FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Gestão e Controlo de um Robô Móvel através do Protocolo VDA5050

Marina Rodrigues Brilhante

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Pedro Gomes da Costa

July 31, 2023

Resumo

Manifestamente os robôs são utilizados numa variedade de funções e participam ativamente na cadeia de processamento de uma indústria. Muitas vezes, são necessários diferentes tipos de veículos de vários fornecedores para um fluxo de trabalho completo, originando um problema associado ao controlo e a gestão dessas frotas heterogéneas de uma forma eficiente. Para resolver este problema de interoperabilidade, a norma VDA 5050 propõe uma interface normatizada para a comunicação de informações sobre ordens e status entre os robôs e o controlo principal central.

Usando a norma VDA 5050, pretende-se apresentar um módulo de software capaz de gerir e controlar um robô móvel autónomo desenvolvido pelo CRIIS do INESC TEC e comunicar com o seu sistema de gestão de frotas de robôs baseado em ROS. Este módulo é composto por dois nós de tradução, desenvolvidos em ROS, e uma aplicação cliente MQTT para troca de mensagens. Os resultados confirmam que o módulo pode enviar corretamente as mensagens via MQTT e que o robô recebe e executa corretamente a ordem.

Palavras-chave: Norma VDA5050, protocolo MQTT, interoperabilidade, robôs móveis, sistema de gestão de frotas, frota heterogénea.

Abstract

It is well known that robots are used in various roles and actively participate in the process chain of an industry. Different types of vehicles from various vendors are often needed to accomplish a complete workflow, creating a problem associated with efficiently controlling and managing such heterogeneous fleets. The VDA 5050 standard proposes a standardized interface for communicating order and status information between AGVs and master control to address the interoperability problem.

Using the VDA 5050 standard, this research presents a software module capable of managing and controlling an autonomous mobile robot developed by INESC TEC's CRIIS and communicating with its ROS-based robot fleet management system. This module consists of two translation nodes, developed in ROS and a MQTT Client Application for exchanging messages. The results confirm that the module can correctly send the messages via MQTT and that the robot correctly receives and executes the order.

Keywords: VDA5050 standard, MQTT protocol, interoperability, mobile robots, fleet management system, heterogeneous fleet.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Pedro Costa, for overseeing the entire development process of this thesis and always providing help when needed. I'm also very grateful to Paulo Rebelo, researcher at INESC TEC's iiLab, for all the great help and encouragement I received at every step of this process, and to INESC TEC researcher Héber Sobreira for all his help and guidance since the beginning.

I would like to express my sincere gratitude to the entire team at INESC TEC's CRIIS, who were always ready to help and provided a positive working environment. I would also like to thank all the people I have met during my studies at FEUP, who have helped me in one way or another to get to this point.

Finally, I would like to thank my family for all their support and good wishes, especially my parents and my sister, who have been very supportive and motivating not only in this last stage of my studies, but also in the last five years.

Marina Brilhante

Contents

Acknowledgments v

Abbreviations and Symbols xiii

1 Introduction 1

2 Bibliography Review 3
2.1 MQTT and Machine-to-Machine/IoT systems . . . 3
2.2 Multi-robot Systems . . . 4
2.2.1 Centralized and Decentralized Control . . . 5
2.2.2 Homogeneous and Heterogeneous Systems . . . 6
2.3 Fleet Management Systems . . . 6
2.3.1 Meili Robots - Meili FMS . . . 6
2.3.2 InOrbit . . . 6
2.4 Interoperability . . . 7
2.5 Interoperability Efforts . . . 8
2.5.1 VDA 5050 Standard . . . 8
2.5.2 Open Robotics Middleware Framework (OpenRMF) . . . 9
2.5.3 MassRobotics Interoperability Standard . . . 10
2.5.4 Open Platform Communications Unified Architecture (OPC UA) . . . 11
2.6 Integration of Information Systems with Fleet Management Systems . . . 12
2.7 Conclusion . . . 13

3 VDA 5050 Standard 15
3.1 Communication Procedure . . . 15
3.2 VDA 5050 Subtopics for Communication . . . 16
3.2.1 Order . . . 17
3.2.2 Instant Actions . . . 17
3.2.3 State . . . 18
3.2.4 Connection, Fact sheet, and Visualization . . . 19
3.3 Field Testing . . . 21
3.4 Conclusion . . . 22

4 Implementation 23
4.1 Introduction . . . 23
4.2 Tools and Technology Used . . . 23
4.3 Software Architecture . . . 25
4.4 Implementation Details . . . 27
4.4.1 Translation Nodes . . . 27

4.4.2 MQTT Client Application 30
4.5 Testing and Conclusion 33

5 Results and Discussion **35**
5.1 Tests and Results 35
5.1.1 Simulated Environment Tests 35
5.1.2 Real-life Scenario Tests 38
5.2 Discussion 38

6 Conclusion and Future Work **43**

List of Figures

2.1 Diagram of Meili FMS features. Adapted from [1] . . . 7
2.2 The diagram illustrating how OpenRMF communicates with the robots. Adapted from [2]. . . . 10
2.3 Overview of the OpenRMF architecture. Adapted from [2] . . . 10
2.4 Information flow in the MassRobotics Interoperability Standard. Adapted from [3]. 11
2.5 Overview of the OPC UA Architecture. Adapted from [4] . . . 12

3.1 Overview of the information flow during the process. Adapted from [5]. . . . 16
3.2 Graph transmitted to the robot in an Order . . . 18
3.3 Example of a VDA 5050 Order message in JSON format . . . 18
3.4 Example of a VDA 5050 Instant Action message in JSON format . . . 19
3.5 State topic fields during order handling . . . 19
3.6 Example of a VDA 5050 State message in JSON format . . . 20
3.7 Example of a VDA Connection message in JSON format . . . 20
3.8 Example of a VDA Factsheet message in JSON format . . . 21
3.9 Mobile robots that participated in the test performed at the AGV Mesh-Up. Adapted from [6]. . . . 21

4.1 Navigation Stack Simulation Environment in Rviz . . . 24
4.2 Illustration about the process after receiving a new order . . . 26
4.3 Depiction of the exchange of information among INS Nodes and Topics . . . 26
4.4 Parallel between the Navigation Stack and the VDA5050 Standard . . . 27
4.5 Diagram illustrating the exchange of messages to the MQTT broker between the Master Control (INS Nodes) and AGV . . . 27
4.6 Diagram representing the exchange of messages between the Navigation Stack and the Translation Nodes . . . 28
4.7 Diagram illustrating the arrangement for the MQTT Client Application and the communication with the MQTT Broker . . . 30
4.8 Configuration File associated with a supervisor and two robots . . . 31
4.9 Contents of the MQTT Packages Connect, Publish and Subscribe, sent to the MQTT broker . . . 33

5.1 VDA 5050 Connection Messages seen through MQTTX . . . 36
5.2 Robot path to Node 10, representation of nodes and edges . . . 37
5.3 VDA 5050 Order Message seen through MQTTX . . . 37
5.4 Terminal screenshot showing the two published messages of type *target_route* . . 38
5.5 VDA State Messages seen through MQTTX . . . 39
5.6 VDA-compliant messages published by the translation node on the right, compared to messages published by the Navigation Stack during order execution . . . 40

5.7 Stack-compliant messages published by the translation node on the right in comparison to messages published by the Navigation Stack 40
5.8 Map used in the real-life scenario test 41

List of Tables

3.1 VDA 5050 Subtopics for Communication 17

4.1 Indicates the ROS and MQTT topics published and subscribed by the supervisor and robots 31

Abbreviations and Symbols

AGV	Autonomous Guided Vehicle
AMR	Autonomous Mobile Robots
API	Application Programming Interface
ERP	Enterprise Resource Planning
FMS	Fleet Management System
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
MES	Manufacturing Execution Systems
MQTT	Message Queuing Telemetry Transport
MRS	Multi-Robot System
RMF	Robotics Middleware Framework
ROS	Robot Operating System
TEA*	Time Enhanced A*
WCS	Warehouse Control System
WES	Warehouse Execution System
WMS	Warehouse Management System

Chapter 1

Introduction

Today, autonomous mobile robots (AMRs) and autonomous guided vehicles (AGVs), among others, are increasingly used in warehouses and distribution centers, healthcare facilities, security robots, industrial environments, and many other applications. These mobile robots improve operational efficiency and productivity, increase speed, safety, and flexibility, and ensure precision.

The widespread use of robots in the industry has stimulated the emergence of numerous vendors in the mobile robot market who, in turn, develop different vehicles with a high degree of specialization. In practice, it is usually necessary for several robots to work together, and it is often difficult for a single manufacturer to provide vehicles that can perform all tasks. As a result, the working fleet is usually composed of AMRs or AGVs from different manufacturers. However, only their proprietary fleet management software controls these robots.

These heterogeneous working fleets, consisting of different types of vehicles from multiple vendors, lead to implementation difficulties as each vehicle requires individual integration efforts, traffic control and route management problems, and communication and interoperability issues. As a result, there is a growing need for a standardized interface that can communicate, control and manage a diverse fleet without these interoperability issues.

One of the initiatives to address this lack of interoperability began in Germany, with two companies working together to develop VDA 5050, an open standard for communication between AGV fleets and central master control. Simply put, the standard defines messages of order and state to be exchanged between AGVs and fleet management systems that aim to enable AGVs from different manufacturers and conventional systems to operate in parallel in the same working environment. The aim of the study carried out in the context of this thesis is to design and implement a software module capable of communicating with and controlling an AMR developed entirely by INESC TEC's Center for Robotics in Industry and Intelligent Systems (CRIIS) using the VDA 5050 standard.

This research will contribute to a better understanding of the VDA 5050 standard and how the standard establishes the standardized interface for communication between AGVs. It will also provide context on how VDA 5050 can be implemented in INESC TEC's fleet management system and robots and how the VDA 5050 order and state messages can be assembled from the

information given by the fleet manager.

The thesis is structured as follows: Chapter 1 introduces the study context and motivation. Chapter 2 presents essential theoretical concepts and the state of the art on mobile robots, fleet management systems, and interoperability standards. Chapter 3 focuses on the VDA 5050 standard, its particularities, and the current studies related to it. Chapter 4 explores the implementation of the software module, along with the technologies and tools used. Chapter 5 explains the results obtained and discusses their implications. Chapter 6 will conclude the research with the critical points explored and touch on future work to develop.

Chapter 2

Bibliography Review

Over the past century, the industry has transformed operations, leading to innovation and profound social and economic change. Today, the industrial sector is experiencing its fourth revolution, Industry 4.0. It is characterized by increasing automation, intelligent machines and smart factories, and technologies like the Internet of Things (IoT), cloud computing, AI and machine learning, Machine-to-Machine (M2M) communication, and cyber-physical systems.

These smart factories can make informed decisions by collecting and analyzing data from advanced sensors, embedded software, and other intelligent machines on the production floor [7]. This digital transformation and intelligent automation lead to unprecedented efficiency and responsiveness to customers. Smart factories are characterized by their flexibility in highly challenging and dynamic situations. They can produce customized products that meet individual customer needs more cost-effectively. While the Second Industrial Revolution is known for introducing mass production using electricity, the Fourth Industrial Revolution is about mass customization.

2.1 MQTT and Machine-to-Machine/IoT systems

With product customization increasing, the need for flexible production systems rises. Mobile robots, including AGVs and AMRs, are characterized by their collaboration ability, increasing production flexibility, and becoming a critical enabling technology for agile production systems [8]. In addition, AGVs have further evolved into more intelligent autonomous mobile robots in anticipation of the arrival of the smart factory era.

Because AGVs and AMRs handle internal operations, their cooperation with other information systems and manufacturing equipment is crucial. In favor of easily integrating multi-robot systems with the production environment, Machine-to-Machine (M2M) communication can be applied. These robots must be able to interact with workstations, other vehicles, and higher-level factory systems to update their location and obtain details on their next destination [9].

M2M and IoT (Internet of Things) are two technologies frequently mentioned when discussing device networking. They aim to link smart devices for data collection, transmission, and remote

monitoring with little human intervention. In IoT and M2M systems, there are several communication protocols for data transmission, MQTT being one of them.

In 1999 the MQTT protocol was invented for use in the oil and gas industry. The objective was to create a protocol that could transfer a message reliably even if there is a low-bandwidth network condition and long network delay [10]. It is lightweight and efficient, allows bi-directional and asynchronous communications, and uses a publisher/subscriber communication pattern [11]. These characteristics make it ideal for use in many situations, including limited operating environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) [12].

The core of MQTT is the MQTT Broker and the MQTT Client. A client can be a sender (publisher) and a receiver (subscriber) and communicate via separate topics. The broker handles the connection between them by filtering all incoming messages and distributing them correctly to the subscribers. The clients will use the broker to send messages rather than communicate directly with one another.

There are many open-source clients available in a variety of programming languages. One of the most popular clients, Eclipse Paho [13], provides C/C++ implementations that are easy to use and can fully implement MQTT subscriptions and messaging with a small amount of code. In the same way, MQTT brokers are offered in open source, commercial implementations, and managed cloud services. In 2012, a year later after the publication of MQTT 3.1 as a free and open protocol, Eclipse Mosquitto [14], the first open-source MQTT broker, was released.

2.2 Multi-robot Systems

Multi-robot Systems (MRSs) have emerged as a suitable alternative to single-robot systems to improve and enable new chores. MRSs are a set of robots created to perform some collective activity [15]. The many advantages of MRSs over standalone robot systems have been discussed in the robotics world for a while. These advantages are [16]:

- **Effectiveness:** Using a fleet of homogeneous robots improves the performance obtained by a single one. An MRS can cover a larger area or spend less time on a task.

- **Efficiency:** In various missions, using heterogeneous fleets results in more efficient management of resources than using a single robot.

- **Flexibility:** A multi-robot system can better adapt to changes. Changes such as the search area size, tasks, or robots with difficulties during missions.

- **Fault tolerance and robustness:** If a robot experiences problems, such as malfunctioning, getting trapped, or using up all its battery, the rest of the fleet can continue without affecting task completion.

These advantages have drawn academic and business researchers to explore the application of MRSs in numerous fields like intelligent security [17], search and rescue [18], surveillance [19], and environment monitoring [20].

Complex issues must be resolved to create and implement strong MRS in real-world applications. The multi-robot task allocation (MRTA) problem is one of the most difficult MRS problems, particularly when it comes to heterogeneous unreliable robots equipped with various types of sensors and actuators that must carry out various tasks with multiple requirements and constraints in an optimal way [15].

MRTA strategies may be categorized based on the organizational team perspective. This perspective attests to how the various robots/agents of the system are structured by describing the interactions between the agents and the specific functions of each agent within the system. The following subsection describes centralized and decentralized organizational views.

