEE Conference on Technologies for Homeland Security (HST)**. Waltham, USA: IEEE, 2012. p. 585–590. ISBN 978-1-4673-2709-1. Citations on pages 82, 88, and 138.

JING, Q.; VASILAKOS, A. V.; WAN, J.; LU, J.; QIU, D. Security of the Internet of Things: perspectives and challenges. **Wireless Networks**, v. 20, n. 8, p. 2481–2501, 2014. ISSN: 1022-0038. Citation on page 62.

KANISTRAS, K.; MARTINS, G.; RUTHERFORD, M. J.; VALAVANIS, K. P. Survey of Unmanned Aerial Vehicles (UAVs) for Traffic Monitoring. In: VALAVANIS, K. P.; VACHTSEVANOS, G. J. (Ed.). **Handbook of Unmanned Aerial Vehicles**. 1. ed. Dordrecht: Springer Netherlands, 2015. chap. 122, p. 2643–2666. Available: <https://link.springer.com/10.1007/978-90-481-9707-1{_}122>. Citation on page 30.

KHAN, F. U.; AWAIS, M.; RASHEED, M. B.; MASOOD, B.; GHADI, Y. A Comparison of Wireless Standards in IoT for Indoor Localization Using LoPy. **IEEE Access**, v. 9, p. 65925–65933, 2021. Available: <https://doi.org/10.1109/ACCESS.2021.3076371>. Citation on page 46.

KHAN, M. A.; SAFI, A.; QURESHI, I. M.; KHAN, I. U. Flying ad-hoc networks (FANETs): A review of communication architectures, and routing protocols. In: **First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)**. Karachi, Pakistan: IEEE, 2017. p. 9. ISBN 978-1-5386-2969-7. Available: <https://doi.org/10.1109/INTELLECT.2017.8277614>. Citation on page 35.

KHAN, M. N.; RAO, A.; CAMTEPE, S. Lightweight cryptographic protocols for iot-constrained devices: A survey. **IEEE Internet of Things Journal**, v. 8, n. 6, p. 4132–4156, 2021. Available: <https://doi.org/10.1109/JIOT.2020.3026493>. Citation on page 59.

KHAN, S. Z.; MOHSIN, M.; IQBAL, W. On GPS spoofing of aerial platforms: a review of threats, challenges, methodologies, and future research directions. **PeerJ Computer Science**, v. 7, may 2021. ISSN 2376-5992. Available: <https://doi.org/10.7717/peerj-cs.507>. Citation on page 37.

KHARRUFA, H.; AL-KASHOASH, H. A. A.; KEMP, A. H. RPL-Based Routing Protocols in IoT Applications: A Review. **IEEE Sensors Journal**, IEEE, v. 19, n. 15, p. 5952–5967, 2019. ISSN 1558-1748. Available: <https://doi.org/10.1109/JSEN.2019.2910881>. Citation on page 52.

KIM, T.; CHANG, J. M. Enhanced Power Saving Mechanism for Large-Scale 802.11ah Wireless Sensor Networks. **IEEE Transactions on Green Communications and Networking**, IEEE, v. 1, n. 4, p. 516–527, dec 2017. ISSN 2473-2400. Available: <https://doi.org/10.1109/TGCN.2017.2727056>. Citation on page 65.

KIM, Y.; JO, J.; SHAW, M. A lightweight communication architecture for small UAS Traffic Management (sUTM). In: **Integrated Communication, Navigation and Surveillance Conference (ICNS)**. Herndon, USA: IEEE, 2015. ISBN 978-1-4799-8952-2. Available: <https://doi.org/10.1109/ICNSURV.2015.7121263>. Citation on page 32.

KIM, Y.; JO, J.; SHRESTHA, S. A Server-Based Real-Time Privacy Protection Scheme against Video Surveillance by Unmanned Aerial Systems. In: **International Conference on Unmanned Aircraft Systems (ICUAS)**. Orlando, USA: IEEE, 2014. p. 684–691. ISBN 978-1-4799-2376-2. Available: <https://doi.org/10.1109/ICUAS.2014.6842313>. Citations on pages 29 and 63.

KONDURU, V. R.; BHARAMAGOUDRA, M. R. Challenges and solutions of interoperability on IoT: How far have we come in resolving the IoT interoperability issues. In: **2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)**. [s.n.], 2017. p. 572–576. Available: <https://doi.org/10.1109/SmartTechCon.2017.8358436>. Citation on page 45.

KONG, J.; LUO, H.; XU, K.; GU, D. L.; GERLA, M.; LU, S. Adaptive security for multilevel ad hoc networks. **Wireless Communications and Mobile Computing**, John Wiley & Sons, Ltd., v. 2, n. 5, p. 533–547, aug 2002. ISSN 1530-8669. Citation on page 72.

KOUBAA, A.; QURESHI, B.; SRITI, M.-F.; JAVED, Y.; TOVAR, E. A service-oriented Cloud-based management system for the Internet-of-Drones. In: **IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)**. Coimbra, Portugal: IEEE, 2017. p. 329–335. ISBN 978-1-5090-6234-8. Available: <https://doi.org/10.1109/ICARSC.2017.7964096>. Citations on pages 61, 62, and 67.

KRISHNA, C. G. L.; MURPHY, R. R. A review on cybersecurity vulnerabilities for unmanned aerial vehicles. In: **IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)**. Shanghai, China: IEEE, 2017. p. 194–199. Citation on page 82.

LAGKAS, T.; ARGYRIOU, V.; BIBI, S.; SARIGIANNIDIS, P. UAV IoT Framework Views and Challenges: Towards Protecting Drones as "Things". **Sensors**, v. 18, n. 11, 2018. ISSN 1424-8220. Available: <https://doi.org/10.3390/s18114015>. Citations on pages 70 and 71.

LAROUI, M.; NOUR, B.; MOUNGLA, H.; CHERIF, M. A.; AFIFI, H.; GUIZANI, M. Edge and fog computing for IoT: A survey on current research activities & future directions. **Computer Communications**, Elsevier B.V., v. 180, n. July 2020, p. 210–231, 2021. ISSN 1873-703X. Available: <https://doi.org/10.1016/j.comcom.2021.09.003>. Citation on page 55.

LAWRANCE, N.; SUKKARIEH, S. Wind Energy Based Path Planning for a Small Gliding Unmanned Aerial Vehicle. In: **AIAA Guidance, Navigation, and Control Conference**. Chicago, USA: American Institute of Aeronautics and Astronautics, 2009. ISBN 978-1-60086-978-5. ISSN 2009-6112. Available: <https://doi.org/10.2514/6.2009-6112>. Citation on page 64.

LI, S. Introduction. In: **Securing the Internet of Things**. 1. ed. Cambridge, MA: Syngress, 2017. chap. 1, p. 1–25. ISBN 9780128044582. Available: <https://doi.org/10.1016/B978-0-12-804458-2.00001-9>. Citation on page 88.

LI, S.; TRYFONAS, T.; LI, H. The Internet of Things: a security point of view. **Internet Research**, Emerald Group Publishing Limited, v. 26, n. 2, p. 337–359, 2016. ISSN 1066-2243. Available: <https://doi.org/10.1108/IntR-07-2014-0173>. Citations on pages 56, 57, and 58.

