

Orchestration of Network Applications Using Middleware for 5G Enhanced Autonomous Robots

Mr Bartosz Bratus

This is a digitised version of a dissertation submitted to the University of Bedfordshire.

It is available to view only.

This item is subject to copyright.

Orchestration of Network Applications Using Middleware for 5G Enhanced Autonomous Robots

Bartosz Bratus (1807956)

MSc Thesis

University of Bedfordshire

MSc by Research

Full Time

DS: Dr R. Qiu, Sup 2: Dr E. Liu

Word count: 18988

I, Bartosz Bratus, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have drawn on or cited the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- Where the thesis or any part of it is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- None of this work has been published before submission

Abstract

The demand for multi-purpose solutions grows in the era of the dynamic evolution of cloud-native solutions and robotic applications. The cloud-native applications allow flexibility regarding the number of deployed resources and the variety of available solutions. Currently, robotic applications require a monolithic approach to the solutions presented, which is the opposite of the microservices architecture that is the most popular approach when creating cloud-native solutions. The movement of the robotic software to the cloud accelerated by 5G networks faces the problem of resource orchestration from the perspective of the dynamic changes of the objectives and the complex domain processes. As the movement to the cloud happens, there will be a need for dedicated resource and service orchestration methods that will enable end-to-end communication between the vertical applications and the cloud platforms. The dynamic orchestration and the optimisation of the resources could potentially be one of the biggest challenges that the movement of the robotic network applications to the cloud will face.

1 Table of Contents

Abstract 3
1 Table of Contents 4
2 Introduction 6
2.1 Overview 6
2.2 Main challenges 7
2.3 Problem definition 7
2.4 Methodology 8
2.5 Evaluation 9
2.6 Thesis overview 10
3 Literature Review 11
3.1 Software Architecture Models 11
3.1.1 Monolith 11
3.1.2 Microservices 12
3.2 Distributed Monolith 14
3.3 Cloud computing service models 15
3.3.1 Infrastructure as a Service 15
3.3.2 Platform as a Service 15
3.4 Existing solutions 16
3.4.1 Kubernetes 16
3.4.2 OSM 17
3.4.3 Infrastructure as Code 17
3.4.4 ROS 18
3.4.5 NetApps 19
3.5 5G Network 20
3.6 Summary 21
4 Middleware Design 23
4.1 General overview 23
4.2 Middleware architecture 25
4.2.1 Architecture description 25
4.3 Communication between layers 28
4.3.1 Basic Communication 28
4.3.2 Middleware and Resource Network Layer 28

4.3.3 Middleware and Vertical Application Network Layer 28
4.3.4 Components description and functionality 30
4.3.5 Gateway 31
4.3.6 Redis Interface 31
4.3.7 Task Planner 32
4.3.8 Resource Planner 34
4.3.9 Orchestrator 35
4.4 Summary 41
5 Evaluation 42
5.1 Current Middleware Edition 42
5.1.1 Middleware example 43
5.2 Middleware improvements 47
5.2.1 Middleware and OSM 47
5.2.2 Middleware and Kubernetes 48
5.2.3 Combining OSM and Kubernetes 49
5.2.4 Benefits for ROS applications 50
5.2.5 Expected optimizations 51
5.3 KPI assessment 51
5.4 Summary 53
6 Conclusion 54
6.1 Future Directions 55
7 Appendix 56
7.1 5G-ERA Middleware GitHub repository 56
8 References 57

2 Introduction

2.1 Overview

The deployment of autonomous robots is expected to be accelerated by cloud and 5G networking (Qiu, et al., 2022). On the one hand, the 5G network extends its range and coverage worldwide. The telecommunication operators grow their networks by trying to offer state-of-the-art technology to more and more vertical applications. On the other hand, orchestration and software-defined networks deliver customised deployment platforms that can further optimise the performance of 5G infrastructure delivery. The 5G technology has an advantage over the previous generation networks with its improved speed and latency. It also increases the capability of scaling up robot deployment, which means that more devices can be connected simultaneously with improved resource management and service delivery. For example, the 5G platform will allow the robots to seamlessly integrate into the IoT and Edge computing environment and enable them to utilise cloud-based resources effortlessly. This thesis focuses on deploying digital platforms for 5G-enhanced autonomous robots with the Network Applications (NetApps) orchestrated by the 5G-ERA Middleware (Middleware). Innovations will be realised by improved resource orchestration, enhanced vertical applications enablement, scalable service delivery and effective platform management for 5G-enhanced cloud robotics.

NetApps, in the context of the 5G System, is defined as a set of services that provide certain functionalities to the verticals and their associated use cases (5GASP, 2021). It is a separate middleware layer and common services to simplify the implementation and deployment of vertical systems (autonomous robots in the thesis) on a large scale (5G-PPP Software Network Working Group, 2022). With the middleware embracing the service-based architecture and cloud-native design paradigms, the NetApps, like Middleware, can be utilised to translate current state-of-the-art robots deployed mainly for local applications towards large-scale robot deployment under a Platform as a Service paradigm.

The Orchestration is the automated configuration, coordination and management of computer systems and software (Earl, 2005). In the context of this document, orchestration is essential for designing, creating, and delivering end-to-end NetApps required by the robot deployment at a large scale under distributed environments. Furthermore, it is a method of implementing on-demand delivery of the NetApps and realising their lifecycle management.

The 5G-ERA Middleware is a virtual platform between the digital infrastructure and the robots. Instead of a robot talking with the digital infrastructure directly, which can be sub-optimal and unmanageable at a large scale, the robots communicate with the virtual platform to schedule the deployment of the services for task completion under the 5G-ERA Middleware through orchestrated NetApps. In the scenario where the robot utilises the distributed services, the Middleware delivers the customised platform optimised for the specific use case the robot has requested by intelligently orchestrating the NetApps. The Middleware can recognise the task and its critical elements to provide the optimised platform for the robot that will work at scale. At the same time, the Middleware works with infrastructures such as Edge, Cloud, and 5G Soft Defined Networks on behalf of the robots to enable efficient communication and optimise the consumption of the infrastructure. The additional layer of resource management

enabled by the Middleware provides the enhanced scalability of the distributed robot applications. Using Kubernetes as an orchestration base enables the almost infinite scaling of the applications with simplified delivery procedures. The 5G-ERA Middleware realises the cloud-native NetApp design by providing a vertical-oriented Platform as a Service (PaaS) instead of the classic 5G testbed-oriented Infrastructure as a Service (IaaS).

2.2 Main challenges

The scalability and transparency of robot applications have been improved significantly in the past decade (Qiu, et al., 2022). In particular, the robotic solutions built with the Robot Operating System (ROS) provide a good way of creating monolithic vertical applications and hardware abstraction used in the local network. However, the ROS applications were designed for communication solely in the local networks that use peer-to-peer communication over OSI Layer 2 (Quigley, et al., 2009). This method of communication was not intended to be used in distributed systems. Unfortunately, the monolithic approach to building vertical applications is not cloud-ready due to the high coupling between the robotic software and the hardware and software of the robot (Tellez, 2019). To fully utilise the capabilities of the robots and the 5G network, robots must be aware of the surrounding digital infrastructure, such as Cloud and Edge and operate the infrastructure as much as possible. The NetApps are designed to solve this issue. They allow the robot to use the distributed resources to achieve the same result as they would run locally on the robot platform. However, the development and the deployment of the NetApps are mainly driven by the Networking community with little vertical user participation. The gap led to ineffective policy being delivered by the infrastructure on resource provision. Furthermore, most vertical applications are unaware of the need for external resources during the distribution. Therefore, transparency on the infrastructure is expected to be delivered to assist the end-users. However, the modern 5G networks and testbeds are not designed to be application-transparent, and the quality of experience can differ from place to place.