2.2.1 Centralized and Decentralized Control

In centralized architectures, each agent is connected to a central control agent. The individual agents send all information to the central control agent, which processes the information and participates in the decision-making process, communicating within the fleet of robots and assigning tasks to the robots.

Since the central control has a global view of the processes, it can produce a globally optimal plan and consequently reduce duplication of effort and resources and increase savings of cost and time [15]. This architecture is typical for a small number of robots and ineffective for large teams; it is not robust regarding environmental changes or communication failures and is highly vulnerable since, if the central control agent malfunctions, the entire fleet cannot continue.

Many centralized approaches are proposed to solve the MRTA problem. In [21], an implementation based on A* algorithms is proposed for centralized multi-robot task allocation for industrial plant inspection. Also, [22] presents a centralized algorithm to solve the task allocation problem of assigning mobile robots to static sensors.

In the decentralized approach, there is no central control agent to allocate tasks to other agents, and all the robots are completely autonomous in the decision-making process. In this architecture, each agent communicates its information with the other agents. In general, decentralized approaches have many advantages over centralized systems, including that the vehicles can better respond to unknown or changing environments and have improved flexibility, scalability, adaptability, and robustness to single-agent failures. It also has some disadvantages related to the control and coordination of multi-robot systems in real-world settings [23] because of the complexity of the system and the need for good communication between robots.

There have been many proposals for decentralized approaches to the MRTA problem. The work in [24] describes a production planning and scheduling decentralized framework that solves the task allocation problem through a distributed version of the Hungarian Method. Additionally, in [25], an evolutionary computational decentralized allocation algorithm was presented for solving the MRTA problem.

2.2.2 Homogeneous and Heterogeneous Systems

All members of homogenous MRSs share the same physical characteristics and functionality. A heterogeneous fleet can consist of different types of robots (mobile robots, robotic arms, and others), robot manufacturers, and several actuators (systems that open/close doors, elevator managers). Homogeneous groups are frequently used in swarm robotics, a sub-field of MRS where the collective behavior of the robots results from local interactions among the robots and the environment in which they operate. [26]. Moreover, heterogeneous groups benefit from cooperative object transport scenarios, which often require complex and diverse behavioral competencies.

2.3 Fleet Management Systems

A Fleet Management System (FMS) applied to automated robotics is a tool capable of managing a fleet of robots and optimizing their performance. In most cases, fleet managers must be able to optimize the cost of execution of a task, assign the right task to the correct robot, create new tasks, monitor the robot fleet, and detect and resolve conflicts. Many companies develop fleet management solutions. All of the fleet managers mentioned in this section are capable of managing a heterogeneous fleet of AMRs.

2.3.1 Meili Robots - Meili FMS

Meili FMS [1] is a universal solution that handles all types and brands of robots and forklifts in one platform. It can be fully integrated with third-party systems such as Warehouse Management Systems and Enterprise Resource Planning (ERP). In addition to the general features of a fleet management system mentioned above, Meili FMS has features to authorize and control user access, data analysis to get an overview of the fleet's productivity, and map edition and enrichment. Figure 2.1 shows a complete diagram of Meili FMS features.

The Meili Fleet Management System manages all types of mobile robots based on ROS or ROS2 technology. Meili FMS integrates three communication protocols [27] to exchange data and messages between computer systems: i) **HTTP** via RESTful API for simple and flexible user access with easy integration, and Webhooks, which allow user-defined HTTP callbacks that are triggered by actions in the system. ii) **WebSockets** to enable bi-directional client-server communication, and iii) **MQTT** for wireless message delivery and data transmission of data from remote sources.

2.3.2 InOrbit

InOrbit [28] is a secure, scalable, cloud-based RobOps (Robot Operations) and analytics platform that enables robotics companies and their customers to develop, deploy and operate intelligent robots globally. The platform can easily monitor, control and optimize a robot connected to the cloud from anywhere. InOrbit can control any autonomous mobile robot; ROS and ROS2-based

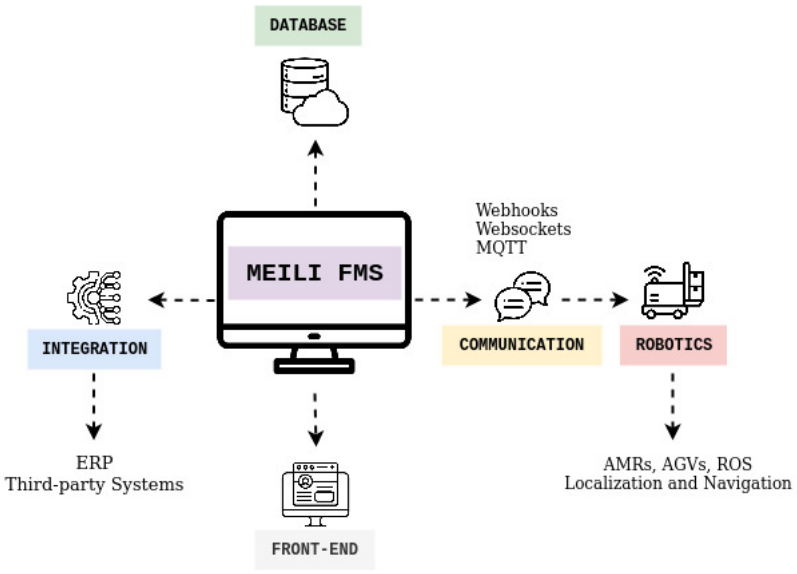


Figure 2.1: Diagram of Meili FMS features. Adapted from [1]

robots are easy to configure, but it is possible to activate the platform on many non-standard platforms.

The robots and the cloud communicate via WiFi and Cellular Data. Also, the MQTT protocol is used to deal with connection unreliability and custom protocols to ensure robust connectivity. InOrbit can send commands to the robots in two ways: i) executing scripts that customers provide on the robot or ii) sending a message via a ROS topic; all these commands are transmitted via MQTT [29].

2.4 Interoperability

According to a research report published by Spherical Insights & Consulting [30], the Autonomous Mobile Robot Market Size was valued at USD 3.07 billion in 2022, and the Worldwide Autonomous Mobile Robot Market Size is expected to reach USD 14.02 billion by 2032, with an annual growth rate of 16.4%. This growth can be attributed to the increasing use of AMRs in the automation process of the manufacturing industry, warehouses, logistics centers, healthcare, and other applications.

With the demand for robots at a higher level than ever before, several types of businesses are adopting a variety of mobile robots in their facilities. In most cases, these robots are supplied by different manufacturers, are highly specialized in their functions, and come with different operating systems or are controlled by a specific fleet management system provided by the vendor. In addition, robots are becoming more autonomous, allowing them to operate without a human operator, highlighting the importance of communication between robots to ensure efficiency and safety.

These multi-vendor, multi-robot systems have one problem in common - a need for interoperability. Interoperability is the ability of a computer system or software to exchange information with other computer-based systems. Outside of the robotic scope, interoperability allows several different systems to work together, aiding the work of information technology. Google and Apple have addressed this subject during the COVID-19 pandemic to develop an API that enables interoperability between Android and iOS [31] devices using applications from public health authorities to assist in enabling contact tracing.

In robotics, increased interoperability between different robotic systems would benefit large manufacturing companies and small industrial facilities. Robotic system designers, manufacturers, and vendors also benefit from interoperability because modular, adaptable robotic systems in the marketplace can increase interest in adopting automation and robotics in the industry.

2.5 Interoperability Efforts

The problem of lack of interoperability can be solved with a universal solution that addresses all the different types and brands of robots with a common interface for control and management. Over the past few years, efforts have been made to create an interoperability standard to satisfy the need for better communication, implementation, and lower integration costs for robots from different vendors. The following subsections describe some of the interoperability initiatives in detail.

2.5.1 VDA 5050 Standard

A universal solution can solve the problem of lack of interoperability. This solution must handle all types and brands of robots with a standard interface for control and management. Over the past few years, efforts have been made to create an interoperability standard to satisfy the need for better communication, implementation, and lower integration costs for robots from different vendors. The following subsections describe some of the interoperability initiatives in detail.

VDA 5050 Standard [5] started with the cooperation between the Verband der Automobilindustrie e.V. (German abbreviation VDA) and Verband Deutscher Maschinen-und Anlagenbau e.V. (German abbreviation VDMA) to define a standardized interface that simplifies the connection of a new robot to existing master control, allowing the exchange of order and status information between them, and enabling operation with multiples AGVs from different manufactures.

Since its release, the standard has been used in many projects to support interoperability between AGVs from different manufacturers. One example is implementing a line-free assembly demonstrator [32], where assembly resources and stations are distributed freely on the production floor and connected by AGVs. For product transport, a message with information about the station and product is sent to the fleet management, which plans an optimal route and then sends the route to the AGV. The communication between the fleet manager and the robots is done via VDA 5050 because the demonstrator uses three different robots from different manufacturers.

In [33], a distributed control testbed system for supply chain and logistics has been designed based on the Resmanio platform. This platform is VDA 5050 compliant and MQTT ready, and it offers improvements over other platforms, such as the ability to integrate any AMR/AGV vehicles that are VDA 5050 compliant and provides a VDA 5050 adapter for legacy AGVs [34].

InOrbit, OTTO Motors, and Ekumen collaborated to create an open-source connector to increase interoperability within the robotics ecosystem. By allowing the design of adapters that perform a wide range of tasks utilizing the VDA 5050 Standard, this connector enables VDA 5050 compliance in ROS2 robots [35]. The VDA 5050 Connector for ROS2 can be considered a bridge between a VDA 5050 Master Control and the ROS2 robot API.

The VDA connector consists of 3 ROS nodes, one of which translates VDA messages into ROS2 messages and vice versa; another one, the Controller node, handles the robot execution as specified in the standard; it validates, executes, or cancels orders and assembles feedback information; the other one is responsible for establishing a link between the controller and the Robot API, it is the one that sends a navigation goal request, executes an action or retrieves information about the robot.

NVIDIA's interoperability initiative, Isaac Mission Dispatch [36], is a cloud service that allows fleet management software to send missions to multiple robots and monitor the robot and mission status. Integrated with the fleet management system, Mission Dispatch communicates with the robots using the VDA 5050 standard and via MQTT.

A collaboration between OTTO Motors, InOrbit, and NVIDIA made it possible to perform tests to verify the functionality of VDA 5050 in the AMRs manufactured by OTTO [37]. The tests also included NVIDIA's Mission Dispatch software. In the simulation, the software worked smoothly with OTTO Motors' VDA 5050 connector. The tests demonstrated that seamless communication and integration between AMRs and third-party controllers can be achieved through VDA 5050 compatibility.

2.5.2 Open Robotics Middleware Framework (OpenRMF)

OpenRMF [2] is a collection of reusable, scalable open-source libraries and tools building on top of Robot Operating System (ROS) 2 that enables the interoperability of heterogeneous fleets of any robotic systems.

A design goal of OpenRMF is to allow multiple unrelated fleet managers to cooperate without changing how those fleet managers are implemented.

The framework specifies RMF, an umbrella term encompassing open specifications and tools, to facilitate the integration and interoperability of robotic systems. The RMF Core consists of the following:

- RMF Traffic: scheduling and traffic management systems;
- RMF Task: task allocation;
- RMF Battery: provides APIs for modeling a robot's battery life;

- RMF Utilities: low-level C++ programming utilities used across all OpenRMF packages.

There are various ways for systems and users to interact with OpenRMF. Usually, the communications between the user interface and the RFM core are made through Websockets API. A Fleet Adapter connects the RMF core and the fleet managers with any user-preferred API. The provided fleet manager demos explained in their documentation use REST and WebSocket APIs to interface between the adapter and the simulation robots. Figures 2.2 and 2.3 summarize the RMF structure explained previously.

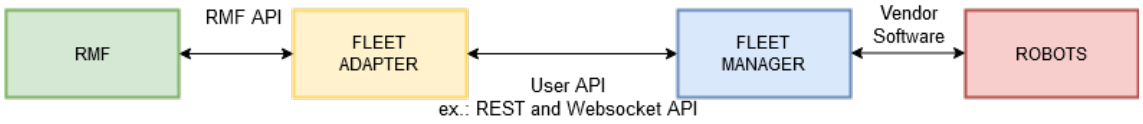


Figure 2.2: The diagram illustrating how OpenRMF communicates with the robots. Adapted from [2].

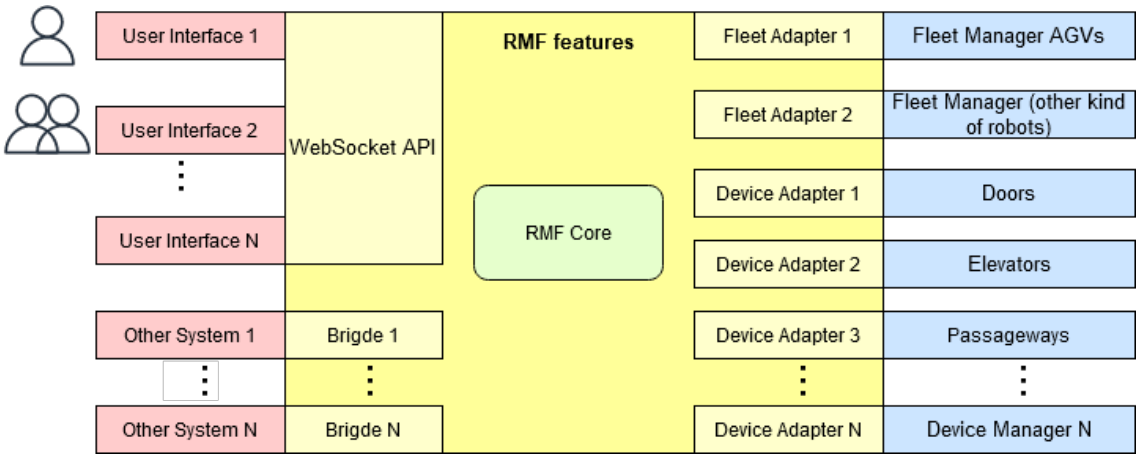


Figure 2.3: Overview of the OpenRMF architecture. Adapted from [2]

2.5.3 MassRobotics Interoperability Standard

The first version of the Interoperability Standard [3] was published in 2021 by MassRobotics, an independent nonprofit robotics innovation center. The developed standard is open-sourced and highly lightweight in terms of both implementation and computation times. The MassRobotics standard allows all connected agents to share their operational status, location, speed, capabilities, and availability to enable organizations to deploy AMRs from multiple vendors and have them coexist effectively.