LIN, J.; YU, W.; ZHANG, N.; YANG, X.; ZHANG, H.; ZHAO, W. A survey on internet of things: Architecture, enabling , security and privacy, and applications. **IEEE Internet of Things Journal**, v. 4, n. 5, p. 1125–1142, 2017. Available: <https://doi.org/10.1109/JIOT.2017.2683200>. Citations on pages 39, 40, 47, 50, 51, 55, 57, 68, 69, and 70.

LONE, A. H.; NAAZ, R. Applicability of blockchain smart contracts in securing internet and iot: A systematic literature review. **Computer Science Review**, v. 39, 2021. ISSN 1574-0137. Available: <https://doi.org/10.1016/j.cosrev.2020.100360>. Citation on page 60.

LoRa Alliance. 2022. Https://lora-alliance.org. Accessed in 2022-04-20. Citations on pages 49 and 53.

MAGOTEAUX, K. C.; SANDERS, B.; SODANO, H. A. Investigation of an energy harvesting small unmanned air vehicle. In: **Proc. SPIE 6928, Active and Passive Smart Structures and Integrated Systems 2008**. San Diego, USA: [s.n.], 2008. v. 6928. ISBN 978-0-8194-7114-7. ISSN 0277-786X. Available: <https://doi.org/10.1117/12.775851>. Citation on page 64.

MAHMOUD, S.; JAWHAR, I.; MOHAMED, N. A Softwarization Architecture for UAVs and WSNs as Part of the Cloud Environment. In: **International Conference on Cloud Engineering Workshop (IC2EW)**. Berlin, Germany: IEEE, 2016. p. 13–18. ISBN 978-1-5090-3684-4. Available: <https://doi.org/10.1109/IC2EW.2016.17>. Citation on page 67.

MAHMOUD, S.; MOHAMED, N. Toward a Cloud Platform for UAV Resources and Services. In: **IEEE Symposium on Network Cloud Computing and Applications (NCCA)**. Munich, Germany: IEEE, 2015. p. 23–30. ISBN 978-1-4673-7741-6. Available: <https://doi.org/10.1109/NCCA.2015.14>. Citations on pages 26, 65, and 85.

MAHMOUD, S.; MOHAMED, N.; AL-JAROODI, J. Integrating UAVs into the Cloud Using the Concept of the Web of Things. **Journal of Robotics**, p. 1–10, 2015. ISSN 1687-9600. Available: <https://doi.org/10.1155/2015/631420>. Citations on pages 26, 61, 62, 65, and 66.

MAKHDOOM, I.; ABOLHASAN, M.; LIPMAN, J.; LIU, R. P.; NI, W. Anatomy of threats to the internet of things. **IEEE Communications Surveys & Tutorials**, v. 21, n. 2, p. 1636–1675, 2019. Available: <https://doi.org/10.1109/COMST.2018.2874978>. Citation on page 56.

MAPLE, C. Security and privacy in the internet of things. **Journal of Cyber Policy**, Taylor & Francis, v. 2, n. 2, p. 155–184, may 2017. ISSN 2373-8871. Available: <https://doi.org/10.1080/23738871.2017.1366536>. Citation on page 69.

MARSHALL, D. M. Coordination with Manned Aircraft and Air Traffic Control. In: **Encyclopedia of Aerospace Engineering**. Chichester: John Wiley & Sons, Ltd, 2016. ISBN 9780470686652. Available: <https://doi.org/10.1002/9780470686652.eae1129>. Citation on page 31.

MEDDEB, A. Internet of Things Standards: Who Stands Out from the Crowd? **IEEE Communications Magazine**, IEEE, v. 54, n. 7, p. 40–47, jul 2016. ISSN 0163-6804. Available: <https://doi.org/10.1109/MCOM.2016.7514162>. Citations on pages 25 and 48.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. [S.l.], 2011. 7 p. Available: <https://doi.org/10.6028/NIST.SP.800-145>. Citation on page 54.

MIORANDI, D.; SICARI, S.; De Pellegrini, F.; CHLAMTAC, I. Internet of things: Vision, applications and research challenges. **Ad Hoc Networks**, Elsevier, v. 10, n. 7, p. 1497–1516, 2012. ISSN 1570-8705. Available: <https://doi.org/10.1016/j.adhoc.2012.02.016>. Citations on pages 26, 45, 47, 68, 69, and 70.

MOHAMED, N.; AL-JAROODI, J. Service-Oriented Middleware for Collaborative UAVs. In: **IEEE International Conference on Information Reuse and Integration (IRI)**. San Francisco, USA: IEEE, 2013. p. 185–192. ISBN 9781479910502. Available: <https://doi.org/10.1109/IRI.2013.6642471>. Citations on pages 33 and 66.

MOHAMED, N.; AL-JAROODI, J.; JAWHAR, I.; NOURA, H.; MAHMOUD, S. Uavfog: A uav-based fog computing for internet of things. In: **2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)**. [S.l.: s.n.], 2017. p. 1–8. Citations on pages 62 and 70.

MOKHTARZADEH, H.; COLTEN, T. **Small UAV Position and Attitude, Raw Sensor, and Aerial Imagery Data Collected over Farm Field with Surveyed Markers**. 2015. Available: <https://doi.org/10.13020/D6BC7Z>. Citation on page 133.

MONTENEGRO, G.; KUSHALNAGAR, N.; HUI, J.; CULLER, D. **Transmission of IPv6 Packets over IEEE 802.15.4 Networks**. RFC Editor, 2007. RFC 4944. (Request for Comments, 4944). Available: <https://doi.org/10.17487/rfc4944>. Citations on pages 49 and 51.

MORADBEIKIE, A.; KESHAVARZ, A.; ROSTAMI, H.; PAIVA, S.; LOPES, S. I. GNSS-Free Outdoor Localization Techniques for Resource-Constrained IoT Architectures: A Literature Review. **Applied Sciences**, v. 11, n. 22, 2021. ISSN 2076-3417. Available: <https://doi.org/10.3390/app112210793>. Citation on page 46.

MOTLAGH, N. H.; TALEB, T.; AROUK, O. Low-Altitude Unmanned Aerial Vehicles-Based Internet of Things Services: Comprehensive Survey and Future Perspectives. **IEEE Internet of Things Journal**, v. 3, n. 6, p. 899–922, 2016. ISSN 2327-4662. Available: <https://doi.org/10.1109/JIOT.2016.2612119>. Citations on pages 30, 31, 32, 33, 35, 61, and 63.

MQTT: The Standard for IoT Messaging. 2022. Https://mqtt.org. Accessed in 2022-04-20. Citation on page 49.

MUSADDIQ, A.; ZIKRIA, Y. B.; HAHM, O.; YU, H.; BASHIR, A. K.; KIM, S. W. A Survey on Resource Management in IoT Operating Systems. **IEEE Access**, v. 6, p. 8459–8482, feb 2018. ISSN 2169-3536. Available: <https://doi.org/10.1109/ACCESS.2018.2808324>. Citation on page 42.