2.3 Problem definition

In the current situation, NetApp development is infrastructure-oriented for network operators. The resource provision is based on the infrastructure where the programs are running. The resources are provisioned under the Infrastructure as a Service (IaaS) model using Management and Orchestration (MANO) with solutions like Open-Source MANO (OSM) (ETSI OSM, 2020). Under the existing paradigm, the applications are developed for network operators around the pre-existing static infrastructure. It often forces robot developers to learn about the topology and modify their existing applications to function correctly in a given environment.

With the orchestration of NetApps using 5G-ERA Middleware for 5G enhanced autonomous robots, the problem is how to shift the network operator cantered NetApp development to user/vertical centric NetApp development. The transition will be helped by a Platform as a Service (PaaS) paradigm, where the Middleware can adapt to infrastructure by deploying the NetApps using a virtual platform which is adjusted dynamically to the current environment and vertical requirement. Middleware needs to organise and integrate the resources into a

single unified platform that can handle different vertical applications and distributed environments efficiently and effectively.

In the scenario that the fleet of autonomous robots is utilising cloud-based resources, the middleware provides a set of benefits. Under the current robotic environment, the changes to the configuration of the robot and its services require manual intervention. The movement of the robot resources could improve the management process of robots and their services. The Middleware would provide the robotic Platform as a Service that facilitates the management process of the services running inside the robot by offloading them from the robot to the cloud. Managing the resources on the cloud level enables the configuration change of the robot fleet by modifying a single service configuration.

While providing a customised platform for robotic applications, Middleware ensures the availability of the desired services for the robot. By delivering the end-to-end experience, Middleware focuses on providing the robots with all the capabilities necessary for executing the task with even greater capacities than during the local processing. Furthermore, the Middleware offers the simplified deployment process of the robot NetApps with lifecycle management and customisable resource planning capabilities that enable the robot fleet to be utilised at a large scale.

The research aims to tackle the problem of providing a dedicated orchestration process for third-party applications by utilising domain knowledge to provide the paradigm of bridging these concepts together. As a result, improved resource orchestration, enhanced vertical applications enablement, and effective platform management for 5G-enhanced cloud robotics will be realised.

2.4 Methodology

This research addresses the problem as follows:

First, it enables 5G-ERA Middleware to tackle the transparency and scalability of the infrastructure by using the PaaS using the Kubernetes-based platform rather than OSM-based IaaS. The research explores the difference between stateless and stateful NetApps. Each of them has different needs and use cases. The Middleware will be optimised to handle both types of applications.

Second, to optimise IaaS specifically for the vertical-oriented lifecycle, event management will be explored under the PaaS. In particular, it will be verified by coordinating the information flow between the robot and the Middleware using the ROS Action Server and Action Client. As a result, the Middleware continuously communicates with the robot and informs it about the state of the utilised NetApps. On the other hand, Middleware also receives the Heartbeat information from the NetApp to gain insights into the operation of the NetApp and later use the knowledge to optimise the resources needed.

Third, the research aims to provide the simplified deployment of the custom NetApps that will enable the quicker onboarding process of the robot. On top of this, the on-demand deployment allows the robot to be more flexible regarding the task it conducts. By handling the deployment process and the lifecycle management, 5G-ERA Middleware should improve

the robot developer experience by providing the infrastructure needed to test the vertical application in the production environment.

The 5G-ERA Middleware aims to provide the highest possible Quality Of Experience (QoE) for vertical applications. The QoE can be defined as a set of properties that ties together user perception, experience, and expectations of application and network performance, typically expressed by the quality of service parameters (Fiedler, et al., 2010).

Finally, the innovations will be evaluated and verified by selected use cases under a large-scale distributed environment.

2.5 Evaluation

The evaluation process of the 5G-ERA Middleware can be complicated and extend to many areas. Therefore, the evaluation will focus on assessing the functionality of the Middleware. The assessment will cover the aspects of the Middleware, like dedicated resource orchestration, lifecycle management improvements, and how it performs as a user/vertical-centred platform. Furthermore, the benefits that a large-scale distributed environment brings will be described.

The transparency of the 5G-ERA PaaS paradigm will also be examined. This means that the Middleware should be able to run both in the local Edges and the public cloud environments like Amazon AWS or Azure without modifying the application logic on a vertical level.

The evaluation will address the set of KPIs that will indicate if the system has met the desired functionality. The desired functionality can be either provided by the Middleware itself or by wrapping the functionality of other state-of-the-art applications.

5G-PPP Test, Measurement, and KPIs Validation (TMV) Working Group was founded as part of the 5G PPP effort to promote commonalities across projects that have a strong interest in the Time and Material (T&M) methodologies needed to provide support to the vertical use cases in the 5G Trial Networks (5G-PPP Software Network Working Group, 2022). The white paper defined three types of assurances in 5G vertical application:

1. Viable performance assurance addresses the needs of the business operations and customers that depend on the underlying network performance. This type of assurance tries to provide network reliability on sustained high throughput and real-time communication services.
2. Predictable performance assurance addresses the behaviour of the Network Applications under the designated networking conditions. The objectives of the services are guaranteed through the Service-Level Agreement (SLA) and are provided by the 5G network. This performance assurance can be applied to a variety of applications and conditions
3. Last, the plug-and-play performance assurance enables the operation of vertical applications at scale with the expectation of interoperability and sustained performance levels through the variety of 5G networks worldwide

The T&M in the report will be conducted on the user side and according to the user experience based on the above KPU performance.

1. Viable performance assurance:
   1.1. 5G ERA Middleware realises native digital resource orchestration by connecting cloud-based virtual resources
   1.2. Enables the provision of the 5G network over the testbeds
   1.3. Flexible placement of the services in the available digital infrastructure
   1.4. Middleware increases QoE for vertical applications by providing increased resource availability and network connectivity with the best performance available
2. Predictable performance assurance:
   2.1. Middleware realises semantic resource planning with the integration of domain knowledge
   2.2. Allows robotic fleet management at a large scale by providing end-to-end connectivity between the vertical applications and online services
   2.3. Provides simple service orchestration management
   2.4. Middleware enables automatic service scalability
3. Plug-and-play performance assurance
   3.1. Integrates vertical-level applications with the resource provision under a unified platform
   3.2. QoE realised by providing a seamless deployment service provision through a dedicated orchestration process
   3.3. Middleware acts as a bridge between existing ROS applications to provide an easier transition to utilise the cloud-based services

2.6 Thesis overview

The later parts of this document will cover the literature review in which the existing solutions that can be used as building blocks to the 5G-ERA Middleware will be identified. On top of this, the chapter will explain the connection between state-of-the-art solutions to 5G-ERA Middleware and their role in achieving Middleware's goal. The next chapter will present in detail the idea behind the Middleware. It will cover the functionality and the processes behind them and how the Middleware utilises multiple services to achieve the end goal. Afterwards, the architecture of the Middleware will be explained as to how it interconnects with services like Kubernetes, OSM and Machine Learning tools for resource prediction. Finally, the data design will be presented and how data replication improves the quality of the service for the robots by allowing quick and unified access to the same data.