The interoperability standard uses IETF RFC 6455 WebSocket Protocol to transport messages. The protocol supports constant communication allowing for real-time telemetry transmission and other data updated as the AMRs operate. Messaging will be done in JSON structure. The diagram in Figure 2.4 illustrates the message transport and where the MassRobotics standard is applied.

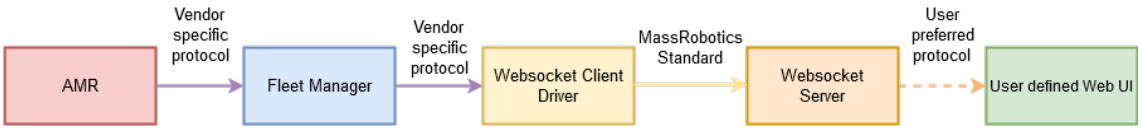


Figure 2.4: Information flow in the MassRobotics Interoperability Standard. Adapted from [3].

The first public demonstration of the MassRobotics Interoperability Standard took place during the 2021 A3 Automate AMR and Logistics Conference [38]. The demonstration involved three companies, Vecna Robotics, Waypoint Robotics, and WiBotic Autonomous Charging. A Vecna mobile robot created a map of the facility and shared it via the MassRobotic standard with the visualization application created by Waypoint. The map displayed all the robots moving through the facility and other companies' systems.

2.5.4 Open Platform Communications Unified Architecture (OPC UA)

The Open Platform Communications (OPC) interoperability standard is a machine-to-machine communication protocol used in industrial automation and other sectors for safe and dependable data transmission [39]. By creating abstract services, the standard offers a Service-Oriented Architecture (SOA) for industrial applications, ranging from factory floor devices to corporate applications.

OPC UA is an evolution of the original OPC standard, which only worked on Microsoft Windows platform devices. Following vendors' demand, a platform-independent specification, the OPC UA application now runs on non-Microsoft systems by defining an abstract set of services and mapping them to different technologies, like Web Services.

Figure 2.5 shows that the OPC architecture consists of three main elements: field devices (instruments, Modbus, CANBUS, PLC), OPC Server, and OPC Client. The OPC Server is a software that implements the OPC standard and provides the OPC interface to all client applications. It uses a proprietary communication protocol and can be provided by different vendors. Hardware vendors can provide an OPC server for their system, allowing standardized access, and there can be independent OPC Servers for the systems where the manufacturers do not offer an OPC server of their own.

The OPC client software is any program that needs to connect to the hardware to extract data. The server makes the data available to any connected client. OPC Clients can be SCADA systems, MES systems, Raspberry PI, or any other system dependent on data exchange with industrial systems.

With its ability to connect services to multiple technologies, OPC UA addresses the need to manage device access and provide pure Web services to MES and ERP applications. An ERP requires real-time Industrial Internet of Things data and analytics for an intelligent factory. Due to the need to speed up processes, reduce errors and manual labor, and collect real-time data as close to the source as possible, integrating machine data via OPC UA to the ERP system is essential [40].

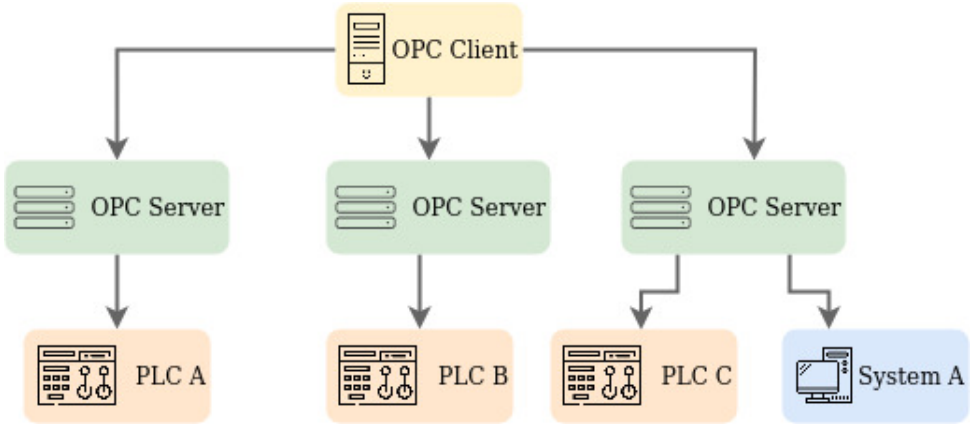


Figure 2.5: Overview of the OPC UA Architecture. Adapted from [4]

2.6 Integration of Information Systems with Fleet Management Systems

Industries from many different sectors share the goal of maximizing workflow efficiency on factory floors and in distribution centers. Moreover, multi-robot systems are one of the most versatile solutions to achieve this goal. Although, in some cases, a robot fleet management system can act as a complete application, other software systems such as ERP, Manufacturing Execution Systems (MES), Warehouse Management, Execution, or Control Systems (WMS, WES, WCS) in the operational environment frequently require integration.

In order to provide a standard-based model to support the integration of AMR fleets and these information systems, the ISA-95 enterprise model is adopted in [41]. The ISA-95 [42] standard was developed to provide common models and terminologies for exchanging information between enterprise and control functions. ISA-95 defines different functional levels, with emphasis on Levels 3 and 4. Level 4 – Business planning and logistics establish the primary production, delivery, shipping schedules, and inventory levels. Furthermore, Level 3 – Manufacturing Operations Management and Control implements functions to perform detailed scheduling of operations and functions to control and optimize the production process. These levels and levels 0 to 2 compose the generic integration model.

ISA-95 (Part 3) [43] defines activity models of manufacturing operations management that enable enterprise systems to control system integration. Production, maintenance, and inventory are examples of the generic activity model. Fleet management system (FMS) activities can also be an instantiation of the generic model that handles AMR operations. Below is an outline of some Level 3 activities and how they are instantiated in AMR fleet management.

- Resource management: information management describing resources (machines, tools, labor, skills, materials, energy) required in operations. The resources in FMS are mainly AMRs, which include a description of their skills concerning what tasks they can do and the location and navigation properties of work areas.

- Detailed scheduling: determine the optimal use of resources to meet the production schedule requirements. In a fleet management system, this activity represents the optimal assignment of tasks to vehicles and the routing of the fleet in order to guarantee the production plan.
- Execution management and control: coordination of the processes to ensure the proper performance of the work plan. For AMRs, this means the detection, prevention, and correction of deadlock and low-battery situations that may appear in the execution of the work.

Even though these standards and reference models are essential, it should be noted that the integration is much more difficult in real life due to the high heterogeneity of interfaces. In these cases, the integration is frequently done by establishing a middleware system or a broker between the FMS and external applications.

2.7 Conclusion

This chapter presents the current state of the art in mobile robots, communication protocols, fleet management systems, and interoperability standards. The chapter first introduces the concept of Industry 4.0 and the importance of IoT communication protocols. It also presents the theoretical background of multi-robot systems, how they are categorized, different control architectures, and the associated problems. This chapter presents different approaches to address the interoperability issues in detail. Finally, it discusses how fleet management systems can be integrated with information systems.

Chapter 3

VDA 5050 Standard

As explained briefly in Chapter 2, VDA 5050 is a standardized interface for AGV communication. The standard primarily defines the exchange of information between the master control (a fleet management software program) and the AGVs/AMRs. As a result, single fleet management can supervise the vehicles of many providers. [44].

The interface's goal is to make it simpler to connect new robots to existing master control and to communicate in a uniform language. The resulting simplicity enables the simultaneous operation of AGV from various manufacturers, regardless of its characteristics, and conventional systems (inventory systems) in the same working environment.

According to VDA 5050, the following requirements must be met for a uniform interface between the master control and the AGV to be established.

- The description of a communication standard between AGVs and master control lays the foundation for integrating robotic transportation systems that aim for automation when employing cooperative transport vehicles.
- A shorter deployment period since central services give the necessary data and are often accurate. Vehicles should be able to be operated with the same implementation effort, regardless of the manufacturer.
- Consistent, comprehensive coordination with the appropriate logic for all transport vehicles, vehicle types and manufacturers reduces complexity and increases the "Plug & Play" capability of the systems.

3.1 Communication Procedure

In order to carry out the goals mentioned above, the Standard presents a communication interface for exchanging order and status data. VDA 5050 establishes that the communication between the master control and the vehicles is done over wireless networks using the MQTT protocol, and the messages are packed in a JSON format. The process of communication carried out by VDA 5050 can be summarized in three steps:

1. The operator provides necessary information;
2. The master control receives and manages the operations;
3. The AGV executes the orders.

At the beginning of the process, the operator defines the requirements. The information is either entered manually or imported from other systems. The operator supplies the definition of routes and the route network configuration (localization of battery charging stations, gates, elevators, and stations for loading/unloading) and information about the physical properties of the vehicle.

The master control is responsible for assigning the resulting orders to the AGV. The orders are transferred to the robot via an MQTT message broker, and using the same via, the AGV continuously reports its status to the master control. Besides assigning orders, the master control has the functions of route calculation and guidance of the robot, traffic control, detection and resolution of blockages, and communication failures. Along with continuously transmitting its status, the AGV has the functions of localization and navigation. Figure 3.1 outlines a summary of the communication content in the operational phase.

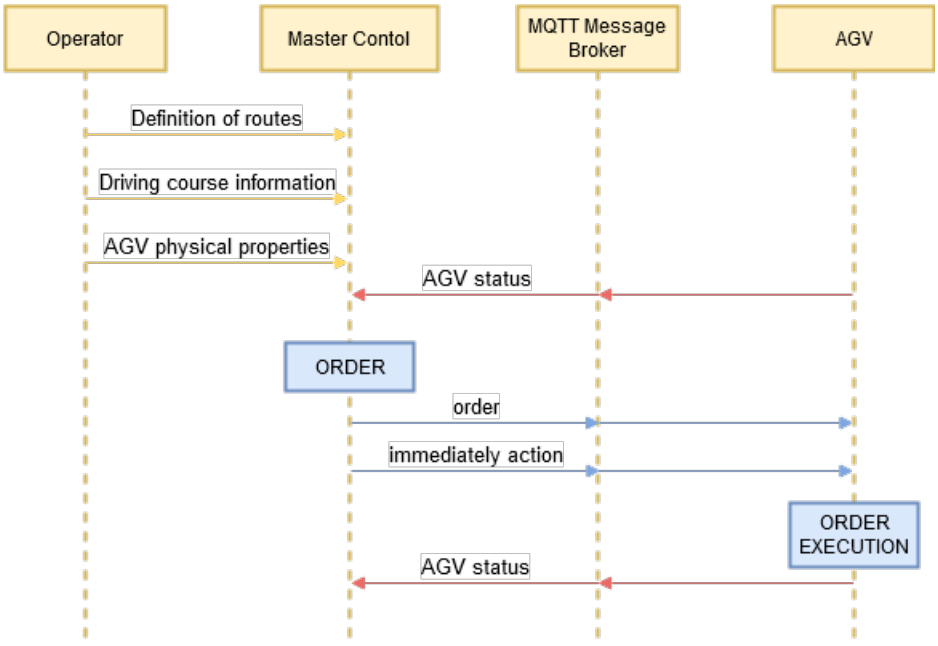


Figure 3.1: Overview of the information flow during the process. Adapted from [5].

3.2 VDA 5050 Subtopics for Communication

For sharing information between master control and AGV, VDA 5050 defines the type of messages to be exchanged. These messages are JSON files that depending on their content, specify a different kind of message, called subtopics. The standard provides in their GitHub JSON Schemas

for validation of the created JSON. These subtopics are presented in Table 3.1 along with their respective publisher and subscriber and the need for implementation. The subsections below provide precise details for each subtopic [45].

Table 3.1: VDA 5050 Subtopics for Communication

Subtopic name	Published by	Subscribed by	Implementation
order	master control	AGV	mandatory
instantActions	master control	AGV	mandatory
state	AGV	master control	mandatory
connection	Broker/AGV	master control	mandatory
factsheet	AGV	master control	mandatory
visualization	AGV	visualization systems	optional

3.2.1 Order

A graph characterizes it as its fundamental structure. The Order subtopic includes a path from node X to node Y and actions that need to be executed, as seen in Figure 3.2. The order message consists of two lists of nodes and edges, along with actions chosen from a predefined set (start/stop charging, pick, drop, cancel an order). An example of an order message in JSON format is in Figure 3.3. The AGV must navigate through all the nodes and edges to complete the order. For a valid message, at least one node must be present, and the number of edges must be the number of nodes minus one.

The boolean attribute "released" is one of the attributes in the order message. The vehicle is expected to travel through any released nodes or edges and avoid crossing the ones that have not been released. The set of released nodes and edges is called "base," and the set of unreleased nodes and edges is called "horizon."

Since blocking the entire route that other vehicles may use to traverse would be very inefficient in practice, VDA 5050 defines that an order is divided into two parts:

1. Driving to the decision point "base": part of the order that the AGV is expected to execute immediately and is not assumed to be canceled.

2. Estimated journey from the decision point "horizon": defined by the route the AGV is likely to drive. The route has yet to be confirmed and, therefore, is only informative and subject to change.

3.2.2 Instant Actions

The standard defines Instant Actions as actions that need to be performed by the robots immediately. Instant actions may be essential to change tasks quickly, prevent crashes or "deadlocks" at

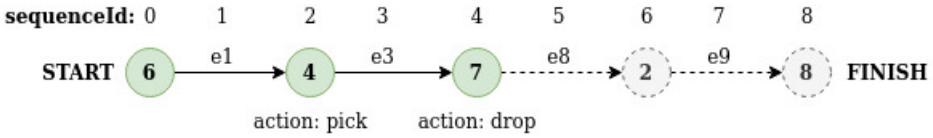


Figure 3.2: Graph transmitted to the robot in an Order


```
{
  "headerId": 1,
  "timestamp": "2023-03-01T11:00:00Z",
  "version": "1.1.1",
  "manufacturer": "m_01",
  "serialNumber": "1234",

  "orderId": "o-001",
  "orderUpdateId": 0,
  "nodes": [
    {
      "nodeId": "n6",
      "sequenceId": 0,
      "released": "true",
      "actions": [ ]
    },
    {
      "nodeId": "n4",
      "sequenceId": 2,
      "released": "true",
      "actions": [
        {
          "actionId": "a-n4-001",
          "actionName": "pick",
          "blockingType": "HARD",
          "actionParameters": [
            {
              "key": "stationType",
              "value": "floor"
            },
            {
              "key": "loadType",
              "value":"EPAL"
            }
          ]
        }
      ]
    },
    {
      "nodeId": "n7",
      "sequenceId": 4,
      "released": "true",
      "actions": [
        {
          "actionId": "a-n7-001",
          "actionName": "drop",
          "blockingType": "HARD",
          "actionParameters": [
            {
              "key": "stationType",
              "value": "floor"
            },
            {
              "key": "loadType",
              "value": "EPAL"
            }
          ]
        }
      ]
    },
    {
      "nodeId": "n2",
      "sequenceId": 6,
      "released": "false",
      "actions": [ ]
    },
    {
      "nodeId": "n8",
      "sequenceId": 8,
      "released": "false",
      "actions": [ ]
    }
  ],
  "edges": [
    {
      "edgeId": "e1",
      "sequenceId": 1,
      "released": "true",
      "startNodeId": "n6",
      "endNodeId": "n4",
      "action": [ ]
    },
    {
      "edgeId": "e3",
      "sequenceId": 3,
      "released": "true",
      "startNodeId": "n4",
      "endNodeId": "n7",
      "action": [ ]
    },
    {
      "edgeId": "e8",
      "sequenceId": 5,
      "released": "false",
      "startNodeId": "n7",
      "endNodeId": "n2",
      "action": [ ]
    },
    {
      "edgeId": "e9",
      "sequenceId": 7,
      "released": "false",
      "startNodeId": "n2",
      "endNodeId": "n8",
      "action": [ ]
    }
  ]
}
```


Figure 3.3: Example of a VDA 5050 Order message in JSON format

junctions, or react to errors of other vehicles. It is important to emphasize that when publishing an *instantAction* message to the subtopic *instantActions*, its content must not contrast with the content of the AGV's current order. An example of the Instant Action message and its contents can be seen in Figure 3.4.