NAGOWAH, S. D.; STA, H. B.; GOBIN-RAHIMBUX, B. A. An Overview of Semantic Interoperability Ontologies and Frameworks for IoT. In: **2018 Sixth International Conference on Enterprise Systems (ES)**. [s.n.], 2018. p. 82–89. Available: <https://doi.org/10.1109/ES.2018.00020>. Citations on pages 44 and 45.

NAHRSTEDT, K.; LI, H.; NGUYEN, P.; CHANG, S.; VU, L. Internet of Mobile Things: Mobility-Driven Challenges, Designs and Implementations. In: **IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)**. Berlin, Germany: IEEE, 2016. p. 25–36. ISBN 978-1-4673-9948-7. Available: <https://doi.org/10.1109/IoTDI.2015.41>. Citation on page 68.

NAIK, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: **IEEE International Systems Engineering Symposium (ISSE)**. Vienna, Austria: IEEE, 2017. ISBN 978-1-5386-3403-5. Available: <https://doi.org/10.1109/SysEng.2017.8088251>. Citations on pages 49 and 50.

NAOUI, S.; ELHDHILI, M. E.; SAIDANE, L. A. Security analysis of existing iot key management protocols. In: **2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)**. [s.n.], 2016. p. 7. Available: <https://doi.org/10.1109/AICCSA.2016.7945806>. Citation on page 59.

NESHENKO, N.; BOU-HARB, E.; CRICHIGNO, J.; KADDOUM, G.; GHANI, N. Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. **IEEE Communications Surveys Tutorials**, v. 21, n. 3, p. 2702–2733, 2019. Available: <https://doi.org/10.1109/COMST.2019.2910750>. Citation on page 58.

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**, v. 4, n. 1, p. 1–20, Feb 2017. ISSN 2372-2541. Available: <https://doi.org/10.1109/JIOT.2016.2615180>. Citation on page 41.

NI, J.; ZHANG, K.; LIN, X.; SHEN, X. Securing fog computing for internet of things applications: Challenges and solutions. **IEEE Communications Surveys Tutorials**, v. 20, n. 1, p. 601–628, 2018. Available: <https://doi.org/10.1109/COMST.2017.2762345>. Citation on page 55.

NING, H.; ZHEN, Z.; SHI, F.; DANESHMAND, M. A survey of identity modeling and identity addressing in internet of things. **IEEE Internet of Things Journal**, v. 7, n. 6, p. 4697–4710, 2020. Available: <https://doi.org/10.1109/JIOT.2020.2971773>. Citation on page 40.

Object Management Group. **Data Distribution Service (DDS) Version 1.4**. Needham, 2015. 1–20 p. Available: <http://www.omg.org/spec/DDS/1.4>. Citations on pages 49 and 50.

ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS. **OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0**. Boston, 2012. 124 p. Available: <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>. Citations on pages 49 and 50.

____. **MQTT Version 3.1.1**. Boston, 2014. 81 p. Available: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Citations on pages 49 and 132.

OUBBATI, O. S.; LAKAS, A.; ZHOU, F.; GüNEş, M.; YAGOUBI, M. B. A survey on position-based routing protocols for Flying Ad hoc Networks (FANETs). **Vehicular Communications**, v. 10, p. 29 – 56, 2017. ISSN 2214-2096. Available: <https://doi.org/10.1016/j.vehcom.2017.10.003>. Citation on page 35.

PAKROOH, R.; BOHLOOLI, A. A Survey on Unmanned Aerial Vehicles-Assisted Internet of Things: A Service-Oriented Classification. **Wirel. Pers. Commun.**, Springer Science and Business Media LLC, v. 119, p. 1541–1575, 2021. ISSN 1572-834X. Available: <https://doi.org/10.1016/j.comcom.2020.10.023>. Citations on pages 25 and 61.

PATWARY, A. A. N.; NAHA, R. K.; GARG, S.; BATTULA, S. K.; PATWARY, M. A. K.; AGHASIAN, E.; AMIN, M. B.; MAHANTI, A.; GONG, M. Towards secure fog computing: A survey on trust management, privacy, authentication, threats and access control. **Electronics (Switzerland)**, v. 10, n. 10, p. 1–52, 2021. ISSN 2079-9292. Available: <https://doi.org/10.3390/electronics10101171>. Citation on page 55.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context Aware Computing for The Internet of Things: A Survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 1, p. 414–454, 2014. ISSN 1553-877X. Available: <https://doi.org/10.1109/SURV.2013.042313.00197>. Citation on page 71.

PEREZ-GRAU, F.; RAGEL, R.; CABALLERO, F.; VIGURIA, A.; OLLERO, A. Semi-autonomous teleoperation of UAVs in search and rescue scenarios. In: **International Conference on Unmanned Aircraft Systems (ICUAS)**. Miami, USA: IEEE, 2017. p. 1066–1074. ISBN 978-1-5090-4495-5. Available: <https://doi.org/10.1109/ICUAS.2017.7991349>. Citation on page 30.

PERZ, R.; WRONOWSKI, K. UAV application for precision agriculture. **Aircraft Engineering and Aerospace Technology**, Emerald Group Publishing Limited, p. AEAT–01–2018–0056, oct 2018. ISSN 0002-2667. Available: <https://doi.org/10.1108/AEAT-01-2018-0056>. Citation on page 30.

PIGATTO, D. F. **HAMSTER - Healthy, mobility and security-based data communication architecture for Unmanned Aircraft Systems**. 201 p. Phd Thesis (PhD Thesis) — USP – Universidade de São Paulo, São Carlos, 2017. Available: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-11072017-085511/>. Citations on pages 26, 63, 75, 83, 141, and 157.

PIGATTO, D. F.; GONÇALVES, L.; ROBERTO, G. F.; Rodrigues Filho, J. F.; Floro da Silva, N. B.; PINTO, A. R.; Lucas Jaquie Castelo Branco, K. R. The HAMSTER Data Communication Architecture for Unmanned Aerial, Ground and Aquatic Systems. **Journal of Intelligent & Robotic Systems**, IEEE, v. 84, n. 1-4, p. 705–723, dec 2016. ISSN 0921-0296. Available: <https://doi.org/10.1007/s10846-016-0356-x>. Citations on pages 68 and 75.

POPLI, S.; JHA, R. K.; JAIN, S. A Survey on Energy Efficient Narrowband Internet of Things (NBIoT): Architecture, Application and Challenges. **IEEE Access**, IEEE, v. 7, p. 16739–16776, 2019. ISSN 21693536. Citation on page 49.

POURGHEBLEH, B.; HAYYOLALAM, V.; ANVIGH, A. A. Service discovery in the Internet of Things: review of current trends and research challenges. **Wireless Networks**, v. 26, p. 5371–5391, 2020. ISSN 1572-8196. Available: <https://doi.org/10.1007/s11276-020-02405-0>. Citation on page 44.

PULIAFITO, C.; MINGOZZI, E.; LONGO, F.; PULIAFITO, A.; RANA, O. Fog computing for the Internet of Things: A survey. **ACM Transactions on Internet Technology**, v. 19, n. 2, p. 41, 2019. ISSN 1557-6051. Citations on pages 54 and 55.