In the end, the process of testing and evaluating the 5G-ERA Middleware will be presented. This chapter will cover the process and results of integrating the Middleware with the real NetApps and the robot. In the beginning, the Middleware will be connected to NetApps to verify that their integration is happening correctly and to ensure cooperation.

In the end, the summary chapter will recap the goals of Middleware and NetApps. Finally, the evaluation scenario will assess if Middleware has met its objectives and what the future holds for Middleware.

3 Literature Review

The Literature Review chapter will cover the existing solutions that give a better understanding of what 5G-ERA Middleware is. In addition, this chapter will review and assess the building blocks and related technologies.

In this section, the basic software architecture models will be covered. These will be monolithic architecture, microservices architecture model and the hybrid of them, the distributed monolith. Further on, the software service types will be analysed. Finally, the section will focus on the IaaS and PaaS models. These models will give a better understanding of Middleware in terms of its role as a whole system and how it differs from other existing solutions.

The following will describe the current state-of-the-art solutions for creating robotic software, cloud computing and orchestration services. They will be explained, analysed and assessed to determine how they fit into the Middleware and whether they are the correct solution for the presented problem. Furthermore, the solutions described will cover the dedicated orchestration mechanisms and how they can be used for resource optimisation.

In the end, the chapter will cover the 5G network itself. How does it differ from the previous generations' networks, and what are its benefits, especially for robotic applications? The chapter will also cover the problems identified with the current cellular network generation and the challenges in using it.

3.1 Software Architecture Models

This section will cover the software architecture models to understand the Middleware and processes behind it fully. The knowledge of the architecture paradigms profoundly impacts understanding the later parts of the thesis. This section will cover the monolithic architecture, the microservices, and the distributed monolith model.

The correct selection of the architecture model for the Middleware impacts its features and scalability of the system. Based on the chosen architecture the development of the 5G-ERA Middleware can be either accelerated by the simplified architecture or allow greater scalability by increasing the development time and complexity of the system.

3.1.1 Monolith

Monolithic architecture is a traditional way of developing software followed by most large companies and implemented by most older systems (Lauretis, 2019). Monolithic applications are single standalone programs that combine all the functionality required from the software into a single unit deployed as a standalone application. These applications combine the User Interface with the Business Logic and the Data Access Layer into a single solution, as shown in Figure 1. Figure 1 presents an example of the Monolithic architecture, on which we can see a single application built from the User Interface, Business Logic and Data Interface connected to the backing database.

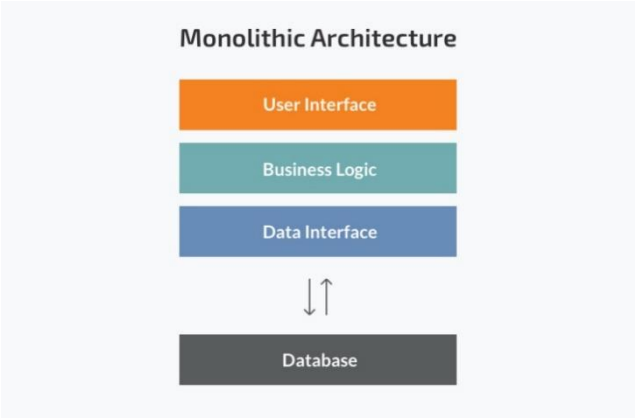

Figure 1 – Example of Monolithic architecture

Using the monolithic architecture for software development has advantages when the software is reasonably sized. Due to the high coupling between the services, they are easy to develop and debug and with the simple structure, they are easy to deploy in the desired environment. Furthermore, the simple scaling of the application can be achieved by running multiple copies of the same program and exposing it behind a Load Balancer if it is the web application.

Unfortunately, with the growth of monolithic applications, the problems also grow, and some advantages can become disadvantages. With the extensive monolithic application, the build time grows, and when it comes, it also increases the deployment time. The configuration is becoming hard to manage due to the external dependencies if the system has to grow. The monolithic architecture forces a long-term commitment to the specified technology stack due to the high coupling between the program elements (Richardson, 2018). Another disadvantage of the monolithic architecture that comes from a high software coupling is its reliability. With a failure in a single component of the system, the loss can affect the entire system (Harris, 2022).

Another issue is that the highly-coupled applications are usually not running on the web but are enterprise-grade desktop applications that provide a business-specific functionality. Due to an increased amount of configurations that can be additionally increased by the need to connect to external services, these applications make their deployment hard to manage.

Considering the advantages of the smaller-scale monolithic applications, the monolithic applications are a good starting point for most of the software being created. They provide a rapid initial product development that can be easily extended over time. Unfortunately, the monolithic architecture has its threshold that can be handled. Over this threshold, monolithic applications are becoming harder to develop, scale and maintain. When this threshold is reached, it is worth considering rebuilding the application to have a different architecture. The microservices architecture is an alternative pattern that has been created to address the limitations of the monolithic architecture.

3.1.2 Microservices

The microservices are an evolution of monolithic architecture modelling. The key specification of the microservices is that the system is broken into small, independent services. In the

microservices architecture, the principle is to deploy system components into different machines or inside separate containers. By placing multiple small services in the cloud-based environment, the microservices can provide the same functionality as the classic monolithic applications. An example of microservices architecture is showcased in Figure 2. The example showcases the example webshop based on the Microservice architecture. Contrary to the monolithic architecture, here each functionality of the shop is divided into separate applications. In Figure 2, we can see four services: Account, Inventory, and Shipping services. In the front of the system, there is an API Gateway and the Web application for the interaction with the client devices. Incorporation of such architecture allows independent scaling of specific components of the system to meet the current demands. Usually, in the microservices architecture, the maintenance of the system is divided between multiple development teams.

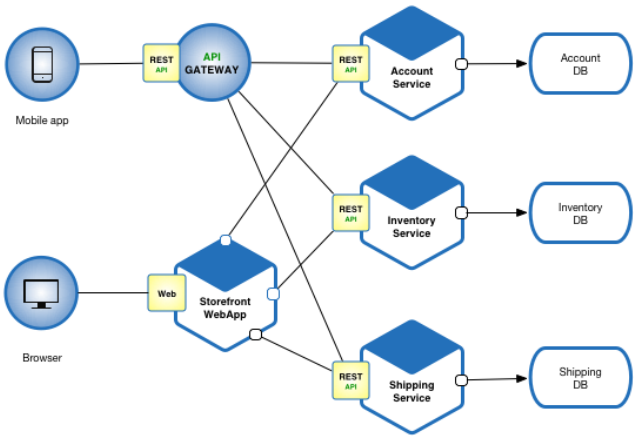

Figure 2 - Example of Microservice architecture

The main benefits of a microservices architecture are its scalability, deployment ease, and fault isolation. Thanks to their idea, the services are small, bringing them many benefits ranging from quicker deployment through easier testability to a better understanding of the services by developers (Richardson, 2018). The faster deployment time results from isolating the services by running them in a separate VM or a container. The microservice size allows for easy and quick tests in the Continuous Integration and Continuous Delivery (CI/CD) pipelines, which shortens the time needed to deploy the application. The small size of the services also results in easier maintainability. As the application is smaller than the monolith, it has fewer functions to fulfil and is more decoupled from the rest of the system, making it easier to maintain. This isolation also gives back to the fault tolerance of the whole system. Opposite to the monolith architecture, in microservices, when a single service is down, it does not fail the entire system. On top of this is the main benefit of microservices: scalability. Since the system is distributed, the essential parts that require more overhead or experience the increased load at times can be scaled independently to ensure the correct functioning of the whole system. With this approach, the system can handle a high traffic volume and respond to the sparks of usage.