3.2.3 State

The state message is sent to the master control by the robot. It includes lists of nodes and edges that the AGV must still traverse (*nodeStates* and *edgeStates*), a list of actions that the vehicle will handle in the future (*actionStates*), and information about safety, errors, battery state and comprehensive data about the nodes and edges (*nodeState* and *edgeState*). These messages are sent at regular intervals of 30 seconds or are triggered on unique occasions (receiving an order, errors, the arrival at a node).

Two attributes monitor the order's progress inside the state message: *nodeStates* and *edgeStates*. When traversing a node, the vehicle reports the traversal by deleting its *nodeState* from the

```
{
  "headerId": 1,
  "timestamp": "2023-03-01T11:00:00Z",
  "version": "1.1.1",
  "manufacturer": "m_01",
  "serialNumber": "1234",

  "actions": [
    {
      "actionId": "ia-001",
      "actionName": "initPosition",
      "blockingType": "HARD",
      "actionParameters": [
        {
          "key": "x",
          "value": 0
        },
        {
          "key": "y",
          "value": 0
        },
        {
          "key": "theta",
          "value": 0
        },
        {
          "key": "mapId",
          "value": "map-001"
        },
        {
          "key": "lastNodeId",
          "value": "n8"
        }
      ]
    }
]
```


Figure 3.4: Example of a VDA 5050 Instant Action message in JSON format

nodeStates array. The traversal of a node also signifies the end of the edge that leads up to the node. The edge must then be removed from the *edgeStates*. An illustration of this procedure can be seen in Figure 3.5. When the vehicle receives an action on the topic, connected to a node or an edge, the *actionStates* array stores the message. The *status* field can track the stage of the action and can have the values: waiting, initializing, running, paused, finished, or failed.

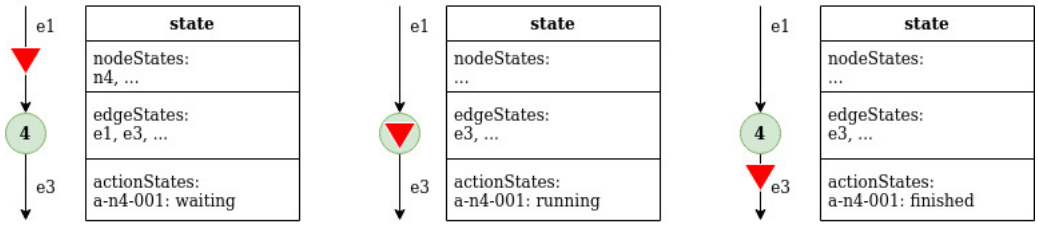


Figure 3.5: State topic fields during order handling

3.2.4 Connection, Fact sheet, and Visualization

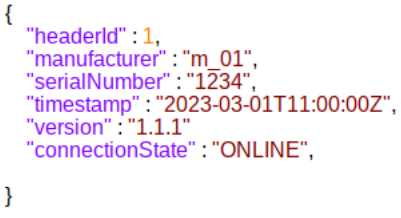
The connection message is an MQTT protocol-level check of connection. During the connection of an AGV client to the MQTT Broker, a last-will topic and message can be set. The client publishes this last-will message upon the abrupt disconnection from the broker. This method allows the master control to detect disconnection by subscribing to the connection topic. An example of the VDA connection message can be seen in Figure 3.7.

```
{
  "headerId": 1,
  "timestamp": "2023-03-01T11:00:00Z",
  "version": "1.1.1",
  "manufacturer": "m_01",
  "serialNumber": "1234",

  "orderId": "o-001",
  "orderUpdateId": 0,
  "lastNodeId": "n6",
  "lastNodeSequenceId": 0,
  "driving": "true",
  "operatingMode": "AUTOMATIC",

  "nodeStates": [
    {
      "nodeId": "n4",
      "released": "true",
      "sequenceId": 2
    },
    {
      "nodeId": "n7",
      "released": "true",
      "sequenceId": 4
    },
    {
      "nodeId": "n2",
      "released": "false",
      "sequenceId": 6
    },
    {
      "nodeId": "n8",
      "released": "false",
      "sequenceId": 8
    }
  ],
  "edgeStates": [
    {
      "edgeId": "e1",
      "sequenceId": 1,
      "released": "true"
    },
    {
      "edgeId": "e3",
      "sequenceId": 3,
      "released": "true"
    },
    {
      "edgeId": "e8",
      "sequenceId": 5,
      "released": "false"
    },
    {
      "edgeId": "e9",
      "sequenceId": 7,
      "released": "false"
    }
  ],
  "actionStatus": [
    {
      "actionId": "a-n4-001",
      "actionStatus": "WAITING"
    },
    {
      "actionId": "a-n7-001",
      "actionStatus": "WAITING"
    }
  ],
  "batteryState": [
    {
      "batteryCharge": 100,
      "charging": "false"
    }
  ],
  "safetyState": [
    {
      "eStop": "AUTOACK",
      "fieldViolation": "false"
    }
  ],
  "errors": []
}
```


Figure 3.6: Example of a VDA 5050 State message in JSON format

Primary to the operational phase, the operator stores the vehicle configuration sent from the robot via the subtopic *factsheet*. The fact sheet describes a specific AGV type series' fundamental characteristics. It also includes information about communication interfaces required to integrate an AGV into a VDA5050-compliant master control. When designing, dimensioning, and simulating an AGV system, this information can be used to compare various AGV models. The fact sheet message includes fields that specify the fundamental physical properties of the robot, the definition of the AGV geometry, and the specification of load capabilities and localization. Figure 3.8 shows an example of a Factsheet message.

VDA 5050 defines the subtopic *visualization*. It allows robots to report their current position in a near real-time frequency. The subtopic is marked as optional implementation since, for a transport guidance system, these position updates are of no relevance [45]. In contrast, when working with a 2D map, this type of information is critical.



```
{
  "headerId" : 1,
  "manufacturer" : "m_01",
  "serialNumber" : "1234",
  "timestamp" : "2023-03-01T11:00:00Z",
  "version" : "1.1.1"
  "connectionState" : "ONLINE",

}
```


Figure 3.7: Example of a VDA Connection message in JSON format

```
{
  "headerId": 1,
  "timestamp": "2023-03-01T11:00:00Z",
  "version": "1.1.1",
  "manufacturer": "m_01",
  "serialNumber": "1234",

  "typeSpecification": [
    {
      "seriesName": "r-series",
      "agvKinematic": "THREEWHEEL",
      "agvClass": "FORKLIFT",
      "maxLoadMass": 1000,
      "localizationTypes": "DMC",
      "navigationTypes": "PHYSICAL_LINDE_GUIDED"
    }
  ],
  "physicalParameters": [
    {
      "speedMin": 0.3,
      "speedMax": 1.2,
      "accelerationMax": 0.3,
      "decelerationMax": 0.1,
      "heightMax": 1.9,
      "width": 0.99,
      "length": 1.64
    }
  ],
  "protocolLimits": [
    {
      "maxStringLens": [] ,
      "maxArrayLens": [] ,
      "timing": [
        {
          "minOrderInterval": 30,
          "minStateInterval": 30
        }
      ]
    }
  ],
  "protocolFeatures": [
    {
      "optionalParameters": [],
      "agvActions": []
    }
  ],
  "agvGeometry": [],
  "loadSpecification": []
}
```


Figure 3.8: Example of a VDA Factsheet message in JSON format

3.3 Field Testing

The first live test of the VDA 5050 standard took place at the AGV Mesh-Up on March 2021. During the event, a total of six vehicles from Arculus, DS AUTOMOTION, SAFELOG, Siemens AG, SSI SCHÄFER, and STILL, see Figure 3.9 operated under one master control and drove together as a unit. The name AGV Mesh-Up is based on a mesh WLAN in which different components are linked together and viewed as a single WLAN; for the robots, this means that they drive with different types of navigation (e.g., line-guided or contour-based) but communicate with the higher-level control system in a common data language [6].

Figure 3.9: Mobile robots that participated in the test performed at the AGV Mesh-Up. Adapted from [6].

Following the success of the 2021 event, the second and third edition of AGV Mesh-Up was held in 2022 and 2023. In 2022, the latest version was tested, and more robots participated. Moreover, in 2023, the previous test scenarios culminated, and eleven companies tested the VDA 5050 communication interface live. The variety of vehicles increased the difficulty of the test. In addition to 5 underride vehicles and a mobile robot, three forklifts in the common driving area demonstrated storage and retrieval processes on two racks installed for this purpose [46].

3.4 Conclusion

In summary, this chapter focuses solely on the VDA 5050 standard, presenting its definition, intent, and functionality. The first topic covered in the chapter is what elements are involved in the communication process and their functions. Later, this chapter presents the VDA 5050 sub-topics for communication and explains each message that can be exchanged in detail with examples. Finally, mentions of a live test of the VDA 5050 standard and the conditions for the test are contextualized.

Chapter 4

Implementation

4.1 Introduction

In a practical context, when starting a new project or migrating to standardized VDA 5050 communication, implementing the standard does not just depend on a superficial knowledge of its objectives and scope, and it is not as simple as telling AGV system manufacturers or fleet management systems that the standard must be supported.

A successful implementation requires a detailed study. An analysis of the standard and the system in which it is to be implemented is essential. The study aims to identify a point of interaction between the system and VDA 5050. To this end, four questions can be asked as an initial guide to the implementation. The question listed below will also be used as a guide for the first sub-section.

1. Which VDA sub-topics are relevant for this project?
2. Which information do the fleet manager and AGV send?
3. Which information do the fleet manager and AGV receive?
4. Which system must react to which information?

4.2 Tools and Technology Used

The software has been developed entirely in the C++ programming language and on top of the ROS framework. In order to control a robot fully developed by INESC TEC's Centre for Robotics in Industry and Intelligent Systems (CRIIS), it is necessary to use the INESC TEC Navigation Stack (INS). The Stack is a complex system developed on top of a ROS framework. It includes localization algorithms, drivers to interface with the robot hardware, a path planner, a parametric trajectory controller, and a map server. As a ROS-based software, its structure consists of nodes, topics for communication, and services.

The INS also has an interface that simulates the environment in which the robot is located. This interface runs in Rviz and loads a map created by the robot, and then it is possible to define

nodes and edges that represent the possible paths the robot can take. The nodes and edges form a graph, each element with a unique identifier used to send commands to the robot. In figure 4.1, it is possible to observe in the map the nodes, blue circles, edges, red or orange lines, and the two robots represented by the yellow and red arrows.

Figure 4.1: Navigation Stack Simulation Environment in Rviz

As defined in the VDA 5050 standard, information is exchanged between the master control and the robot using the MQTT protocol, and the messages are packaged in JSON format. The MQTT protocol works as follows: first, the MQTT client connects to the MQTT broker; once connected, the client can publish messages, subscribe to messages, or both. When the MQTT broker receives a message, it sends it to the interested subscribers. The MQTT broker filters the messages using keywords called MQTT topics. Topics are strings that enable communication between clients.

The Eclipse Paho library was chosen for the implementation of the MQTT client. This library is one of the most popular clients available. It is easy to use and can fully implement MQTT subscriptions and messaging with a small amount of code. Using the MQTT X tool, it was possible to visualize the messages received via MQTT. The MQTT X interface displays the published/received messages in the form of a chat, making it easier to operate the page.

Since C++ has no built-in support for working with JSON data, several libraries have been created for this purpose. One of them is the nlohmann JSON library used in this project. The nlohmann JSON has a trivial integration, consisting of a single header file, is complete with extensive documentation, and facilitates the process of serialization and deserialization/parsing.

4.3 Software Architecture

To better understand the software's overall structure and organization, a case study has been developed to investigate how the VDA 5050 establishes standardized communication between a robot fleet manager and an AMR. This study consists of two parts, one more general, focusing on the exchange of messages with the master control and their content, and the other more specific, breaking down the topics and nodes of the Navigation Stack to find where the standard can interact with the system.

In Chapter 3, when discussing the VDA 5050 subtopics for communication, it was mentioned that the standard uses JSON formatted files to exchange messages over wireless networks using the MQTT protocol. The Order, State, Connection, InstantAction, and Factsheet messages follow a structure described in the JSON schemas available on the VDA 5050 Github. An example of all these messages was created to illustrate the content of the information exchanged with the master control. These examples were presented in Chapter 3.

After analyzing the structure of the Navigation Stack and considering that the VDA subtopics Order and State are used to exchange messages about the robot's assigned path and its state of completion, three INS nodes and two topics were considered the most important. These nodes and topics are described below and illustrated in Figures 4.2 and 4.3.

- **Navigation Handler Node:** responsible for receiving and executing an order. When a new task arrives, consisting of the robot type and its target edge, this node sends the information to the Supervisor node.

- **Supervisor Node:** responsible for monitoring the path and assigning tasks. After receiving and analyzing the new order, this node assigns the best robot for the task and sends the destination edge to the Path Planner node.

- **Path Planner Node:** responsible for managing the path through TEA*. With the destination edge, it assigns a complete path of edges to the task, allowing the robot to reach its destination. This information makes the inverse path until it arrives at the Navigation Handler Node.

- **Target Route Topic:** exchange information about the best path for the robot to follow, which the path planner node decides. The Supervisor node sends this information to the Navigation Handler node.