RAGHUNATHAN, V.; KANSAL, A.; HSU, J.; FRIEDMAN, J.; SRIVASTAVA, M. Design considerations for solar energy harvesting wireless embedded systems. In: **International Symposium on Information Processing in Sensor Networks (IPSN)**. Boise, USA: IEEE, 2005. p. 457–462. ISBN 0-7803-9201-9. Available: <https://doi.org/10.1109/IPSN.2005.1440973>. Citation on page 64.

RANGANATHAN, P. Recipe for efficiency. **Communications of the ACM**, ACM, New York, USA, v. 53, n. 4, p. 60–67, apr 2010. ISSN 0001-0782. Available: <https://doi.org/10.1145/1721654.1721673>. Citation on page 64.

RASTOGI, E.; SAXENA, N.; ROY, A.; SHIN, D. R. Narrowband internet of things: A comprehensive study. **Computer Networks**, v. 173, 2020. ISSN 1389-1286. Citation on page 49.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low power wide area networks: An overview. **IEEE Communications Surveys Tutorials**, v. 19, n. 2, p. 855–873, 2017. Citation on page 52.

RAZZAQUE, M. A.; MILOJEVIC-JEVRIC, M.; PALADE, A.; CLARKE, S. Middleware for internet of things: A survey. **IEEE Internet of Things Journal**, v. 3, n. 1, p. 70–95, 2016. Available: <https://doi.org/10.1109/JIOT.2015.2498900>. Citation on page 41.

RELIC. **RELIC Toolkit**. 2018. Available: <https://github.com/relic-toolkit>. Citation on page 133.

RIOT — The friendly Operating System for the Internet of Things. 2021. Https://www.riot-os.org. Accessed in 2021-12-20. Citation on page 42.

RODRIGUES, M.; AMARO, J.; OSORIO, F. S.; KALINKA, B. R. Authentication Methods for UAV Communication. **Proceedings - International Symposium on Computers and Communications**, 2019. ISSN 1530-1346. Citation on page 151.

RODRIGUES, M.; BRANCO, K. R. L. J. C. Uavs at your service: Towards iot integration with hamster. In: **2019 International Conference on Unmanned Aircraft Systems (ICUAS)**. [s.n.], 2019. p. 856–865. Available: <https://doi.org/10.1109/ICUAS.2019.8797790>. Citations on pages 27, 77, 88, and 156.

____. Cloud–SPHERE: Towards Secure UAV Service Provision. **Journal of Intelligent & Robotic Systems**, v. 97, p. 249–268, 2020. ISSN 1573-0409. Available: <https://doi.org/10.1007/s10846-019-01046-6>. Citations on pages 28, 70, 77, 88, 130, 131, 132, 134, 135, 136, 137, 138, and 157.

____. Enabling uav services in the iot with hamster. In: **2021 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2021. p. 1–6. Available: <https://doi.org/10.1109/ISCC53001.2021.9631461>. Citations on pages 28, 77, 87, 88, 143, 146, 147, 148, and 157.

____. Providing UAV Secure Services in the Internet of Things with HAMSTER: a Tutorial. [Manuscript submitted for publication]. 2022. Citations on pages 28, 76, 80, 86, 89, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 108, 109, 110, 111, 113, 114, 116, 117, 119, 147, 149, 150, 151, 156, and 157.

RODRIGUES, M.; PIGATTO, D. F.; BRANCO, K. R. L. J. C. Cloud-SPHERE: A Security Approach for Connected Unmanned Aerial Vehicles. In: **International Conference on Unmanned Aircraft Systems (ICUAS)**. Dallas, EUA: IEEE, 2018. p. 769–778. ISBN 978-1-5386-1354-2. Available: <https://doi.org/10.1109/ICUAS.2018.8453302>. Citations on pages 27, 77, 91, 123, 124, 125, 126, 127, 134, and 156.

____. Navigation phases platform: Towards green computing for uavs. In: **2018 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2018. p. 01171–01176. Available: <https://doi.org/10.1109/ISCC.2018.8538713>. Citations on pages 27, 77, 89, 127, 128, 129, 130, 131, and 156.

____. Context-aware operation for unmanned systems with hamster. In: **2020 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2020. p. 1–6. Available: <https://doi.org/10.1109/ISCC50000.2020.9219657>. Citations on pages 28, 77, 84, 135, 138, 139, 140, 141, and 156.

RODRIGUES, M.; PIGATTO, D. F.; FONTES, J. V. C.; PINTO, A. S. R.; DIGUET, J.-P.; BRANCO, K. R. L. J. C. UAV Integration Into IoIT : Opportunities and Challenges. **International Conference on Autonomic and Autonomous Systems (ICAS)**, IARIA, p. 6, 2017. Citation on page 27.

ROVIRA-SUGRANES, A.; RAZI, A.; AFGHAH, F.; CHAKARESKI, J. **A Review of AI-enabled Routing Protocols for UAV Networks: Trends, Challenges, and Future Outlook**. 2021. 30 p. Available: <https://arxiv.org/abs/2104.01283>. Citations on pages 33, 35, 38, and 39.

ROYO, P.; LOPEZ, J.; BARRADO, C.; PASTOR, E. Service Abstraction Layer for UAV Flexible Application Development. In: **AIAA Aerospace Sciences Meeting and Exhibit**. Reno, USA: American Institute of Aeronautics and Astronautics, 2008. p. 1–19. ISBN 978-1-62410-128-1. Available: <https://doi.org/10.2514/6.2008-484>. Citations on pages 65 and 66.

SABRI, C.; KRIAA, L.; AZZOUZ, S. L. Comparison of iot constrained devices operating systems: A survey. In: **IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)**. Hammamet, Tunisia: [s.n.], 2017. p. 369–375. ISSN 2161-5330. Available: <https://doi.org/10.1109/AICCSA.2017.187>. Citation on page 42.

SAHINGOZ, O. K. Networking Models in Flying Ad-Hoc Networks (FANETs): Concepts and Challenges. **Journal of Intelligent & Robotic Systems**, v. 74, n. 1-2, p. 513–527, apr 2014. Available: <https://doi.org/10.1007/s10846-013-9959-7>. Citations on pages 31 and 35.

SALMAN, T.; JAIN, R. **A Survey of Protocols and Standards for Internet of Things**. 2016. Available: <https://arxiv.org/abs/1903.11549>. Citations on pages 48 and 50.

SAMAILA, M. G.; NETO, M.; FERNANDES, D. A. B.; FREIRE, M. M.; INÁCIO, P. R. M. Challenges of securing Internet of Things devices: A survey. **Security and Privacy**, v. 1, n. 2, p. 32, mar 2018. ISSN 2475-6725. Available: <https://doi.org/10.1002/spy2.20>. Citation on page 58.

SHAABANA, A.; Fangyun Luo; Ying Chen; JAEKEL, A. Green networking strategies for distributed computing. In: **IEEE Consumer Communications and Networking Conference (CCNC)**. Las Vegas, USA: IEEE, 2014. p. 29–34. ISBN 978-1-4799-2355-7. Available: <https://doi.org/10.1109/CCNC.2014.6940503>. Citation on page 65.