The microservices architecture also comes with its drawbacks. The microservices architecture is much more complex to maintain on a small scale than a monolith. Since the services are spread, the deployment for each of them must be organised and managed. Not only did the deployment become more complex. With the distributed system comes a set of complexity.

When creating the distributed system, the development team must face several issues. To achieve fault tolerance, one of the key benefits of the microservice architecture, the rest of the services must be able to handle the problems with the specific service being unavailable. Another drawback of the distributed system is that a single service call can span multiple other services, so a single call creates much more significant network overhead and latency. On top of this, the cost of maintaining the infrastructure is also increased because of an increased number of services. The isolation of the services requires them to run in separate machines or containers to allow for the desired fault management.

With the microservices comes a set of benefits over standard monolithic software architecture as well as a set of drawbacks. There is no specific guidance on when the microservice architecture is better than the monolith. It depends on a few decisions and considerations. The decision to use the microservices architecture should be based on the knowledge of the application dynamics and the domain knowledge. Another question is whether the development team can be divided into multiple parts and whether the company can afford the expense of running the microservices architecture. As Martin Fowler said, “don't even consider microservices unless you have a system that's too complex to manage as a monolith” (Fowler, 2015). On top of this, it is hard to keep the microservices architecture pure. Introducing the high coupling between the services and the data is one of the ways to create a distributed monolith.

3.2 Distributed Monolith

The distributed monolith is a set of physically distributed services that are intertwined (Etheredge, 2018). It provides the middle ground between the monolith and the microservices. In monolithic architecture, the system components are interwoven, and in microservices, they are separate components. The distributed monolith presents the transition stage that is interwoven components in the separate logically and (or) physically separated. This type of architecture characterises itself by having its core components divided into different services but still relying on each other to enable the whole system to function correctly. Usually, the distributed monolith is referred to as a product of an unsuccessful move from a monolithic architecture to microservices. It is viewed as an anti-pattern created as a side product of the microservice architecture.

As a mix of the monolith and the microservices, the distributed monolith shares its disadvantages and advantages. One of the critical benefits of distributed monolith architecture is the coupling between services that allows for more rapid development by the team, which is also increased by the single repository in which the application is usually placed. On the other hand, the disadvantage of the high coupling would be the need to deploy the whole system at once, and as in the case of the monolithic architecture, the failure of the single component could mean the loss of the entire system. Fortunately, the resilience of the distributed monolith is improved mainly by its modularity. As with the microservices, the modular monolith has its services separated and running in different VMs or containers. This allows for the more effortless scalability of the separate modules if the specific service is under an increased load. Another benefit of this is the somewhat increased reliability. Having two services running instead of one provides redundancy, so if one of the services has a failure,

the second service can take over its tasks. Moreover, when these services are deployed as containers, the start-up times are usually lower than the start of the monolith application.

The knowledge of the Software Architecture models for cloud computing has profound implications for how the Middleware will utilise this. With this expertise, the correct decision can be applied to the development process of 5G-ERA Middleware. In this case, the Distributed Monolith has been chosen as an architecture model for the system to allow rapid development and market delivery. On top of this, it enables the Middleware to be scalable and resilient enough to justify the exemption from the full microservices architecture.

3.3 Cloud computing service models

Cloud computing provides different models for hosting services. The main three models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Mell & Grance, 2011). These models provide different services but are not directly coupled with each other. For example, the SaaS can provide functionality in the cloud without needing to provide the infrastructure. The models can coexist, but offering one is not equivalent to giving every model with it. This thesis will focus on two of these models, IaaS and PaaS to give the broader context on the desired Middleware functionality.

3.3.1 Infrastructure as a Service

Infrastructure as a Service is a cloud computing service that offers essential computing, storage, and networking resources on a pay-as-you-go basis (Microsoft, 2022). It provides online resources in public, hybrid and private cloud environments. The IaaS is not only limited to providing these solutions but also covers firewalls, load balancers, IP addresses and many more. In addition, the IaaS provides its client with the high-level APIs used to manage the low-level network, compute resources and more. In this model, the compute resources are run through a hypervisor as a guest. In this model, the end-user does not manage or control the cloud infrastructure itself but has control over a limited set of operating systems, storage, deployed applications and selected networking components (Mell & Grance, 2011).

3.3.2 Platform as a Service

Platform as a Service is a service that allows customers to use the application bundles without the need to provision and maintain the infrastructure associated with launching the application (Butler, 2013). The PaaS model aims to deliver to the customers the whole software package needed to provide applications ranging from simple cloud-based applications to sophisticated enterprise applications. An additional component of PaaS over IaaS is the runtime and the middleware that a PaaS provider manages when PaaS is used in a public cloud or an IT department when used in a private cloud.

The PaaS model aims to provide developers with an environment that allows them to quickly deliver the applications to the market and save time managing infrastructures like servers or databases. The PaaS leaves the management of the data and the applications themselves up to the end user. The PaaS was selected as the desired cloud-computing service type to enable smooth integration between the infrastructure and the end-user applications.

The 5G-ERA Middleware is supposed to present a bridge between all three cloud computing service models, SaaS, PaaS and IaaS. By cooperating with the IaaS managed by the OSM, with the NetApps working as a SaaS solution, the Middleware becomes the PaaS serving the optimisation for the robotic use case and bridging the resource provision with the dedicated orchestration process.

3.4 Existing solutions

On the market, there are a few existing solutions that support the idea behind the Middleware to enable and orchestrate cloud-based resources. This document section describes the core idea behind these solutions and their primary use cases. In addition, this section will cover orchestration technologies such as Kubernetes, OSM and Infrastructure as a Code (IaC), as well as describe the NetApps in more detail. On top of this, the Robot Operating System (ROS) will also be covered, and general information about the 5G network. Finally, the chapter will conclude with the benefits of each technology and why they are useful for Middleware development.

3.4.1 Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating containerised applications' deployment, scaling, and management. Kubernetes provides the means to support container-based deployment within PaaS clouds, focusing specifically on cluster-based systems (Victor Medel, 2016).

Kubernetes orchestrates the containerised cloud-native applications and supports their deployment, scaling and lifecycle management. It ensures no downtime on deployed applications by recreating them when one of the containers fails and is also able to kill the container that does not meet the user-defined health check procedure. Kubernetes is used for service discovery and load balancing. It manages the internet traffic to distribute it between deployed containers using the DNS names or IP addresses to ensure the stable deployment of the services. In addition, Kubernetes allows automatically performing updates on the existing applications in a controlled manner. Kubernetes can automatically handle the deployment, removal and adoption of existing resources (The Linux Foundation, 2022). The solution realises most of the demands the microservices software architecture places. It manages the scalability and recovery of the services to ensure the highest possible availability.