- **Execution State Topic:** continuously sends information about the state of the trajectory. The Navigation Handler node sends the state data to the Supervisor node.

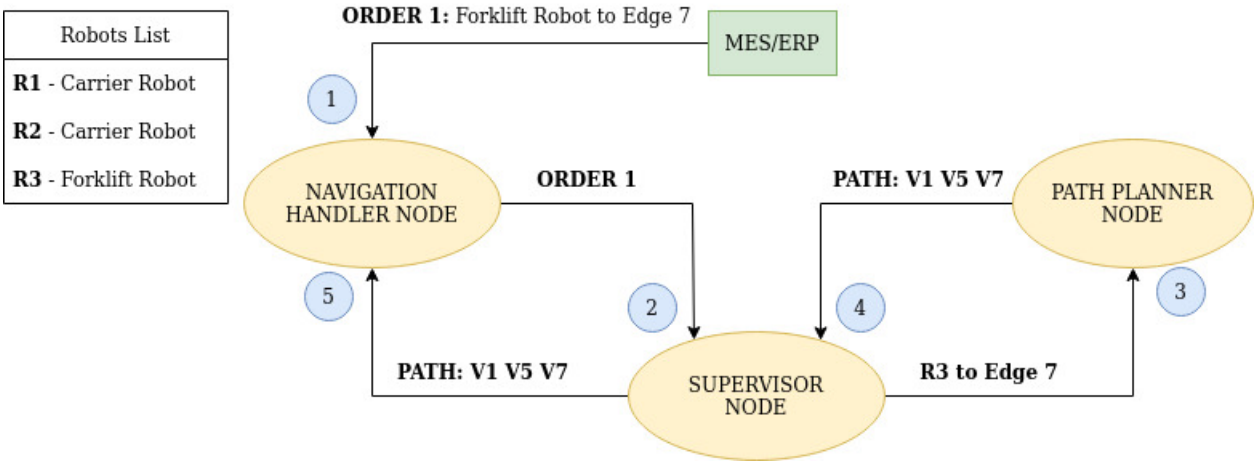


Figure 4.2: Illustration about the process after receiving a new order

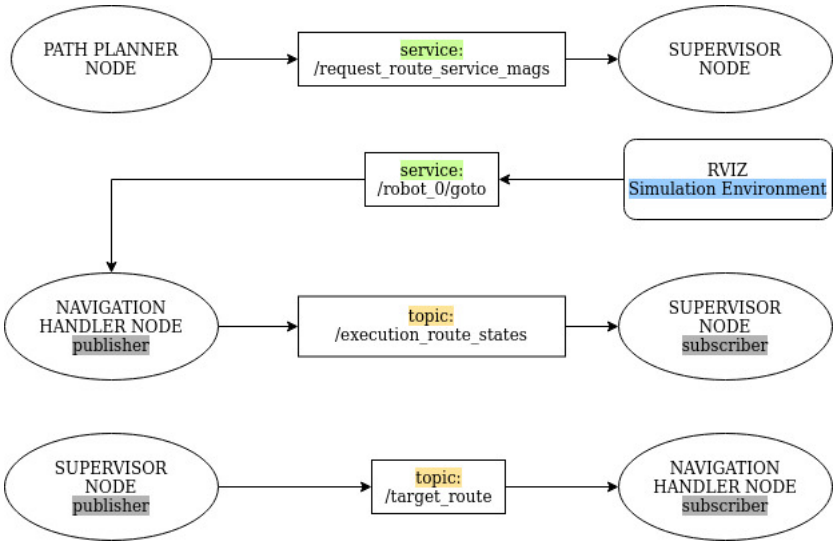


Figure 4.3: Depiction of the exchange of information among INS Nodes and Topics

After considering that the VDA 5050 standard concerns the communication between AGVs and a master control using the messages mentioned above, a parallel can be made with the existing message exchange in the Navigation Stack. The VDA 5050 master control is equivalent to the INS Supervisor and Path Planner nodes (fleet manager), and since the master control is also responsible for sending the order to the AGV, the Navigation Handler node can also be considered part of the master control. This parallel can be seen in Figure 4.4.

The VDA 5050 master control sends and receives messages via the MQTT broker. Figure 4.5 shows a complete diagram of the exchange of the VDA messages. When implementing the MQTT protocol, the content of the Target Route and Execution State topics are used to create the VDA 5050 messages that will be sent through the MQTT broker.

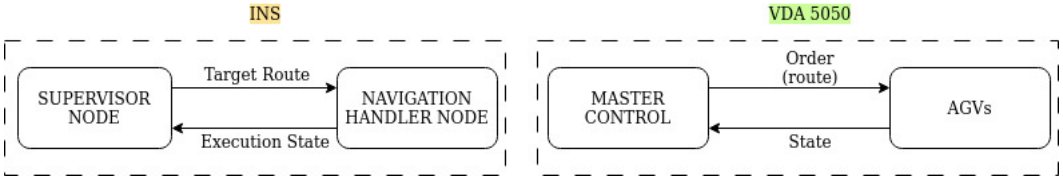


Figure 4.4: Parallel between the Navigation Stack and the VDA5050 Standard

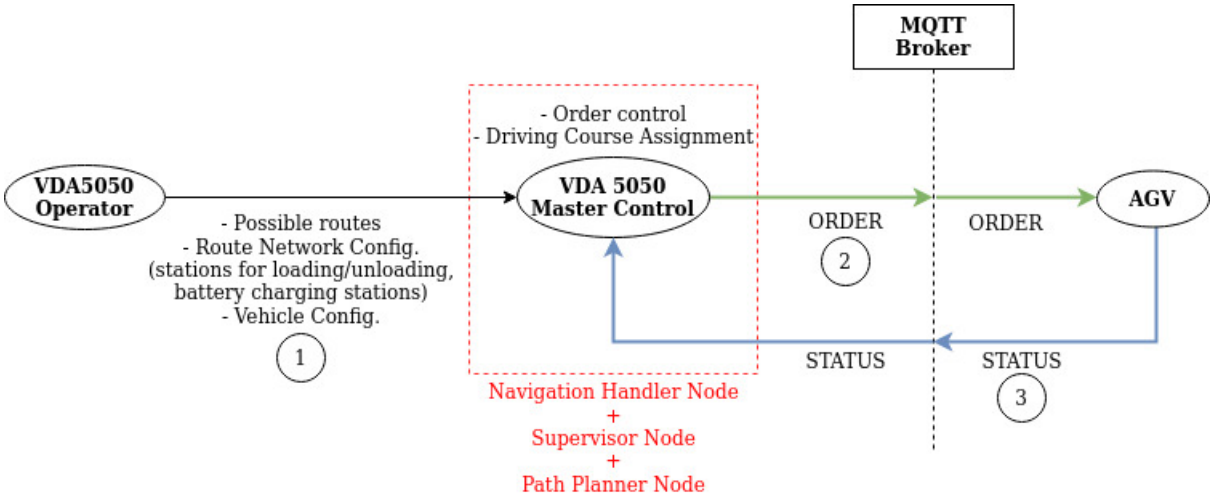


Figure 4.5: Diagram illustrating the exchange of messages to the MQTT broker between the Master Control (INS Nodes) and AGV

The connection between the Navigation Stack and the VDA 5050 standard is enabled by creating two translation nodes, one that converts the information received from the stack into VDA-formatted messages and another node that converts the VDA-formatted message into the type of information accepted by the INS. Each translation node will establish a connection to the MQTT broker and publish and subscribe to the created MQTT topics. The implementation of these nodes and the MQTT protocol is discussed in detail in the following section.

4.4 Implementation Details

This section addresses the Translation ROS Nodes implementation details and the MQTT Client Application. It provides a detailed description of the development process and how the developed algorithm works.

4.4.1 Translation Nodes

The concept behind the ability of VDA-compliant robots from different manufacturers to communicate with each other is that the vehicles can be thought of as speaking in a second language rather than their native language. Therefore, there is a need for a translation module that will allow

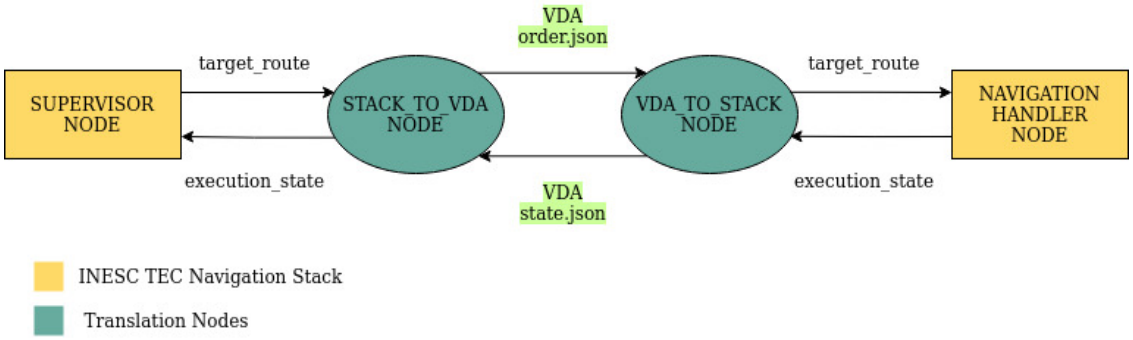


Figure 4.6: Diagram representing the exchange of messages between the Navigation Stack and the Translation Nodes

communication between a robot and the fleet manager of INS in the specific case approached in this thesis.

The translation module consists of two ROS nodes. One receives data from the Navigation Stack, transforms it into a JSON formatted message according to the VDA specification, and then publishes this information in a topic to which the second node will subscribe. The second node takes the JSON, transforms it back into INS formatted messages, and publishes that data.

To illustrate where the translation nodes are located in relation to the Navigation Stack, Figure 4.6 shows the topics to which each node subscribes and publishes. Here, exchanging information between the two nodes created is done using ROS only. Later, this transmission will be done using the MQTT protocol.

4.4.1.1 Navigation Stack to VDA 5050 Standard

This node is responsible for translating the INS formatted messages into the VDA5050 format and creating the Order JSON message. It subscribes to two topics of the Navigation Stack, one where the robot's route, assigned by the Path Planner node, is published (Target Route topic) and the other associated with the graph map.

The Order message, defined by the VDA 5050 standard, includes a list of nodes and edges that make up a path for the robot to follow, as was discussed in Chapter 3. Similarly, INS publishes a list of edges assigned to the robot in the route topic. However, since the VDA Order message includes both the nodes and the edges to be traversed, the translation node must subscribe to the Graph topic to retrieve the information about the nodes and the Target Route topic to get the edges identifier.

The Graph topic contains a list with information about the nodes and edges available in the simulation environment of the Navigation Stack. For each edge, the destination and origin nodes are informed. Moreover, for each node, data about its x, y, and theta positions are present. Using the list of edges provided by the Target Route topic makes it easy to find the corresponding destination and origin nodes.

The data from the Graph topic was first organized into three vectors to facilitate access to the information, one comprising all the edge IDs and the other two, all the destination and origin node IDs, respectively. The next step is to search for edges shared between the list of edges from the Target Route topic and the vector of edge IDs from the Graph topic. A matched edge's destination and origin nodes are accessed at the correct positions and added to the list of nodes.

It is important to note that only for the first edge of the edges list both the destination and origin nodes are retrieved and added to the list of nodes. For the remaining edges, only the destination node is added. This behavior ensures the rule, defined by the standard, that the number of edges is the number of nodes minus one is met.

Once all the required data has been collected, the Order JSON is constructed. The JSON follows the format defined by the standard, starting with the specified VDA header, then the list of nodes created in the previous step, and the list of edges retrieved from the INS Target Route topic. The Order message is sent to the second translation node as a string, simplifying the parsing.

Figure 4.6 shows that this ROS translation node also receives the State message from *vda_to_stack_node*. The content of the message is analyzed, and the significant parts are assembled into the INS-formatted message for the execution feedback and published to the appropriate Execution State topic.

4.4.1.2 VDA 5050 Standard to Navigation Stack

This other node is responsible for translating the INS formatted messages and creating the State JSON message. This time, it subscribes both to the Graph topic and the Execution State INS topic, published by the Navigation Handler node. This node uses the data it acquired from the graph topic like the other node, generating the list of nodes.

Chapter 3 introduced the contents of the VDA State message that is, in short, a list of nodes and edges left to be traversed. The Navigation Stack has its topic where it publishes a feedback message of the state of completion, Execution State topic. In this topic, each edge ID that is part of the Target Route topic list is assigned a state indicating whether the robot is moving on a particular edge, has already completed the traversal, or the traversal has been aborted.

The VDA 5050 State message only includes the nodes and edges that the robot still needs to cross, so some adjustments were needed since INS displays all the edge IDs of the path. This node selects only the edge IDs associated with the previously mentioned state where the robot is moving and stores these edges in a vector. This vector of to-traverse edges is used in the same procedure, as mentioned in the sub-section above, to find the corresponding destination and origin nodes.

With the vector of edge IDs and node IDs created in the process described above, the state JSON message can be put together following the configuration defined by the VDA 5050 standard. The message is updated each time an edge changes its state from moving to completed. This update removes the edge that changes state to complete from the list and its nodes. This node also publishes an INS-compliant State message with all the edge IDs included in the order. The JSON is sent as a string to the *stack_to_vda* node. Additionally, this node receives the Order JSON

message, and its information is translated to an INS formatted message and published to the Target Route topic.

4.4.2 MQTT Client Application

Up to this point, all the information exchange between the two nodes created was done through ROS. The VDA 5050 messages were published in the corresponding topic and retrieved when needed by subscribing to the same topic. With the development of the two translation nodes communicating via ROS messages complete, it is now possible to implement communication via the MQTT protocol, as defined by the standard.

For this purpose, a new arrangement was made. Now consider that the system is divided into supervisors and robots. Each of these elements will have both of the translation nodes running internally. Furthermore, supervisors and robots have a configuration file, as shown in Figure 4.8, that contains a unique identifier defined by the ROS namespace (*ros_id*), a list of connected robots (*robots_ids*), and a boolean parameter that indicates whether the configuration file is associated with a supervisor or not (*sup_flag*).