SHARIAT, A.; TIZGHADAM, A.; LEON-GARCIA, A. An ICN-based publish-subscribe platform to deliver UAV service in smart cities. In: **IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. San Francisco, USA: IEEE, 2016. p. 698–703. ISBN 978-1-4673-9955-5. ISSN 0743166X. Available: <https://doi.org/10.1109/INFCOMW.2016.7562167>. Citation on page 70.

SHI, F.; LI, Q.; ZHU, T.; NING, H. A Survey of Data Semantization in Internet of Things. **Sensors**, MDPI, v. 18, n. 2, p. 313, jan 2018. ISSN 1424-8220. Available: <https://doi.org/10.3390/s18010313>. Citation on page 44.

SICARI, S.; RIZZARDI, A.; GRIECO, L.; COEN-PORISINI, A. Security, Privacy and Trust in Internet of Things: The Road Ahead. **Computer Networks**, Elsevier, v. 76, p. 146–164, jan 2015. ISSN 1389-1286. Available: <https://doi.org/10.1016/j.comnet.2014.11.008>. Citations on pages 56, 57, 68, 69, and 70.

SINGH, J.; PASQUIER, T.; BACON, J.; KO, H.; EYERS, D. Twenty Security Considerations for Cloud-Supported Internet of Things. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 3, p. 269–284, jun 2016. ISSN 2327-4662. Available: <https://doi.org/10.1109/JIOT.2015.2460333>. Citations on pages 42, 54, and 55.

SINGH, K.; VERMA, A. K. Threat modeling for multi-UAV Adhoc networks. In: **IEEE Region 10 Conference (TENCON)**. Penang, Malaysia: IEEE, 2017. p. 1544–1549. ISBN 978-1-5090-1134-6. ISSN 2159-3450. Citations on pages 15, 137, and 138.

____. Flying Adhoc Networks Concept and Challenges. In: Mehdi Khosrow-Pour, D. (Ed.). **Encyclopedia of Information Science and Technology, Fourth Edition**. IGI Global, 2018. chap. 530, p. 6106–6113. ISBN 9781522575986. Available: <https://doi.org/10.4018/978-1-5225-2255-3.ch530>. Citations on pages 33, 34, and 35.

SINGH, K. J.; KAPOOR, D. S. Create your own internet of things: A survey of iot platforms. **IEEE Consumer Electronics Magazine**, v. 6, n. 2, p. 57–68, April 2017. ISSN 2162-2256. Available: <https://doi.org/10.1109/MCE.2016.2640718>. Citation on page 41.

SINGH, S.; SHARMA, P. K.; MOON, S. Y.; PARK, J. H. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. **Journal of Ambient Intelligence and Humanized Computing**, 2017. ISSN 1868-5145. In press. Available: <https://doi.org/10.1007/s12652-017-0494-4>. Citation on page 59.

SOUZA, S. Castro de; FILHO, J. G. P. Semantic Interoperability in IoT: A Systematic Mapping. In: GALININA, O.; ANDREEV, S.; BALANDIN, S.; KOUCHERYAVY, Y. (Ed.). **Internet of Things, Smart Spaces, and Next Generation Networks and Systems**. St. Petersburg, Russia: Springer, Cham, 2019. (Lecture Notes in Computer Science, v. 11660), p. 53–64. Available: <https://doi.org/10.1007/978-3-030-30859-9_5>. Citation on page 45.

Sravanthi Chalasani; CONRAD, J. M. A survey of energy harvesting sources for embedded systems. In: **SoutheastCon**. Huntsville, USA: IEEE, 2008. p. 442–447. ISBN 978-1-4244-1883-1. Available: <https://doi.org/10.1109/SECON.2008.4494336>. Citation on page 64.

STELLIOS, I.; KOTZANIKOLAOU, P.; PSARAKIS, M.; ALCARAZ, C.; LOPEZ, J. A Survey of IoT-Enabled Cyberattacks: Assessing Attack Paths to Critical Infrastructures and Services. **IEEE Communications Surveys Tutorials**, v. 20, n. 4, p. 3453–3495, 2018. Available: <https://doi.org/10.1109/COMST.2018.2855563>. Citation on page 58.

STÖCKER, C.; BENNETT, R.; NEX, F.; GERKE, M.; ZEVENBERGEN, J. Review of the Current State of UAV Regulations. **Remote Sensing**, MDPI AG, Basel, v. 9, n. 5, p. 459, may 2017. ISSN 2072-4292. Available: <https://doi.org/10.3390/rs9050459>. Citation on page 32.

SZILAGYI, I.; WIRA, P. Ontologies and Semantic Web for the Internet of Things - A Survey. In: **42nd Annual Conference of the IEEE Industrial Electronics Society (IECON)**. Florence, Italy: IEEE, 2016. p. 6949–6954. ISBN 978-1-5090-3474-1. Available: <https://doi.org/10.1109/IECON.2016.7793744>. Citation on page 44.

TAHSIEN, S. M.; KARIMIPOUR, H.; SPACHOS, P. Machine learning based solutions for security of Internet of Things (IoT): A survey. **Journal of Network and Computer Applications**, v. 161, 2020. ISSN 1084-8045. Available: <https://doi.org/10.1016/j.jnca.2020.102630>. Citation on page 60.

TAN, Y.; WANG, J.; LIU, J.; ZHANG, Y. Unmanned Systems Security: Models, Challenges, and Future Directions. **IEEE Network**, IEEE, v. 34, n. 4, p. 291–297, jul 2020. ISSN 0890-8044. Available: <https://doi.org/10.1109/MNET.001.1900546>. Citation on page 36.

TEMDEE, P.; PRASAD, R. **Context-Aware Communication and Computing: Applications for Smart Environment**. Cham: Springer International Publishing, 2018. (Springer Series in Wireless Technology). ISBN 978-3-319-59034-9. Available: <https://doi.org/10.1007/978-3-319-59035-6>. Citations on pages 71 and 72.

THING, V. L.; WU, J. Autonomous Vehicle Security: A Taxonomy of Attacks and Defences. In: **IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)**. Chengdu, China: IEEE, 2016. p. 164–170. ISBN 978-1-5090-5880-8. Citation on page 82.

THOMAS, J. P.; QIDWAI, M. A.; KELLOGG, J. C. Energy scavenging for small-scale unmanned systems. **Journal of Power Sources**, Elsevier B.V., v. 159, n. 2, p. 1494–1509, sep 2006. ISSN 0378-7753. Available: <https://doi.org/10.1016/j.jpowsour.2005.12.084>. Citation on page 64.

THOUNTHONG, P.; RAËL, S.; DAVAT, B. Energy management of fuel cell/battery/supercapacitor hybrid power source for vehicle applications. **Journal of Power Sources**, Elsevier B.V., v. 193, n. 1, p. 376–385, aug 2009. ISSN 03787753. Available: <https://doi.org/10.1016/j.jpowsour.2008.12.120>. Citation on page 64.

TinyOS. 2022. Https://github.com/tinyos/tinyos-main. Accessed in 2022-04-20. Citation on page 42.