According to the annual StackOverflow Survey, Kubernetes, alongside Docker, is the most popular and most wanted orchestration software on the market (Stack Exchange, 2022).

While Kubernetes is a very flexible and powerful tool for cloud-native resource orchestration, it cannot fully support the problem of the lack of domain-specific knowledge during the deployment. As a result, creating deployments and deploying the services must be done manually. Additionally, Kubernetes lacks the means to inform the vertical applications about the deployed and ready-to-use cloud-based resources. Still, Kubernetes is a perfect platform to be extended by the additional functionality thanks to the multiple open-source extensions and API clients. Due to the provided extensibility and wide use, Kubernetes allows for the seamless deployment of the services in a transparent manner and a plug-and-play experience for the applications designed to work in the K8s cluster.

The Middleware will be built upon Kubernetes as a platform that will be extended with the functionality to create a domain-specific knowledge base used during the planning and lifecycle management of the deployed services for the vertical applications. The orchestration functionality provided by Kubernetes allows Middleware to intelligently schedule the deployment of the correct Network Applications based on the selected use case. The mechanisms provided by Kubernetes will allow the Middleware to efficiently scale the existing applications as well as manage their lifecycle to deliver the best possible coordination and management of the third-party applications.

3.4.2 OSM

Open Source MANO (OSM) is an open-source platform that allows the implementation of the management and orchestration (MANO) component from the Network Function Virtualization (NFV) architectural framework (Manuela-Ioana Csoma, 2020). It manages Virtual Network Functions (VNF) and deploys applications on demand. In addition, it supports the onboarding process to support the day-0, day-1 and day-2 configurations.

OSM enables the deployment of network solutions to the desired private cloud placements. However, while it can provide the infrastructure that Kubernetes will run on, OSM lacks the methods to monitor the state of the deployed applications and the support for the recovery or any lifecycle management, one of the essential principles of the Middleware. Through the SDN operations enabled by the OSM, the Middleware will be able to realise the provision of high-throughput networks to enable business operations and real-time communication services.

The communication between the Middleware and the OSM as a manager of the IaaS paradigm will allow for fulfilling the highest Quality of Service (QoS) for the vertical applications. Furthermore, the possibility to manage the resources using the API exposed by the OSM will enable the Middleware to provide on-demand resources and network connectivity for the robots to ensure the correct execution of the desired objectives.

3.4.3 Infrastructure as Code

Infrastructure as Code (IaC) is the process of managing and provisioning cloud-based services through machine-readable definition files rather than hardware configuration. The IT infrastructure managed by the tool can manage the virtual machines, Kubernetes clusters, as well as bare-metal servers. Furthermore, the tool has been created to be compatible with version control systems like Git to monitor version changes in the infrastructure in a clear and organised manner (Michael Wittig, 2015).

The advantages of the IaC include the simplified instantiation and linking of the required machines for the software to run. Both physical and virtual machines. IaC allows the installation and configuration of the software on the instantiated machines and provides the Middleware between the devices and the configuration (Artac, et al., 2017). In addition, it enables the automation of resource provision, which is one of the critical principles of DevOps (Senapathi, et al., 2018).

While the IaC enables the automated resource provision and the optimisation of cloud-based resources, it is created and designed for the static environment. It is not suitable for the fast-paced and changing environment. The use cases and the software needed in such an environment can change dramatically. The IaC will not suit the needs of the problem even if it provides the means of provisioning the infrastructure as well as applications. The constant need for change would require constant manual changes in the infrastructure and application configuration, which cannot be automated.

Even if the IaC is a tech-leading solution in terms of resource orchestration in the cloud, its static nature prevents the Middleware from using it during its workflow. In addition, the automated nature of the Middleware requires the possibility of dynamic resource allocation, deletion and optimisation that is not supported by the IaC solutions. Contrary, the IaC will provide an excellent deployment method for the Middleware that will work without OSM management. It allows the provision across multiple Cloud providers and end-to-end configuration of the Kubernetes clusters and supporting services.

3.4.4 ROS

The Robot Operating System (ROS) is an open-source software development kit for robotics applications allowing developers to carry out the whole process of robot software development from research and prototyping to deployment and production (Open Robotics, 2021).

Since version 2, ROS has supported real-time and embedded systems. The decentralised architecture of ROS enables good communication between the robotic platforms and the edge computers in the local network. Since the beginning, ROS communication has been based on the OSI Layer 2 (Data link layer). This type of communication is designed for the node-to-node data transfer between two directly connected nodes. Communication with the computers in the local network is beneficial for computational-heavy operations. Unfortunately, due to the technical implementation of such a communication, it cannot happen if the devices are placed in different subnets. Multicast communication disables the possibility of efficiently using cloud-based resources. Due to the high traffic the robot generates, ROS is not a perfect solution to be used in the cloud environment. While data transfers are not an issue when operating locally, when using cloud-based solutions, the amount of data generated by the robot can be problematic. NetApps are a way to enable the robot access to remote resources.



Figure 3 - Robotnik Summit-XL robot

The ROS is a vast ecosystem that reaches many aspects of robotic applications and is very deep-baked into the process of creating robots. Figure 3 presents the popular robot Summit XL by Robotnik that is running ROS. Therefore, even if the ROS is not an optimal solution for cloud-native communication, its importance in creating vertical robotic applications is so high that it cannot be omitted. Due to this, the Middleware will have to provide a way to communicate the cloud applications with the Robot Operating System. As this can be a complicated topic, multiple solutions may have to be utilised to achieve the desired outcome.

3.4.5 NetApps

As mentioned in the introduction, NetApp is a defined set of services that provide specific functionalities to the verticals, like autonomous robots and their associated use cases. The NetApps also have some particular characteristics that describe them in more detail. NetApp is supposed to deliver its services to the vertical using the 5G network. It can interact with other 5G systems as a standalone service or a part of one.

NetApps can serve the role of any system that is working over the 5G network. For the autonomous robots, these services could be, for example, the object recognition system allowing the robots to recognise the objects in the surveillance area or medicine for the medicine use-case. On the other hand, the NetApp could provide simultaneous localisation and mapping (SLAM) functionality to the robot without the need to perform a compute-heavy task while operating.

NetApps can be divided into two types: stateful and stateless. The stateful NetApp requires the persistence of its data or some already existing data to be modified. Through the operation of this data, the current NetApp status is saved to be later changed by other NetApps or vertical applications. An example of a NetApp designed for autonomous robots would be a NetApp performing SLAM given the output from the robot's camera. The opposite is the stateless NetApp. The stateless NetApp, by its definition, does not need to save its state. It

operates without the need to access external data and directly on the data provided by the robot. It also can work on the data provided by other stateful NetApp that is persisted in a place like an object store or database without modifying it so the NetApp can function correctly.

The NetApps are a crucial component serving the SaaS functionality of autonomous robots. The NetApps can play any role in executing the desired scenario by the robot. Ranging from object recognition to SLAM, the NetApps can provide various services to fulfil the vertical application requirements when programmed in this way. As the Middleware is the NetApp, it realises the paradigm of delivering the network functionality to the verticals. The only difference between the Middleware and the classical SaaS NetApp is that Middleware operates on the PaaS level, providing a higher range of functionality.