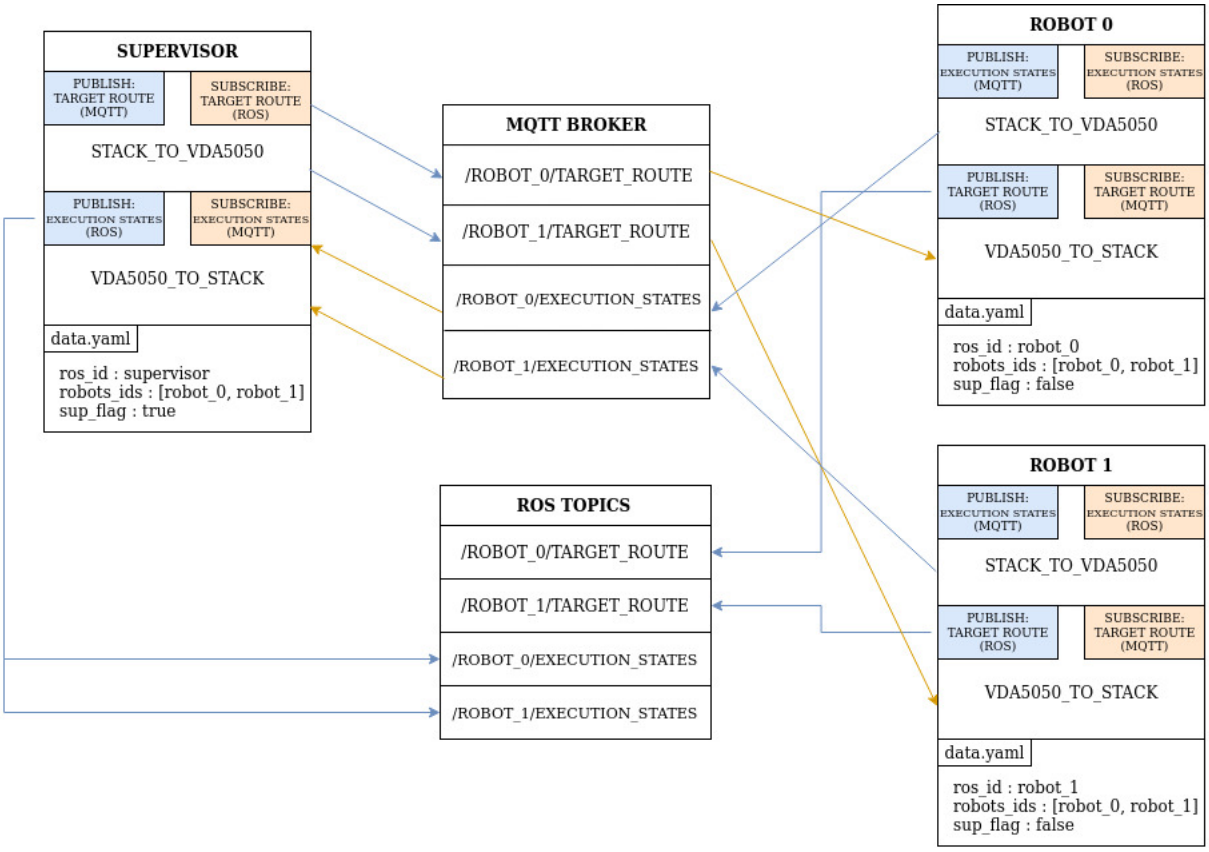


Figure 4.7: Diagram illustrating the arrangement for the MQTT Client Application and the communication with the MQTT Broker

The configuration files are YAML files that can be edited according to the desired scenario. In addition to the parameters mentioned above, this file also contains information about the MQTT

Table 4.1: Indicates the ROS and MQTT topics published and subscribed by the supervisor and robots

		ROS TOPICS		MQTT TOPICS	
		SUBSCRIBE	PUBLISH	SUBSCRIBE	PUBLISH
STACK_TO_ VDA	SUP	**/robot_0**/target_route **/agv**/target_route	-	-	**/robot_0**/target_route **/agv**/target_route
	ROBOT_0	**/robot_0**/execution_states	-	-	**/robot_0**/execution_states
	AGV	**/agv**/execution_states	-	-	**/agv**/execution_states
VDA_TO_ STACK	SUP	-	**/robot_0**/execution_states **/agv**/execution_states	**/robot_0**/execution_states **/agv**/execution_states	-
	ROBOT_0	-	**/robot_0**/target_route	**/robot_0**/target_route	-
	AGV	-	**/agv**/target_route	**/agv**/target_route	-

broker that will be used to establish the connection. Using YAML files allows for easy future extensions, as the application is dynamic and reads the parameters from the file, stores them in variables, and then uses these variables to create the client identifier, the MQTT/ROS topics to subscribe to, and the MQTT topics to publish.

data_sup.yaml

```
ros_id: supervisor
robots_ids: - robot_0
            - robot_1
sup_flag: true

broker:
    broker_address: 172.16.46.200
    username: " "
    password: " "
    qos: 1
    keep_alive_interval: 120
    clean_session: 1
    timeout: 10000
```

data_r0.yaml

```
ros_id: robot_0
robots_ids: - robot_0
            - robot_1
sup_flag: false

broker:
    broker_address: 172.16.46.200
    username: " "
    password: " "
    qos: 1
    keep_alive_interval: 120
    clean_session: 1
    timeout: 10000
```

data_r1.yaml

```
ros_id: robot_1
robots_ids: - robot_0
            - robot_1
sup_flag: false

broker:
    broker_address: 172.16.46.200
    username: " "
    password: " "
    qos: 1
    keep_alive_interval: 120
    clean_session: 1
    timeout: 10000
```

Figure 4.8: Configuration File associated with a supervisor and two robots

To implement the MQTT communication protocol, some changes had to be made to the initial structure presented in the previous sections. Now, each translation node's supervisors and robots become a different MQTT client. The main translation feature has also been modified. The MQTT clients still subscribe to the ROS topics related to the Navigation Stack, but now all the VDA formatted messages will be sent via MQTT. In the end, the messages are converted back to the INS format and published to ROS topic to communicate with the robots.

Table 4.1 and Figure 4.7 illustrates this new arrangement and how the communication with the MQTT broker will be. The supervisor extracts the information needed to create the VDA Order JSON message via the ROS topic, target_route, and sends it via the MQTT topic of the same name. The supervisor also receives continuous feedback via the execution_state MQTT topic. The robots continuously publish the VDA State JSON message via the execution_state MQTT topic and receive the route information via an MQTT topic.

Implementing an MQTT client application that can connect to the broker to publish and subscribe to topics and read the information in a configuration file to assemble the MQTT topics dynamically is possible by following the steps below, which will be explained in detail later.

1. Create a client object with a unique identifier;
2. Define the connection options (Last Will, username, password, Keep Alive, Persistent Session);
3. Set Callback functions that will become active upon the arrival of new messages to the broker;
4. Connect to the MQTT broker;
5. Publish and Subscribe to MQTT topics.

First, the MQTT Application creates a client object with the following information: a unique client identifier and the address of the MQTT broker. These two pieces of information can be obtained from the configuration file. The unique client identifier is the *ros_id* parameter plus a *"stack"* or *"vda"* string at the end to distinguish the clients from each Translation Node since the two translated nodes are launched together, and the configuration file is the same. The address of the MQTT broker is in the parameter called *broker_address*, where it contains the address that allows connecting to the broker.

For the connection to be established later, the Connection Options must be set first. The MQTT protocol defines the Connection Options, which are features that inform the broker how to behave in specific scenarios. These features are detailed below, and in Figure 4.9, it is possible to see the contents of the *CONNECT* package and the configuration used for this implementation. When connected to the broker, the clients send a connection message, defined by the VDA 5050 standard, to indicate the state of their MQTT connection. In addition, the robots send vehicle specification information (VDA 5050 Factsheet message) to the broker.

The **MQTT Last Will** feature includes a last-will message sent to other clients if a client unexpectedly loses its connection to the server. The **Keep Alive Interval** defines the maximum amount of time, in seconds, that the broker and client may not communicate with each other. For client authentication, the protocol provides an option to specify a **username** and **password**. It is also possible to define whether the client wants to establish a **persistent session** with the broker. If the client sets the *cleanSession* flag to true, the broker does not store any information for the client and discards any previous state.

Upon the arrival of new messages to the broker, a **Message Arrived callback** function is called. This callback function receives, most importantly, the MQTT topic name and the incoming message. The part of the translation module regarding transforming messages from JSON to INS format is done inside this callback function. When a client publishes a new JSON message (Order or State) to the broker, the MQTT Application calls the callback function and parses the message. There are two types of messages published by the INS, one associated with the Target Route topic and the other associated with the Execution State topic, from the MQTT topic name in which the new message was published (/.../target_route or /.../execution_states) it is possible to select the type of INS formatted message that it will be translated into and later published in the correct ROS topic.

Finally, the application calls the **Connect** function to connect to the MQTT broker with the client identifier and the previously defined connection options. Once connected, the broker keeps the connection open until the client sends a disconnect command or the connection is terminated. A reconnect process is also implemented. Each time a client loses connection to the server, the application calls the Connection Lost callback and attempts to reconnect or report the problem.

Defining the *Quality of Service (QoS)* level is essential when publishing or subscribing to a message with MQTT. The protocol defines 3 QoS levels to ensure reliability. These levels define the guarantee of delivery for a specific message. At level 0, there is no guarantee of delivery, and the message is sent at most once. Level 1 ensures that the message is sent at least once; however, the message may be sent or delivered multiple times. QoS 2 guarantees that each message is received only once and is, therefore, the most secure but the slowest quality of service level.

The MQTT publish and subscribe processes are straightforward, and the content of their packages are in Figure 4.9. The Publish function receives the message to be sent, the specific MQTT topic in which the client wishes to publish the message, and the QoS. When subscribing, the subscribe function is called with the client identifier, the intended MQTT topic name, and the QoS. After translating the INS formatted messages received from the ROS topic into VDA formatted messages, the MQTT Application calls the publish function to publish this information to the broker. If a client subscribes to this MQTT topic, the application calls the Message Arrived callback function, and the message is converted (VDA to INS) and published to a ROS topic.

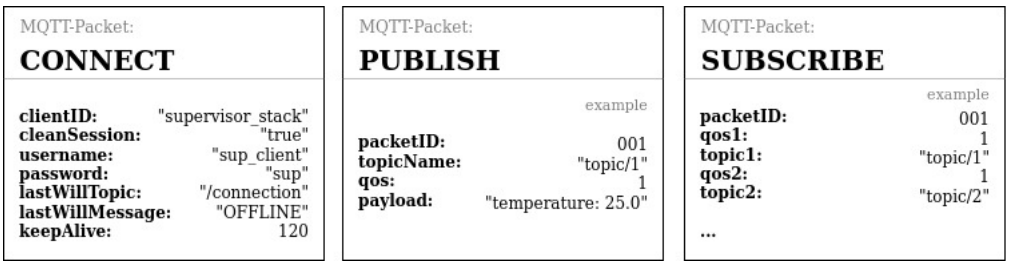


Figure 4.9: Contents of the MQTT Packages Connect, Publish and Subscribe, sent to the MQTT broker

4.5 Testing and Conclusion

In summary, this chapter deals with the implementation of the software responsible for creating a translation module that allows communication, via the MQTT protocol, between the INESC TEC Navigation Stack and its robots, through the VDA 5050 standard. The following paragraphs will answer the questions raised at the beginning of this chapter to summarize the contents presented.

To answer questions 2 and 3, the Fleet Manager, here considered as the Navigation Stack Supervisor, Path Planner, and Navigation Handler nodes, sends the route information to the robots, and the robots continuously send feedback about the execution of the order. This module uses the VDA subtopics Order, State, Connection, and Factsheet, answer to question 1.

When a new message is received in the Target Route topic, the *stack_to_vda5050* translation node creates the VDA Order message and sends it to the corresponding MQTT topic. In the same translation node, the robots publish VDA State messages to the MQTT topic to report the completion status of the assigned order. The other translation node, *vda5050_to_stack*, subscribes to the route and status MQTT topics and transforms them into ROS messages compatible with INS formatted messages. Summarizing to answer question 4, *stack_to_vda5050* node reacts to INS Target Route and Execution State topics, and *vda5050_to_stack* node reacts to new messages on the corresponding MQTT topic.

To test and evaluate the software, the MQTT X program was used. It was possible to assess whether the VDA JSON messages were sent correctly by the clients and to verify their content. The results are discussed in detail in the following chapter.

Chapter 5

Results and Discussion

This chapter presents all the results obtained while testing the developed software module. First, it gives a context of the tests, and how they were performed, then the results are presented. Finally, all the results are explained and evaluated, with a discussion of their meaning and relevance.

5.1 Tests and Results

The tests conducted included one in the simulated environment and the other in a real-life scenario. For the first test, the translation nodes and the MQTT Client application were running on a single computer, and the tests were performed by launching an instance of the two nodes for each of the three elements, the supervisor and the two robots. The simulated environment on RVIZ was used to check the order's progress and to compare the Navigation Stack's information with the VDA messages' content. The second one demonstrates the operation in a real-life scenario. A test environment consisting of INESC TEC's mobile robot and a supervisor running on an industrial computer was set up. The supervisor had a Mosquitto Broker and the translation nodes running in the background, and the robot had only the translation nodes.

5.1.1 Simulated Environment Tests

To test whether the VDA JSON messages were sent correctly and with the correct information to the MQTT broker, the supervisor first creates a new order for the robot, a simple move to a specific node. This order is then processed by the translation nodes and converted into a JSON-formatted message sent to the MQTT broker. Through the MQTTX application, it is possible to analyze the created messages and evaluate if their content is correct.

Upon initialization of the translation module (Translation Nodes and MQTT Client Application), the MQTT broker receives the VDA Connection Messages, indicating the connection status as "ONLINE" shown in Figure 5.1.

When all elements are ready, i.e., when all elements have sent the "ONLINE" message, the supervisor creates a new order for the robot, here in this example, "Go to Node 10", see Figure 5.2 and publishes it to the MQTT broker. The broker receives the VDA Order Message via

Figure 5.1: VDA 5050 Connection Messages seen through MQTTX

MQTT Topic */robot_0/target_route*, with the information formatted as defined by the standard. This message can be seen in Figure 5.3.

The other translation node, *vda5050_to_stack*, is subscribed to the MQTT Topic */robot_0/target_route* and converts it to ROS message, publishing it with the same format used by the Navigation Stack. Figure 5.4 confirms that the message received on the ROS topic at the top, published by *vda5050_to_stack* node, is the same as that at the bottom, published by INS Supervisor node.

The feedback messages received when the robot is in motion are published in the MQTT Topic */robot_0/execution_route_states*. The State VDA Messages change depending on where the robot is and which nodes and edges have already been traversed. The MQTT broker received 168 VDA State messages during the order to go to Node 10. Figure 5.5 shows the evolution of the State messages over time.