TRIANTAFYLLOU, A.; SARIGIANNIDIS, P.; LAGKAS, T. D. Network Protocols, Schemes, and Mechanisms for Internet of Things (IoT): Features, Open Challenges, and Trends. **Wireless Communications and Mobile Computing**, v. 2018, p. 24, 2018. ISSN 1530-8669. Available: <https://doi.org/10.1155/2018/5349894>. Citations on pages 48 and 52.

TSCHOFENIG, H. Fixing User Authentication for the Internet of Things (IoT). **Datenschutz und Datensicherheit**, Springer Fachmedien Wiesbaden, v. 40, n. 4, p. 222–224, apr 2016. ISSN 1614-0702. Available: <https://doi.org/10.1007/s11623-016-0582-1>. Citation on page 54.

TURNER, I. L.; HARLEY, M. D.; DRUMMOND, C. D. UAVs for coastal surveying. **Coastal Engineering**, Elsevier B.V., v. 114, p. 19–24, aug 2016. ISSN 03783839. Available: <https://doi.org/10.1016/j.coastaleng.2016.03.011>. Citation on page 30.

TURNER, R. M. Context-mediated behavior for intelligent agents. **Int. J. of Human-Computer Studies**, Elsevier Ltd., v. 48, n. 3, p. 307–330, mar 1998. ISSN 1071-5819. Citation on page 81.

UHER, J.; HARPER, J.; III, R. G. M.; PATTON, P.; FARROHA, B. Investigating end-to-end security in the fifth generation wireless capabilities and IoT extensions. In: TERNOVSKIY, I. V.; CHIN, P. (Ed.). **Cyber Sensing 2016**. SPIE, 2016. v. 9826, p. 51 – 66. Available: <https://doi.org/10.1117/12.2229608>. Citations on pages 42, 55, 57, 59, and 70.

VALAVANIS, K. P.; VACHTSEVANOS, G. J. **Handbook of Unmanned Aerial Vehicles**. Dordrecht: Springer Netherlands, 2015. ISBN: 9789048197064. Citations on pages 25, 29, and 30.

VANELLI, B.; RODRIGUES, M.; SILVA, M. P. da; PINTO, A.; DANTAS, M. A. R. A Proposed Architecture for Robots as a Service. In: BRANCO, K.; PINTO, A.; PIGATTO, D. (Ed.). **Communication in Critical Embedded Systems. WoCCES 2014, WoCCES 2015, WoCCES 2013, WoCCES 2016.** Cham: Springer, 2017, (Communications in Computer and Information Science, v. 702). chap. 7, p. 117–130. ISBN 978-3-319-61403-8. Available: <https://doi.org/10.1007/978-3-319-61403-8_7>. Citation on page 66.

VARSHNEY, P.; SIMMHAN, Y. Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions. **Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017**, n. 100, p. 115–124, 2017. Available: <https://doi.org/10.1109/ICFEC.2017.20>. Citations on pages 11 and 56.

VIGNAU, B.; KHOURY, R.; HALLé, S.; HAMOU-LHADJ, A. The evolution of IoT Malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives. **Journal of Systems Architecture**, v. 116, p. 102143, 2021. ISSN 1383-7621. Available: <https://doi.org/10.1016/j.sysarc.2021.102143>. Citation on page 59.

VILLAVERDE, B. C.; De Paz Alberola, R.; JARA, A. J.; FEDOR, S.; DAS, S. K.; PESCH, D. Service Discovery Protocols for Constrained Machine-to-Machine Communications. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 1, p. 41–60, 2014. ISSN 1553-877X. Available: <https://doi.org/10.1109/SURV.2013.102213.00229>. Citation on page 51.

WANT, R.; SCHILIT, B. N.; JENSON, S. Enabling the Internet of Things. **Computer**, IEEE, v. 48, n. 1, p. 28–35, jan 2015. ISSN 0018-9162. Available: <https://doi.org/10.1109/MC.2015.12>. Citation on page 47.

WASHINGTON, A.; CLOTHIER, R. A.; SILVA, J. A review of unmanned aircraft system ground risk models. **Progress in Aerospace Sciences**, v. 95, p. 24 – 44, 2017. ISSN 0376-0421. Citation on page 140.

WHITE, G.; NALLUR, V.; CLARKE, S. Quality of service approaches in IoT: A systematic mapping. **Journal of Systems and Software**, Elsevier Inc., v. 132, p. 186–203, oct 2017. ISSN 0164-1212. Available: <https://doi.org/10.1016/j.jss.2017.05.125>. Citation on page 44.

WHITMORE, A.; AGARWAL, A.; Da Xu, L. The Internet of Things — A Survey of Topics and Trends. **Information Systems Frontiers**, v. 17, n. 2, p. 261–274, apr 2015. ISSN 1387-3326. Available: <https://doi.org/10.1007/s10796-014-9489-2>. Citations on pages 25, 39, 57, and 58.

WRONA, K.; GOMEZ, L. Context-aware security and secure context-awareness in ubiquitous computing environments. **Annales Universitatis Mariae Curie-Skłodowska, sectio AI – Informatica**, Wydawnictwo Uniwersytetu Marii Curie-Skłodowskiej, v. 4, n. 1, p. 332–348, 2006. Citations on pages 72 and 81.

XIAO, Z.; XIAO, Y. Security and Privacy in Cloud Computing. **IEEE Communications Surveys & Tutorials**, v. 15, n. 2, p. 843–859, 2013. ISSN 1553-877X. Available: <https://doi.org/10.1109/SURV.2012.060912.00182>. Citations on pages 68, 69, and 70.

YAACOUB, J.-P.; NOURA, H.; SALMAN, O.; CHEHAB, A. Security analysis of drones systems: Attacks, limitations, and recommendations. **Internet of Things**, v. 11, 2020. ISSN 2542-6605. Available: <https://doi.org/10.1016/j.iot.2020.100218>. Citation on page 36.

YAHUZA, M.; IDRIS, M. Y. I.; AHMEDY, I. B.; WAHAB, A. W. A.; NANDY, T.; NOOR, N. M.; BALA, A. Internet of Drones Security and Privacy Issues: Taxonomy and Open Challenges. **IEEE Access**, IEEE, v. 9, p. 57243–57270, 2021. ISSN 2169-3536. Available: <https://doi.org/10.1109/ACCESS.2021.3072030>. Citation on page 36.

YAPP, J.; SEKER, R.; BABICEANU, R. UAV as a service: Enabling on-demand access and on-the-fly re-tasking of multi-tenant UAVs using cloud services. In: **2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)**. Sacramento, CA, USA: IEEE, 2016. p. 1–8. ISBN 978-1-5090-2523-7. Available: <https://doi.org/10.1109/DASC.2016.7778007>. Citations on pages 62 and 67.

YIN, J.; YANG, Z.; CAO, H.; LIU, T.; ZHOU, Z.; WU, C. A Survey on Bluetooth 5.0 and Mesh. **ACM Transactions on Sensor Networks**, v. 15, n. 3, p. 1–29, may 2019. ISSN 15504859. Available: <https://doi.org/10.1145/3317687>. Citation on page 53.