3.5 5G Network

The 5G is the 5th generation standard of cellular networks and a successor to the 4th generation (4G). Like its predecessor, the 5G is responsible for covering the selected part of the geographical area and providing the telephone network and internet through radio waves. The new generation of cellular networks provides an increased speed of up to 10 Gbit/s (Edwards, 2022).

On top of the increased speed, the 5G comes with many other benefits. 5G uses wavelengths ranging between 600–900 MHz on its low band and up to 24–47GHz on its high band, although the most common is the middle band that ranges between 1.7–4.7 GHz (Horwitz, 2019). The increased frequency of the waves provides a service with an increased number of available bands. Their increased number makes the radio waves less likely to interfere with each other, thus allowing more clients to be connected to this type of network. On top of this, 5G improves the latency to let its numbers be under 30ms in most cases (Verizon, 2020).

One of the main application areas of 5G is the Massive Machine-Type communication (mMTC) that is used as a benefit of the increased bandwidth of 5G. It allows connection for many devices like IoT, drones, autonomous vehicles or robots. In addition, another application of 5G, Ultra-Reliable Low-Latency Communications (URLLC), is supposed to provide uninterrupted, mission-critical robust network connectivity. These benefits of the data transmission using 5G are supposed to aid disaster recovery efforts and offer real-time data to the emergency responders by utilising, for example, autonomous robots in mission-critical scenarios. Not only could autonomous robots assist in disaster recovery, but they also could play a critical role in area surveillance.

Using a 5G network also poses a challenge. The biggest one of them is availability. The 5G came to the market in 2019 and is expected to account for 25% of worldwide mobile technology in 2025 (Forest Interactive, 2020). As the technology is relatively fresh to the market, it suffers from the lack of fully developed infrastructure resulting in availability problems. Even though the 5G network grows constantly, it still covers a small enough area that the improvements coming from the fifth network generation will not be used in most cases. An additional threat is the increased bandwidth and number of devices connected to

this network, especially IoT devices. The estimations say that the number of these devices can reach from the current 8 billion devices to up to 21.5 billion by 2025 (Lueth, 2018). This number of devices could pose a substantial security threat by increasing the capacity of the DDoS attack, cryptojacking or other cyberattacks (Elgan, 2018).

The development of the 5G network has a profound impact on Middleware. Utilising the 5G resources enables the ROS-native applications to better communicate over the network with the cloud-based NetApps. The desired communication methods that Middleware uses to allow the ROS verticals to utilise the NetApp's demand increased network throughput and will benefit from the possibility of using the increased number of devices connected to the network.

3.6 Summary

This chapter has covered some technical explanations for state-of-the-art software solutions and technologies. First, it has covered the two most common software architecture paradigms, monolith and microservices and the differences between them. Later, cloud computing service models like IaaS and PaaS explained their roles in bringing the applications and remote computing power closer to the end user.

The later section has explained current state-of-the-art orchestration solutions, how they work and what functionalities they bring to cover how the 5G networks can help increase the number of devices for mission-critical use cases like disaster recovery. Finally, these technologies' benefits are combined into a single system: Middleware, an orchestration platform for the 5G-enabled NetApps. Table 1 summarizes the advantages and disadvantages of the technologies covered in this chapter.

Technology	Advantages	Disadvantages
Software Architecture Models		
Monolith	Simple architecture, Easy to develop	Hard to scale, Complexity increases with the amount of functionality
Microservice	Easy to Scale	Complex architecture, Hard to develop correctly
Distributed Monolith	Good enough scalability	Limited scalability compared to microservices, Harder development than in monolith
Orchestration technologies		
Kubernetes	Infinite scalability, Widely adopted, Easy to integrate with many APIs	Offers only application-level orchestration
OSM	Allows for resource-level orchestration, Kubernetes and VM deployment	Very high learning curve, Limited on the NetApp deployment options

IaC	Perfect for static cloud infrastructure and application deployment	Lacks ways to dynamically create deployment configurations
Networks		
5G	Wider network range, Can connect more devices, Lower latency, Higher throughput	Low commercial network coverage

Table 1 - Advantages and disadvantages of selected technologies

4 Middleware Design

The 5G-ERA Middleware is the NetApp that is supposed to close the gap between the digital infrastructure and the robotic vertical applications. By enabling the robot to utilise cloud-based resources through local Edges and remote Clouds, the Middleware increases the availability of the robot's resources, services and flexibility by providing additional services to extend the robotic platform. Middleware as a NetApp plays a crucial role in optimising resource usage by serving as a management layer between the vertical applications and the remote infrastructure provision service and using the complex orchestration procedures to enable the deployment and operation of the robot at a large scale. With the Middleware using the semantic knowledge delivered by the end-user, it can create the most optimal plan for the robot to enable the flawless execution of the desired task.

This thesis chapter will cover the detailed functionality of the Middleware and will explain the idea behind these functionalities and how they help the robot to operate at scale. It will cover the functionality ranging from resource orchestration, semantic planning of the tasks using the knowledge base, and re-planning capabilities that ensure the robot's highest possible quality of experience.

On top of this, the Middleware has specific features like being the intermediary between the robotic platform, the cloud-based resource level network, and the infrastructure level network. This chapter will explain how this communication is handled and how it can improve the communication between the system elements through the Middleware NetApp and therefore assure the designated networking conditions for the variety of vertical applications.

4.1 General overview

The 5G-ERA Middleware aims to provide the platform for the vertical applications that connect them with the digital infrastructure in a service-platform-oriented approach. It increases the infrastructure's usability and availability by integrating it with vertical applications and connecting virtual resources to the service provision process. The Middleware is supposed to provide the virtual services across the multiple sites managed by the OSM in the IaaS model. As a PaaS covering several locations, the Middleware aims to provide the synchronisation methods to enable utilising a robotic fleet at a large scale by providing connectivity and coordination between multiple cloud solution models. Middleware also provides the means for end-to-end communication between vertical applications and cloud-based robotic services.

To achieve these goals, 5G-ERA Middleware realises multi-level networking by connecting the multiple applications seemingly designed to be unable to contact each other to fulfil its role as an intermediary. Middleware carefully chooses the means and technologies required to enable efficient data interchange for resource enablement and optimisation in the cloud-based environment to allow this level of communication. As showcased in Figure 4, Middleware connects the applications on three different network levels.

As presented in Figure 4 network layers are responsible for connecting seemingly different hardware and application functionalities into the unified system. The physical network layer creates two nested virtual network layers. The Resource Layer hosts the Kubernetes, and the

Vertical Layer Network is responsible for the application-level traffic. The proposed Middleware aims to bridge these functionality layers together to provide a unified system that will handle the administration of the physical network under which works the virtualized Kubernetes hosting the actual applications that the end user will use.

The first is the Vertical Layer Network, utilised mainly by ROS applications. This network layer is responsible for communication within the robot between the robot controller, sensors and actuators. On this level also happens the synchronisation between multiple robots running under the same subnet and their services, realising functionality like SLAM or object recognition. This part of the network is essentially the Software as a Service (SaaS) cloud service type model. It is being run on demand when requested by the robot vertical.