Like the VDA Order Message, the *vda5050_to_stack* node translates VDA-formatted State messages into INS-formatted feedback messages. Analyzing the ROS messages published by *vda5050_to_stack*, it is possible to see that they have the Stack format but follow the content defined by the standard, showing the edges that are yet to be traversed. These messages can be seen in Figure 5.6. Alternately, this node also publishes a Stack-compliant VDA State message,

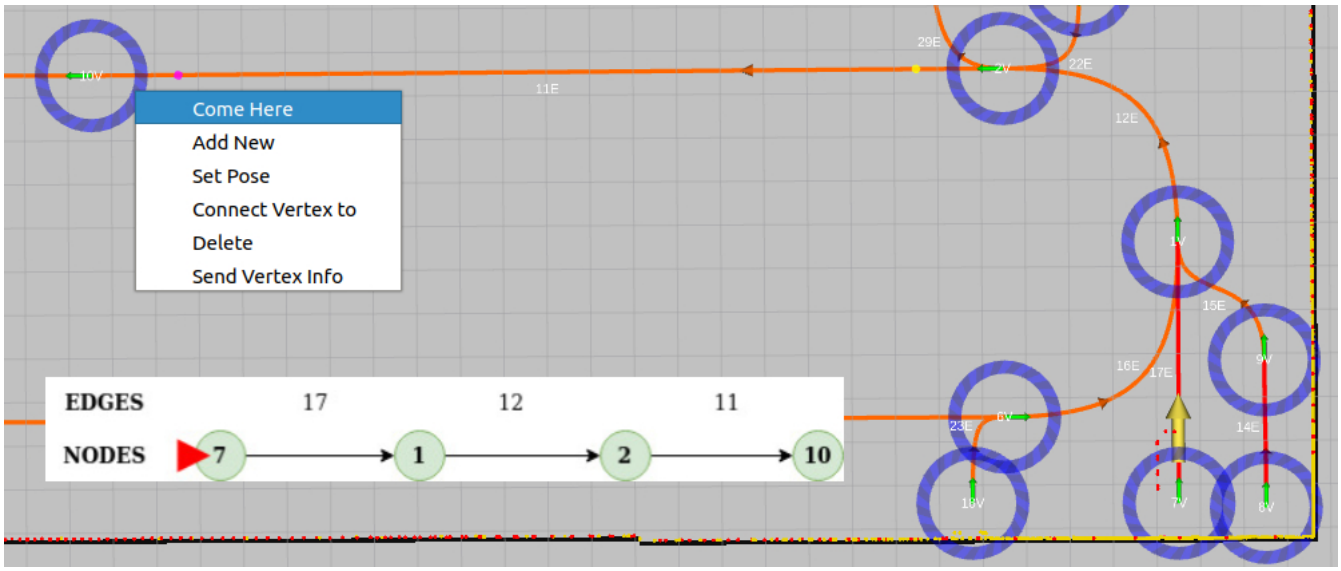


Figure 5.2: Robot path to Node 10, representation of nodes and edges

Figure 5.3: VDA 5050 Order Message seen through MQTTX

which has the same content as the messages published by the Navigation Stack, with all the edges that are part of the route, Figure 5.7 show these Stack-compliant messages.

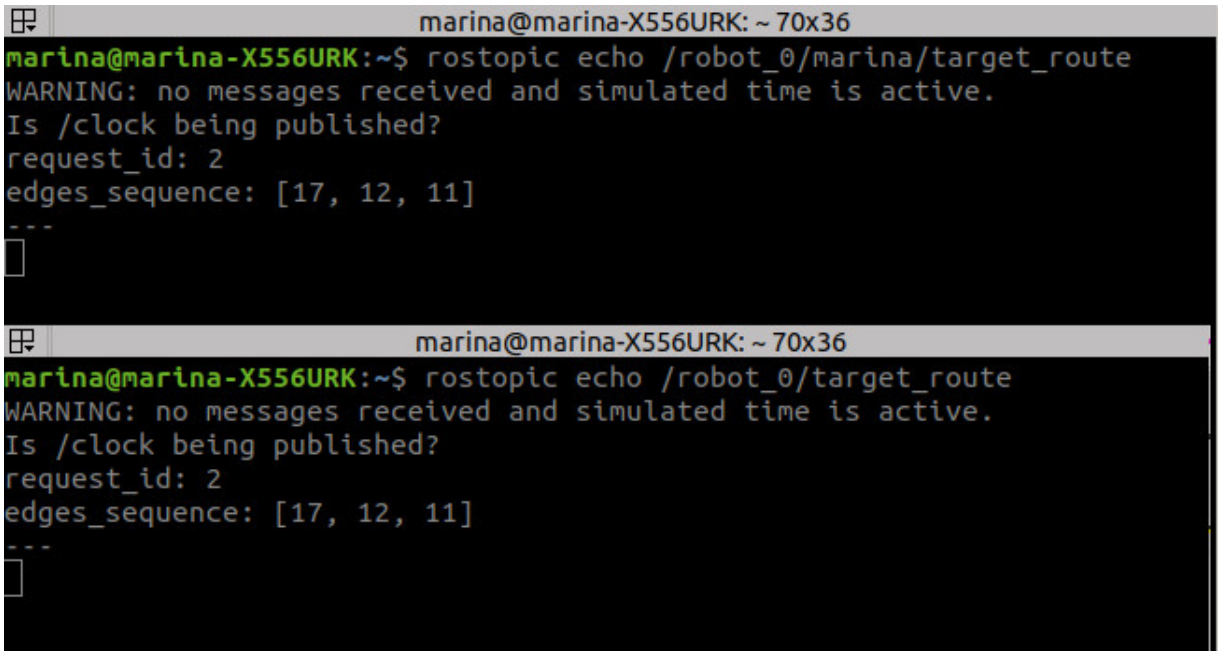


Figure 5.4: Terminal screenshot showing the two published messages of type *target_route*

5.1.2 Real-life Scenario Tests

The real-life scenario tests used a robot and an industrial computer. The robot was configured with the developed software module, which enabled information exchange using the VDA 5050 standard. The industrial computer acted as a supervisor. The supervisor also had the translation software module and the MQTT client application, which allowed exchanging messages with the robot in the VDA 5050 format.

A new map was used for this test, Figure 5.8. It was necessary to remap some of the ROS topics of the Navigation Stack to control the robot with the VDA 5050 standard. The INS ROS topic responsible for sending an order to the robot and making it move, */robot_0/target_route*, was remapped, that is, changed to the ROS topic published by the translation software module (*/robot_0/marina/target_route*).

After all the necessary preparations, a new order request was sent to the robot using a specific Navigation Stack ROS topic called */request_MAGS*. This topic receives the intended destination node, the request identification, and the robot type for the order. The fleet manager uses the contents of the */request_MAGS* topic to send the correct route information to the Target Route topic. During the test, the robot behaved as expected, and after the */request_MAGS* topic was sent, the robot moved to the same node it was commanded to.

5.2 Discussion

Like other approaches to implement the VDA 5050 standard, the developed software module defines a bridge between the VDA 5050 messages and the ROS Navigation Stack messages in the

Figure 5.5: VDA State Messages seen through MQTTX

form of a translation software module. The main objective of the VDA 5050 translation software is to control a mobile robot developed entirely by CRIIS and commanded by the Navigation Stack. This is made possible by the translation node *vda5050_to_stack*, which converts the VDA-formatted messages into ROS messages readable by the INS.

The tests confirmed that the mobile robot can be controlled via VDA 5050 standard. The software module developed for this purpose correctly receives the information from the Navigation Stack, converts it to the appropriate VDA 5050 JSON message, sends it to the MQTT broker, and then converts it back to the INS formatted messages. No change was observed when comparing the robot's behavior before and after implementing the VDA 5050 standard.

During the implementation phase, several tests were performed to confirm that the messages sent by the translation module to the robot were in the format the robot would accept. Particular attention was paid to the messages sent to the Target Route topic since these are responsible for commanding the robot where to go. The preliminary tests showed that the messages published by

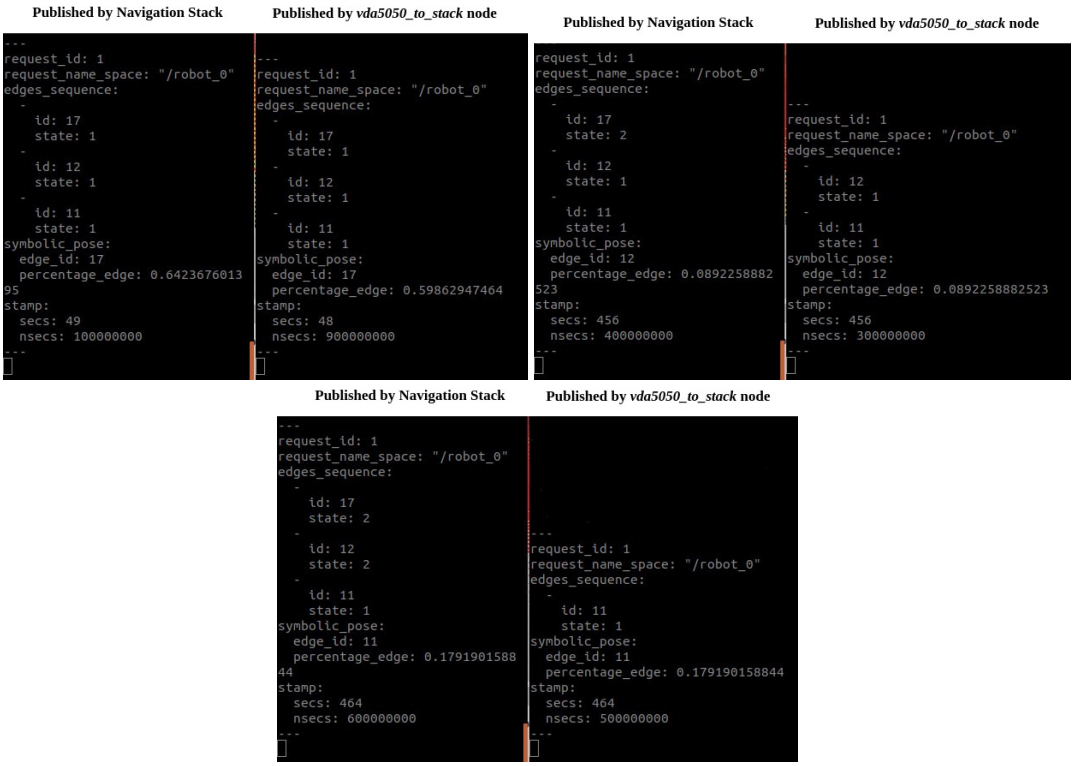


Figure 5.6: VDA-compliant messages published by the translation node on the right, compared to messages published by the Navigation Stack during order execution

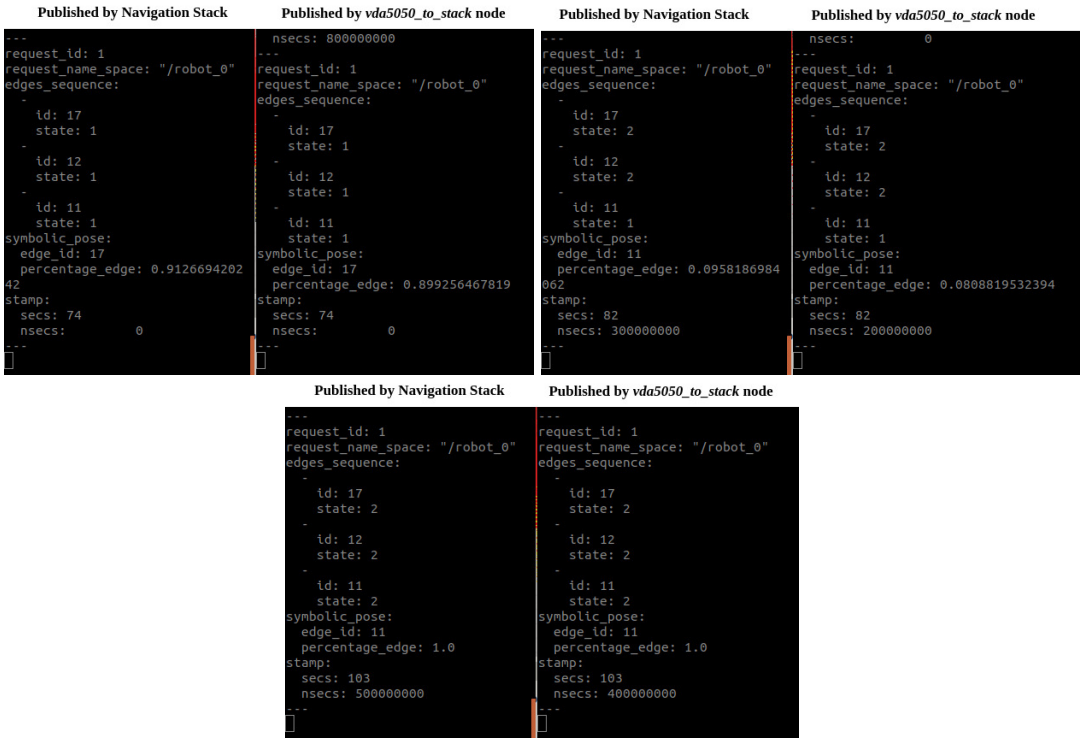


Figure 5.7: Stack-compliant messages published by the translation node on the right in comparison to messages published by the Navigation Stack