YOUSEFPOUR, A.; FUNG, C.; NGUYEN, T.; KADIYALA, K.; JALALI, F.; NIAKANLAHIJI, A.; KONG, J.; JUE, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, Elsevier B.V., v. 98, n. December 2018, p. 289–330, sep 2019. ISSN 1383-7621. Available: <https://doi.org/10.1016/j.sysarc.2019.02.009>. Citations on pages 54, 55, and 60.

YU, H.; YANG, W.; ZHANG, H.; HE, W. A UAV-based crack inspection system for concrete bridge monitoring. In: **EEE International Geoscience and Remote Sensing Symposium (IGARSS)**. Fort Worth, USA: IEEE, 2017. p. 3305–3308. ISBN 978-1-5090-4951-6. Available: <https://doi.org/10.1109/IGARSS.2017.8127704>. Citation on page 30.

YUAN, E.; ESFAHANI, N.; MALEK, S. A Systematic Survey of Self-Protecting Software Systems. **ACM Transactions on Autonomous and Adaptive Systems**, v. 8, n. 4, p. 41, jan 2014. ISSN 1556-4665. Citation on page 72.

ZENG, Y.; ZHANG, R.; LIM, T. J. Wireless communications with unmanned aerial vehicles: opportunities and challenges. **IEEE Communications Magazine**, IEEE, v. 54, n. 5, p. 36–42, may 2016. ISSN 0163-6804. Available: <https://doi.org/10.1109/MCOM.2016.7470933>. Citations on pages 64 and 65.

ZERMANI, S.; DEZAN, C.; HIRECHE, C.; EULER, R.; DIGUET, J.-P. Embedded context aware diagnosis for a UAV SoC platform. **Microprocessors and Microsystems**, Elsevier B.V., v. 51, p. 185–197, jun 2017. ISSN 0141-9331. Citation on page 72.

ZHOU, G.; YUAN, J.; YEN, I.-L.; BASTANI, F. Robust real-time UAV based power line detection and tracking. In: **IEEE International Conference on Image Processing (ICIP)**. Phoenix, USA: IEEE, 2016. p. 744–748. ISBN 978-1-4673-9961-6. Available: <https://doi.org/10.1109/ICIP.2016.7532456>. Citation on page 30.

ZHOU, M.; ZHANG, R.; XIE, W.; QIAN, W.; ZHOU, A. Security and Privacy in Cloud Computing: A Survey. In: **International Conference on Semantics, Knowledge and Grids**. Beijing, China: IEEE, 2010. v. 2, p. 105–112. ISBN 978-1-4244-8125-5. ISSN 21945357. Available: <https://doi.org/10.1109/SKG.2010.19>. Citation on page 68.

ZHOU, Y.; RAO, B.; WANG, W. Uav swarm intelligence: Recent advances and future trends. **IEEE Access**, v. 8, p. 183856–183878, 2020. Available: <https://doi.org/10.1109/ACCESS.2020.3028865>. Citation on page 38.

ZHU, X.; BADR, Y. Identity Management Systems for the Internet of Things: A Survey Towards Blockchain Solutions. **Sensors**, MDPI, v. 18, n. 12, p. 4215, dec 2018. ISSN 1424-8220. Available: <https://doi.org/10.3390/s18124215>. Citation on page 40.

APPENDIX

A

PROOF-OF-CONCEPT LOGS

A.1 PoC 6: UAV Data Gathering Service

A.1.0.1 UAV log

```
Unit started loading.
Service START_MISSION registered successfully.
Service MISSION_FINISHED registered successfully.
Authentication .successful():  for Peer 4.
Message AUTHENTICATION_REQUEST sent to Peer ID 5
Message AUTHENTICATION_RESPONSE_OK received in CSU.
Message AUTHENTICATION_MESSAGE sent to Peer ID 5
Message AUTHENTICATION_MESSAGE received in CSU.
Authentication .successful():  for Peer 3.
Message AUTHENTICATION_MESSAGE received in CSU.
Message AUTHENTICATION_MESSAGE sent to Peer ID 5
Authentication .successful():  for Peer 5.
Unit finished loading.
Message SERVICE_REQUEST_OK for Service ID POSITION_ACHIEVED received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID COLLECTION_RESULT received in SEO from peer 4.
Message SERVICE_REQUEST_OK for Service ID X_POSITION received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID Y_POSITION received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID Z_POSITION received in SEO from peer 3.
Message SERVICE_REQUEST for Service ID START_MISSION received in SEO from peer 5.
NP: New State Number: 1.
Message CURRENT_STATE sent to Peer ID 3
Message CURRENT_STATE sent to Peer ID 4
Message SERVICE_REQUEST for Service ID MISSION_FINISHED received in SEO from peer 5.
Message SERVICE_REQUEST_OK for Service ID NAVIGATE received in SEO from peer 3.
Message SERVICE_NOTIFICATION for Service ID POSITION_ACHIEVED received in SEO from peer 3.
NP: New State Number: 2.
Message CURRENT_STATE sent to Peer ID 3
Message CURRENT_STATE sent to Peer ID 4
Message SERVICE_REQUEST_OK for Service ID NAVIGATE received in SEO from peer 3.
Message SERVICE_NOTIFICATION for Service ID POSITION_ACHIEVED received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID START_COLLECTION received in SEO from peer 4.
Message SERVICE_NOTIFICATION for Service ID COLLECTION_RESULT received in SEO from peer 4.
NP: New State Number: 4.
```

```
Message CURRENT_STATE sent to Peer ID 3
Message CURRENT_STATE sent to Peer ID 4
Message SERVICE_REQUEST_OK for Service ID NAVIGATE received in SEO from peer 3.
Message SERVICE_NOTIFICATION for Service ID POSITION_ACHIEVED received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID START_COLLECTION received in SEO from peer 4.
Message SERVICE_NOTIFICATION for Service ID COLLECTION_RESULT received in SEO from peer 4.
NP: New State Number: 6.
Message CURRENT_STATE sent to Peer ID 3
Message CURRENT_STATE sent to Peer ID 4
Message SERVICE_REQUEST_OK for Service ID NAVIGATE received in SEO from peer 3.
Message SERVICE_NOTIFICATION for Service ID POSITION_ACHIEVED received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID START_COLLECTION received in SEO from peer 4.
Message SERVICE_NOTIFICATION for Service ID COLLECTION_RESULT received in SEO from peer 4.
NP: New State Number: 7.
Message CURRENT_STATE sent to Peer ID 3
Message CURRENT_STATE sent to Peer ID 4
Message SERVICE_REQUEST_OK for Service ID NAVIGATE received in SEO from peer 3.
Message SERVICE_NOTIFICATION for Service ID POSITION_ACHIEVED received in SEO from peer 3.
Message SERVICE_REQUEST_OK for Service ID NAVIGATE received in SEO from peer 3.
Message SERVICE_NOTIFICATION for Service ID POSITION_ACHIEVED received in SEO from peer 3.
NP.finishTask(): Task Finished.
Message TASK_FINISHED sent to Peer ID 3
Message TASK_FINISHED sent to Peer ID 4
Unit destroyed
```