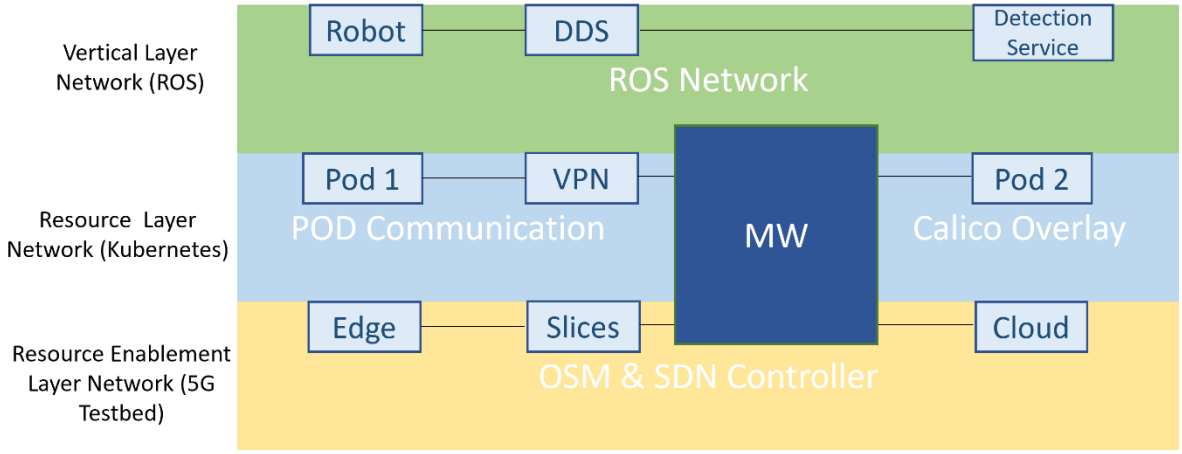

Figure 4 - Middleware across networks

The second is the Resource Layer Network. This network is responsible for hosting the Cloud or Edge-based NetApps. This layer manages the Kubernetes network using the Calico overlay to enable efficient communication between the services in this network and access from outside the network. To access the services from the outside, Middleware supports two ways of granting access to the services. The first one is the standard way of exposing the services in Kubernetes: the Load Balancing or Node Ports. The port exposure is enabled to the NetApps designed to be used by publicly available APIs through the REST or RPC calls over OSI Layer 7 (Application layer) based on HTTP protocol. This communication is native to Kubernetes and easiest to implement from the Middleware perspective. Unfortunately, not every NetApp can be implemented this way from the ground up.

For this reason, the Middleware also supports access into the K8s cluster network by connecting with a VPN. The VPN connection allows simulation of the robot's behaviour and the pods running in the cluster, as they would be placed in the same local network. This allows efficient communication between the vertical applications using ROS and the remote resources designed to work with the ROS-native solutions. This part of the network is the closest to the PaaS cloud service model. As its base, it provides the infrastructure and the management platform to the vertical applications working in the ROS Vertical Layer Network.

The third and last is the Resource Enablement Layer Network. This network is essentially the 5G testbed responsible for enabling connectivity and public access to the platform via the 5G

connection. The coordination of this network layer is conducted by interchanging information between Middleware and OSM that is used as an SDN controller. The communication process is executed using the REST API exposed by the OSM. It allows for monitoring and managing the state of the connected infrastructure. The OSM provides the creation of network slices that will enable the connectivity needed. This network layer through the OSM has to allow the efficient connection between the autonomous robots, local Edge machine and the Cloud. The Middleware is supposed to coordinate with the OSM to create the connectivity method between the vertical and the most effective computing platform, either Edge or Cloud. The process of selecting the best placement for the NetApp deployed by the Middleware to be used by the vertical application is based on the knowledge graph that allows to strategically determine the best parameters to be used for the NetApp to ensure the highest possible QoS and resource optimisation. This network layer in the Middleware, as PaaS, plays the role of infrastructure, by de facto being the IaaS cloud service model. The Middleware utilises this network layer to build itself upon it and present the fully-fledged Platform as a Service to the end user.

The presented placement of the Middleware has a profound impact on its functionality, architecture and the way it is developed.

4.2 Middleware architecture

This section will present the overview of the 5G-ERA Middleware architecture and the decisions behind it. In addition, this section will explain the methods of communication between the services and the network layers presented in the General overview section. With this, the role of each component of the middleware will be explained, what role it plays and how it communicates with other modules. Finally, the following sections of the chapter will cover a detailed description of the functionality of the Middleware.

The Middleware as software can be placed in multiple localisations like public or private clouds or local Edges. Putting it in any environment running Kubernetes allows the Middleware to achieve transparency in the deployment and resource provision. The instances of the Middleware use the shared Redis cluster responsible for the data synchronisation. Each Middleware instance shares the mission-critical data that allows them to operate with each other. The constant synchronisation and communication will enable the Middleware instances to reach the increased planning options and the location switchover capabilities.

On top of the increased replication and planning capability, deploying the Middleware in multiple locations profoundly impacts the software's availability. Multiple Middleware instances act like availability zones. It provides vertical applications with a fallback site for resource deployment. By synchronising the data, one Middleware instance can quickly take over the work being done by a different Middleware instance in case of a failure that makes the whole location unavailable.

4.2.1 Architecture description

The Middleware was designed with flexibility and extensibility in mind. Middleware follows many microservice architecture principles like service isolation, scalability and fault tolerance. However, the 5G-ERA Middleware has not been created as a pure microservices architecture.

The monolithic approach to the system design allows for the easier development and testing of the services. In the end, the Middleware tries to connect the best of both architecture solutions and becomes the distributed monolith.

As a distributed monolith, Middleware allows for rapid development to ease the market delivery process while maintaining the advantages of the microservices architecture. Thanks to keeping the services separated by the domain responsibilities, the Middleware keeps the critical components separated from each other and enables the easier maintenance, development and deployment of each service. Furthermore, thanks to the distribution of the services, the Middleware ensures a good level of scalability for each component. Another benefit of the distributed monolith architecture that the Middleware will utilise is the fault tolerance coming from the distribution of the services. This fault tolerance is greatly enhanced by the Kubernetes that the Middleware is running in. Furthermore, the containerisation allows K8s to quickly restart the failed application components in case of unexpected failure and to scale the application to increase the number of replicas for the application component to increase the system's stability even more.

As shown in Figure 5, the Middleware itself consists of five services: Gateway, Task Planner, Resource Planner, Redis Interface and Orchestrator. These services are separate APIs created using .NET Core. .NET uses the C# programming language to create web applications. C# as a programming language offers the variety of features needed to develop enterprise-grade applications, including the web, that can run on any operating system. With its vast number of features and top-of-the-class support, it is one of the most popular programming languages for creating web applications (Stack Exchange, 2022).

Middleware decouples its functionality into 4 different services and exposes them with a single API Gateway. In the Middleware, the Task Planner is responsible for finding the most appropriate actions to execute and identifying their order. Resource Planner is responsible for identifying the resources that need to be deployed for each of the actions identified by the Task Planner and finding the best possible location for their deployment. Then Orchestrator deploys the applications themselves and provides an interaction point between deployed NetApps and the Middleware. Finally, Redis Interface provides the data access interface for other Middleware services and external users.