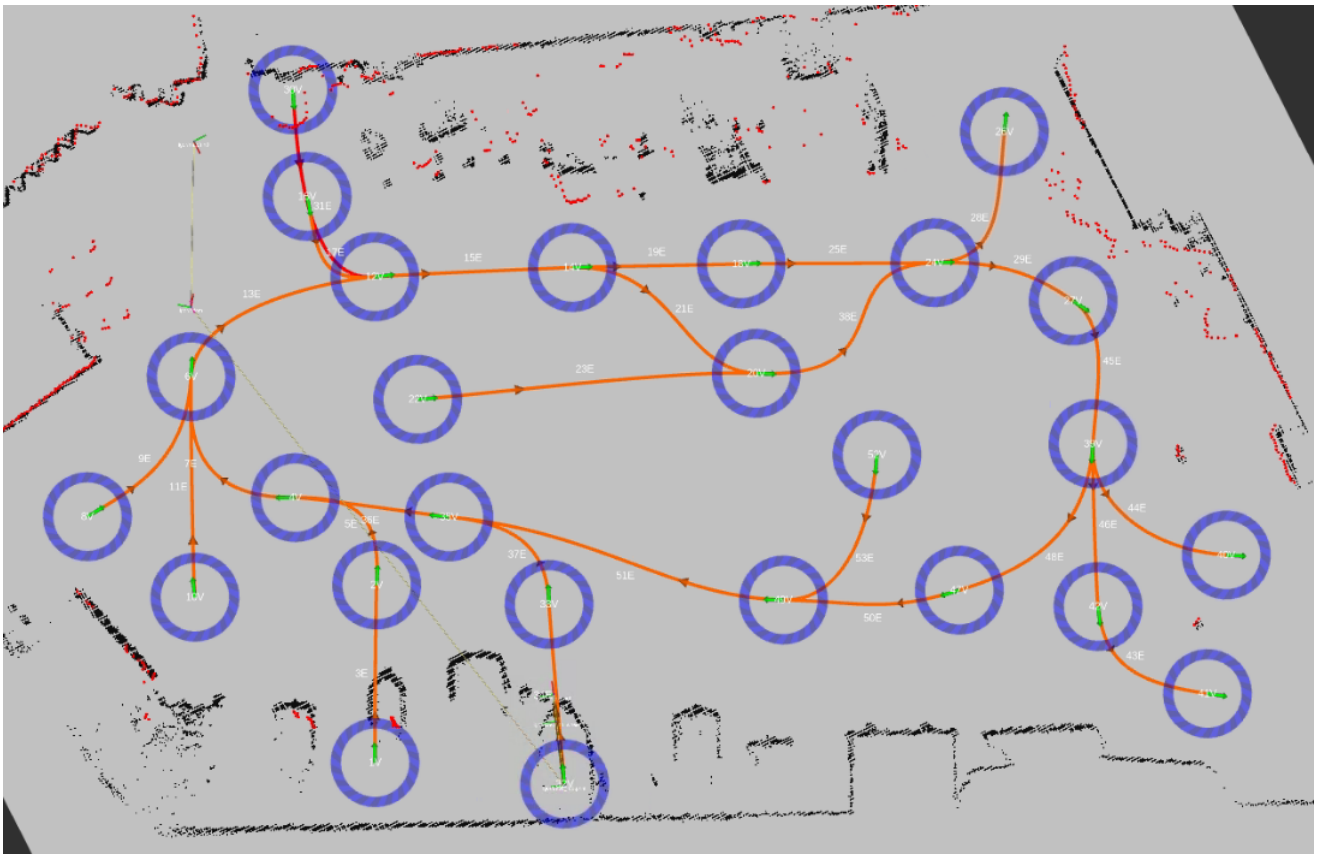


Figure 5.8: Map used in the real-life scenario test

the Navigation Stack and published by the Translation Software Module were identical and that the robot could receive them and act accordingly.

The data extracted from the tests contribute to a clearer understanding of how the VDA 5050 can control a ROS-based mobile robot developed entirely by CRIIS and how the information provided by the Navigation Stack can be used to create the VDA 5050 messages. The scope of this thesis focuses only on controlling the robot, i.e., sending simple commands that only require the robot to go to the specified node. Therefore, sending actions to the robot, such as picking up and dropping off at a specific location, is not part of the context.

Chapter 6

Conclusion and Future Work

This research aimed to develop a software module capable of managing and controlling a ROS-based mobile robot with INESC TEC's fleet management system, Navigation Stack, through the VDA 5050 standard and its defined communication protocol, MQTT. The main objective was to develop software consisting of a translation module and an MQTT client application. The translation module is responsible for converting the INS-formatted messages to VDA-formatted messages and vice-versa and transmitting them via MQTT.

To this end, an in-depth study of the VDA-defined JSON messages and the MQTT protocol was conducted. The study helped to understand where extracting the necessary information to assemble these messages would be possible. Prior knowledge of the Navigation Stack was required to determine which ROS nodes and topics were the most relevant in this context. The MQTT protocol was implemented to exchange messages between the robots and the central master control.

The results confirmed that it is possible to implement the VDA 5050 standard and control the robot by sending it the messages defined in the standard. The translation module developed for this purpose acts as a bridge between the ROS-based robot and system and the VDA 5050 standard. The MQTT client application proved successful and fully reconfigurable to any situation since it uses a configuration file to create the MQTT topics.

This research is limited to managing and controlling a ROS-based mobile robot controlled by INESC TEC's fleet management system. If a non-ROS-based robot and a different fleet manager are used, it is not possible to use the designed translation module to control this type of robot. However, the same approach, a translation module, can be replicated to implement the VDA 5050 standard in this scenario.

It is important to note that the VDA 5050 standard has yet to be finalized. Therefore, when the final version of the standard is released, any current VDA 5050-compliant AGV implementation will need to be updated or even completely redesigned.

In terms of future work related to this thesis, it is necessary to incorporate the Instant Actions messages and the Action field within the Order message to implement the standard fully. These messages require a more complex study and even changes to the Navigation Stack to be successfully implemented. Furthermore, it would be necessary to have mobile robots of different brands

sharing the same space and communicating via VDA 5050 in order to fully test the possibilities of the standard and the developed software.

Ultimately, the effort to develop these interoperability standards makes it possible to operate complex heterogeneous multi-robot systems in a real-world scenario. The work developed in this thesis further proves that it is possible to implement these standards, in this case, VDA 5050, and achieve good results. Although the current version of VDA 5050 is limited to communicating instructions to AGVs, vehicle specifications (fact sheet sharing), and sending actions, it is a step forward to solving the interoperability problem.

Bibliography

[1] Meili Robots, "Fully integrated fleet management system." meilirobots.com https://www.meilirobots.com/product (accessed June 23, 2023).

[2] Open Robotics, "Programming multiple robots with ROS 2." osrf.github.io. https://osrf.github.io/ros2multirobotbook/ (accessed Mar. 1, 2023).

[3] Q. R. Lia, Z. Dydek, and D. Theobald, "Why interoperability is critical to the warehouse of the future," Presented at the ISR Europe 2022; 54th International Symposium on Robotics, Munich, Germany, 2022, pp. 1-7. [Online]. Available: https://ieeexplore.ieee.org/document/9861822.

[4] "What is opc ua? a practical introduction." opc-router.com. https://www.opc-router.com/what-is-opc-ua/ (accessed June 26, 2023).

[5] German Association of the Automotive Industry, "Interface for the communication between automated guided vehicles (AGV) and a master control VDA 5050 Version 2.0.0." vda.de. https://www.vda.de/en/news/publications/publication/vda-5050-version-2.0.0-agv-communication-interface (accessed Mar. 1, 2023).

[6] "Practical test passed: Interface for Driverless Transport Systems." exxpo.com. https://www.exxpo.com/en/automation/practical-test-passed-interface-for-driverless-transport-systems/ (accessed May 26, 2023).

[7] IBM, "What is industry 4.0 and how does it work?." (accessed May 10, 2023).

[8] R. Cupek, M. Drewniak, M. Fojcik, E. Kyrkjebø, J. C.-W. Lin, D. Mrozek, K. Øvsthus, and A. Ziebinski, "Autonomous guided vehicles for smart industries – the state-of-the-art and research challenges," in *Computational Science – ICCS 2020*, (Cham), pp. 330–343, Springer International Publishing, 2020.

[9] T. Kathmann, D. Reh, and J. C. Arlinghaus, "Exploiting the technological capabilities of autonomous vehicles as assembly items to improve assembly performance," *Advances in Industrial and Manufacturing Engineering*, vol. 6, p. 100111, 2023. Accessed: Apr. 3, 2023.

[10] H.-C. Hwang, J. Park, and J. Shon, "Design and implementation of a reliable message transmission system based on mqtt protocol in iot," *Wireless Personal Communications*, vol. 91, 12 2016.

[11] HiveMQ, "Getting started with MQTT." https://www.hivemq.com/blog/how-to-get-started-with-mqtt/ (accessed May 19, 2023).

[12] "Mqtt 5 specification." oasis-open.org. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed June 9, 2023).

[13] I. Craggs, "Eclipse paho: The eclipse foundation." https://www.eclipse.org/paho/ (accessed May 17, 2023).

[14] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[15] A. Khamis, A. Hussein, and A. Elmogy, *Multi-robot Task Allocation: A Review of the State-of-the-Art*, vol. 604, pp. 31–51. 05 2015.

[16] J. Gómez and A. Barrientos, "Multi-robot systems: Challenges, trends and applications," *Applied Sciences*, Mar 2022. Accessed: May 11, 2023.

[17] Y.-L. Liao and K.-L. Su, "Multi-robot-based intelligent security system," *Artif. Life Robot.*, vol. 16, pp. 137–141, 09 2011.

[18] K. Nagatani, Y. Okada, N. Tokunaga, K. Yoshida, S. Kiribayashi, K. Ohno, E. Takeuchi, S. Tadokoro, H. Akiyama, I. Noda, T. Yoshida, and E. Koyanagi, "Multi-robot exploration for search and rescue missions: A report of map building in robocuprescue 2009," in *2009 IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR 2009)*, pp. 1–6, 2009.

[19] G. A. . F. C. Alessandro Marino, Lynne E. Parker, "A decentralized architecture for multi-robot systems based on the null-space-behavioral control with application to multi-robot border patrolling," *Journal of Intelligent & Robotic Systems*, vol. 71, p. 423–444, 2013.

[20] M. V. "Espina, R. Grech, D. De Jager, P. Remagnino, L. Iocchi, L. Marchetti, D. Nardi, D. Monekosso, M. Nicolescu, and C. King, *"Multi-robot Teams for Environmental Monitoring"*, pp. 183–209. Berlin, Heidelberg: "Springer Berlin Heidelberg", 2011. Accessed: May 17, 2023. [Online]. doi: 10.1007/978-3-642-18278-5_8.

[21] C. Liu and A. Kroll, *A centralized multi-robot task allocation for industrial plant inspection by using A* and genetic algorithms*, vol. 7268 LNAI. 2012.

[22] B. Coltin and M. Veloso, "Mobile robot task allocation in hybrid wireless sensor networks," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2932–2937, 2010.

[23] S. Omidshafiei, A. Agha–Mohammadi, C. Amato, S. Liu, J. P. How, and J. Vian, "Decentralized control of multi-robot partially observable markov decision processes using belief space macro-actions," *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 231–258, 2017.

[24] S. Giordani, M. Lujak, and F. Martinelli, "A distributed multi-agent production planning and scheduling framework for mobile robots," *Computers and Industrial Engineering*, vol. 64, no. 1, p. 19 – 30, 2013. Cited by: 71.

[25] P.-a. Gao, Z.-x. Cai, and L.-l. Yu, "Evolutionary computation approach to decentralized multi-robot task allocation," in *2009 Fifth International Conference on Natural Computation*, vol. 5, pp. 415–419, 2009.

[26] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics," *Scholarpedia*, vol. 9, no. 1, p. 1463, 2014. revision #138643.

[27] M. Robots, "Interfleet Software Integration." Meili Robots, Copenhagen, Denmark, 2022. Accessed: June, 23, 2023. [Online]. Available: https://www.meilirobots.com/resources-list/report/interfleet-software-integration.

[28] InOrbit, Inc. inorbit.ai. https://www.inorbit.ai/ (accessed June 23, 2023).

[29] InOrbit, Inc., "FAQ." inorbit.ai. https://www.inorbit.ai/faq (accessed June 23, 2023).

[30] Spherical Insights & Consulting, "Global autonomous mobile robot market size, share, and covid-19 impact analysis, by component (hardware, software, and services), by type (goods-to-person picking robots, self-driving forklifts, autonomous inventory robots, and unmanned aerial vehicles), by battery type (lead battery, lithium-ion battery, nickel-based battery, and others), by end-use (manufacturing and wholesale & distribution), by region (north america, europe, asia-pacific, latin america, middle east, and africa), analysis and forecast 2022 – 2032.." sphericalinsights.com. https://www.sphericalinsights.com/reports/autonomous-mobile-robot-market (accessed June 20, 2023).

[31] "Apple and Google partner on covid-19 contact tracing technology." Apple Newsroom. https://www.apple.com/newsroom/2020/04/apple-and-google-partner-on-covid-19-contact-tracing-technology/ (accessed June 18, 2023).

[32] J. Mathews, J. Rachner, L. Kaven, D. Grunert, A. Göppert, and R. Schmitt, "Industrial applications of a modular software architecture for line-less assembly systems based on interoperable digital twins," *Frontiers in Mechanical Engineering*, vol. 9, 02 2023.

[33] M. Štufi, A. Panajotović, and L. Stoimenov, "Designing distributed controlling testbed system for supply chain and logistics in automotive industry," *Facta Universitatis, Series: Automatic Control and Robotics*, vol. 1, no. 3, pp. 147–157, 2022.

[34] "New revolution in autonomous mobile robots (amr) based on vda 5050." resmanio.com. https://resmanio.com/2020/07/27/new-revolution-in-autonomous-mobile-robots-amr-based-on-vda-5050/ (accessed June 21, 2023).

[35] InOrbit, "VDA5050 connector." github.com. https://github.com/inorbit-ai/ros_amr_interop/tree/galactic-devel/vda5050_connector#readme (accessed June 21, 2023).

[36] NVIDIA, "Isaac Mission Dispatch." github.com. https://github.com/nvidia-isaac/isaac_mission_dispatch (accessed June 22, 2023).

[37] "Otto motors adds support for vda 5050 mobile robot interoperability standard." robottics447.com. https://www.robotics247.com/article otto_motors_adds_support_vda_5050_mobile_robot_interoperability_standard (accessed July 3, 2023).

[38] M. Oitzman, "First public demo of massrobotics interoperability standard." mobilerobotguide.com. https://mobilerobotguide.com/2021/10/18/first-public-demo-of-massrobotics-interoperability-standard/ (accessed June 26, 2023).

[39] OPC Foundation, "What is OPC?." opcfoundation.org. https://opcfoundation.org/about/what-is-opc/ (accessed May 17, 2023).

[40] T. Berndorfer, "How to make sense of machine data in smart factories," Oct 2019.

[41] A. García and F. Fraile, "Reference models and standards for the integration of mobile robotics for internal logistic applications," *Proceedings http://ceur-ws. org ISSN*, vol. 1613, p. 0073, 2022.

[42] International Electrotechnical Commission (IEC 62264-1), "Enterprise-control system integration - Part 1: Models and Terminology," 2013.

[43] International Electrotechnical Commission (IEC 62264-3), "Enterprise-control system integration — Part 3: Activity models of manufacturing operations management," 2016.

[44] N. van Duijkeren, L. Palmieri, R. Lange, and A. Kleiner, "An industrial perspective on multi-agent decision making for interoperable robot navigation following the VDA5050 standard," in *Proceedings of the 2022 IROS Workshop on Decision Making in Multi-Agent Systems (DMMAS '22)*, (Kyoto, Japan), October 2022.

[45] F. Redaktion, "VDA 5050." flexus.de. https://www.flexus.de/en/glossary/vda-5050/ (accessed Mar. 21, 2023).

[46] ""Stress test" for VDA 5050." vdma.org. https://www.vdma.org/viewer/-/v2article/render/77894657 (accessed May 30, 2023).