A.1.0.2 Navigation Unit log

```
Unit started loading.
Service X_POSITION registered successfully.
Service Y_POSITION registered successfully.
Service Z_POSITION registered successfully.
Service NAVIGATE registered successfully.
Service POSITION_ACHIEVED registered successfully.
Authentication .successful():  for Peer 1.
Unit finished loading.
Message SERVICE_REQUEST for Service ID POSITION_ACHIEVED received in SEO from peer 1.
Message SERVICE_REQUEST for Service ID X_POSITION received in SEO from peer 1.
Message SERVICE_REQUEST for Service ID Y_POSITION received in SEO from peer 1.
Message SERVICE_REQUEST for Service ID Z_POSITION received in SEO from peer 1.
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 0, y = 0, z = 2
New Position: x = 0, y = 0, z = 6
New Position: x = 0, y = 0, z = 10
New Position: x = 2.4, y = 1, z = 10
New Position: x = 7.2, y = 3, z = 10
New Position: x = 12, y = 5, z = 10
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 12, y = 5, z = 9
New Position: x = 12, y = 5, z = 5
Message SERVICE_NOTIFICATION sent to Peer ID 1
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 12, y = 5, z = 6
New Position: x = 12, y = 5, z = 7
New Position: x = 12, y = 5, z = 8
New Position: x = 12, y = 5, z = 10
New Position: x = 13.2, y = 5.8, z = 10
New Position: x = 14.1, y = 6.4, z = 10
New Position: x = 15, y = 7, z = 10
Message SERVICE_NOTIFICATION sent to Peer ID 1
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 15, y = 7, z = 9
New Position: x = 15, y = 7, z = 7
New Position: x = 15, y = 7, z = 5
Message SERVICE_NOTIFICATION sent to Peer ID 1
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 15, y = 7, z = 6
New Position: x = 15, y = 7, z = 8
New Position: x = 15, y = 7, z = 10
New Position: x = 16, y = 8, z = 10
New Position: x = 18, y = 10, z = 10
New Position: x = 20, y = 12, z = 10
Message SERVICE_NOTIFICATION sent to Peer ID 1
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 20, y = 12, z = 9
New Position: x = 20, y = 12, z = 7
New Position: x = 20, y = 12, z = 5
Message SERVICE_NOTIFICATION sent to Peer ID 1
NavUnitState ON.
Message SERVICE_REQUEST for Service ID NAVIGATE received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
New Position: x = 20, y = 12, z = 6
New Position: x = 20, y = 12, z = 8
New Position: x = 20, y = 12, z = 10
New Position: x = 14, y = 8.4, z = 10
New Position: x = 8, y = 4.8, z = 10
New Position: x = 3.55271e-15, y = 8.88178e-16, z = 10
New Position: x = 3.55271e-15, y = 8.88178e-16, z = 6
New Position: x = 3.55271e-15, y = 8.88178e-16, z = 2
New Position: x = 3.55271e-15, y = 8.88178e-16, z = 0
Message SERVICE_NOTIFICATION sent to Peer ID 1
Message TASK_FINISHED_ACK sent to Peer ID 1
```

A.1.0.3 Collection Unit log

```
Unit started loading.
Service START_COLLECTION registered successfully.
Service COLLECTION_RESULT registered successfully.
Authentication .successful(): for Peer 1.
Unit finished loading.
Message SERVICE_REQUEST for Service ID COLLECTION_RESULT received in SEO from peer 1.
Message SERVICE_REQUEST_OK sent to Peer ID 1
CollectionUnitState OFF.
CollectionUnitState ON.
Message SERVICE_REQUEST for Service ID START_COLLECTION received in SEO from peer 1.
Authentication .successful(): for Peer 20.
Message SERVICE_REQUEST_OK sent to Peer ID 1
Message SERVICE_REQUEST_OK for Service ID COLLECTION_FINISHED received in SEO from peer 20.
Message SERVICE_REQUEST_OK for Service ID COLLECT_DATA received in SEO from peer 20.
Message SERVICE_NOTIFICATION for Service ID COLLECTION_FINISHED received in SEO from peer 20.
Message SERVICE_NOTIFICATION sent to Peer ID 1
CollectionUnitState OFF.
CollectionUnitState ON.
Message SERVICE_REQUEST for Service ID START_COLLECTION received in SEO from peer 1.
Authentication .successful(): for Peer 30.
Message SERVICE_REQUEST_OK sent to Peer ID 1
Message SERVICE_REQUEST_OK for Service ID COLLECTION_FINISHED received in SEO from peer 30.
Message SERVICE_REQUEST_OK for Service ID COLLECT_DATA received in SEO from peer 30.
Message SERVICE_NOTIFICATION for Service ID COLLECTION_FINISHED received in SEO from peer 30.
Message SERVICE_NOTIFICATION sent to Peer ID 1
CollectionUnitState OFF.
CollectionUnitState ON.
Message SERVICE_REQUEST for Service ID START_COLLECTION received in SEO from peer 1.
Authentication .successful(): for Peer 40.
Message SERVICE_REQUEST_OK sent to Peer ID 1
Message SERVICE_REQUEST_OK for Service ID COLLECTION_FINISHED received in SEO from peer 40.
Message SERVICE_REQUEST_OK for Service ID COLLECT_DATA received in SEO from peer 40.
Message SERVICE_NOTIFICATION for Service ID COLLECTION_FINISHED received in SEO from peer 40.
Message SERVICE_NOTIFICATION sent to Peer ID 1
CollectionUnitState OFF.
```

A.1.0.4 Cloud Server log

```
Unit started loading.
Unit finished loading.
Message AUTHENTICATION_REQUEST received in CSU.
Message AUTHENTICATION_RESPONSE_OK sent to Peer ID 1
Message AUTHENTICATION_MESSAGE received in CSU.
Message AUTHENTICATION_MESSAGE sent to Peer ID 1
Message AUTHENTICATION_MESSAGE received in CSU.
Authentication.successful(): for Peer 1.
Message SERVICE_REQUEST sent to Peer ID 1
Message SERVICE_REQUEST sent to Peer ID 1
Message SERVICE_REQUEST_OK for Service ID START_MISSION received in SEO from peer 1.
Message SERVICE_REQUEST_OK for Service ID MISSION_FINISHED received in SEO from peer 1.
Message SERVICE_NOTIFICATION for Service ID MISSION_FINISHED received in SEO from peer 1.
Mission Finished
```

A.1.0.5 Sensors log

```
Unit started loading.
Service COLLECT_DATA registered successfully.
Service COLLECTION_FINISHED registered successfully.
Unit finished loading.
Authentication .successful():  for Peer 4.
Message SERVICE_REQUEST for Service ID COLLECTION_FINISHED received in SEO from peer 4.
Message SERVICE_REQUEST_OK sent to Peer ID 4
Message SERVICE_REQUEST for Service ID COLLECT_DATA received in SEO from peer 4.
Message SERVICE_REQUEST_OK sent to Peer ID 4
Message SERVICE_NOTIFICATION sent to Peer ID 4
Unit destroyed
```

ICMC USP
SÃO CARLOS