The Middleware services utilise the Redis cluster as a data store for the system, distributed in the cloud and can be utilised by multiple instances of the Middleware.

The connectivity points between the Vertical Layer Network, Resource Layer Network, Resource Enablement Layer Network and the Middleware are two services: Gateway and the Orchestrator. The Gateway plays the role of the entry point of the whole system, including communication with the Vertical Layer Network. On the other hand, the Orchestrator is responsible for communicating with the other two network layers: resource and resource enablement.

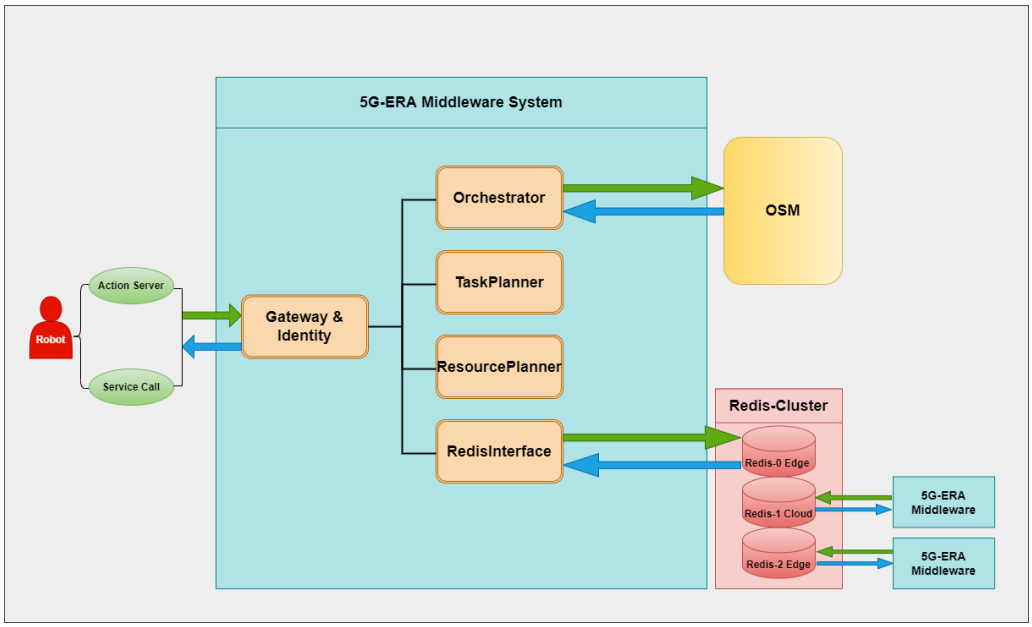

Figure 5 - Middleware architecture

All the services are responsible for a specific part of the communication with the autonomous vertical application. When the transmission starts, the Gateway handles the authorisation of the vertical applications and redirects the network traffic to the appropriate service. Usually, the first contact of the robot with the Middleware after it is registered with the system is to request the plan for the actions needed to execute the specific task. For this, the Gateway redirects the request to the Task Planner. The Task Planner is the first responder to the task request. Using the semantic knowledge dedicated to the specific robotic domain and composes the series of actions required by the robot to complete it. After identifying the key steps, the Task Planner passes the request to the Resource Planner. The Role of the Resource Planner is to identify the algorithms needed to complete each of the actions identified by the Task Planner and to specify the placement of the instances that will have to be created to ensure the highest possible QoS for the robot. When the resources needed are identified, the Resource Planner returns the ready plan to the Task Planner. If the resource plan is completed, it is passed to the Orchestrator to deploy the services needed. Based on the information from the Resource Planner, the Orchestrator coordinates the deployment process in the desired location. It also schedules the deployment of the specific network slices with the OSM to enable the full capabilities of the robot platform. All of the services mentioned above utilise the Redis Interface service that plays the role of the data access service to the knowledge graph persisted in the Redis cache database.

When the robots successfully request the task, it is time to ensure constant communication between the vertical applications and the Middleware to have the most up-to-date information about the deployed resources. For this, several solutions are utilised.

4.3 Communication between layers

The Middleware utilises several communication methods to bridge the gap between the application layer, resource layer and resource enablement layer. All layers realise the communication similarly, but some require more sophisticated solutions to be implemented.

4.3.1 Basic Communication

The primary method of communication is handled through the REST APIs prepared by the Middleware and supporting services. The communication between the Middleware and the network layer based purely on REST is the communication with the Resource Enablement Network.

For the Middleware to communicate with the Resource Enablement Network, there is a need to utilise the API exposed by the OSM. With the mentioned API, the Middleware can request to enable the SDN to provide access to the best resources available to the vertical application. Using this API, the Middleware has excellent control over the deployed resources. It can manage their deployment on demand and schedule removal if no longer needed.

4.3.2 Middleware and Resource Network Layer

The communication between the Middleware and the Resource Network Layer is slightly more complicated. On this network layer, the Middleware has to communicate with not a single component but multiple.

The primary communication protocol between the Middleware and the Resource Network Layer is the Kubernetes API. By calling the Kubernetes API, the Middleware can manage the state of all the resources and monitor them to ensure the highest possible availability. By contacting the K8s API, the Middleware schedules the deployment of the NetApps. The Middleware using this API and REST calls conducts the lifecycle management of the NetApps and enables them to self-report their state back to the Middleware.

After the Middleware deploys the NetApps, they are bound to the contract that focuses on ensuring the highest possible NetApp availability and resource optimisation. The Heartbeat protocol is based on REST and thus is easy to use by both NetApps and Middleware to provide the highest convenience of communication.

Middleware communicates with the Kubernetes REST API to achieve the following actions:

- List existing resources
- Monitor deployed resources,
- Deploy new resources,
- Adjust the configurations of the existing resources.

4.3.3 Middleware and Vertical Application Network Layer

At last, the Middleware connects to the Vertical Application Network layer for efficient communication between the Middleware and the robots. The communication between these two entities is handled by the ROS Action-Server, the middleman between the robot-related applications. For better scalability of these services, as opposed to the general assumption of

the ROS Action Client and Action Server, both services run inside the robot, and only the Action Server connects to the Middleware.

The ROS-native communication handles the communication between the ROS Action Client and the Action Server. The communication between the Action Client and the Action server is based on the concept of goals and topics. When the vertical application requests the Middleware for the resources needed to execute the specific task, the Action Client initiates the whole process. When the Action Client receives the request from the robot, it sends the goal request to the Action Server. Upon receiving the goal, the Action Server communicates with the Middleware through the REST API to establish the plan for the high-level task. When the Action Server receives the information from the Middleware, it informs the Action Client if the requested goal was accepted. If not, the whole pipeline is terminated, and the robot has to request a new goal. If the Middleware has accepted the task, the Action Server passes this information back to the Action Client and opens the feedback topic that updates the Action Client on the current state of the services deployed by the Middleware. The example communication process is showcased in Figure 6. Figure 6 consists of the two components, Action Client and Action Server and their communication through Actions to achieve a certain long-running goal. Action Client initializes an Action to be conducted by sending the Goal to the Action Server and receiving the response. Action Server starts the communication and the Feedback publisher provides live Feedback to the Action Client. When finished, the Action Client sends the Result information to the Action Server indicating that the Action has been completed.

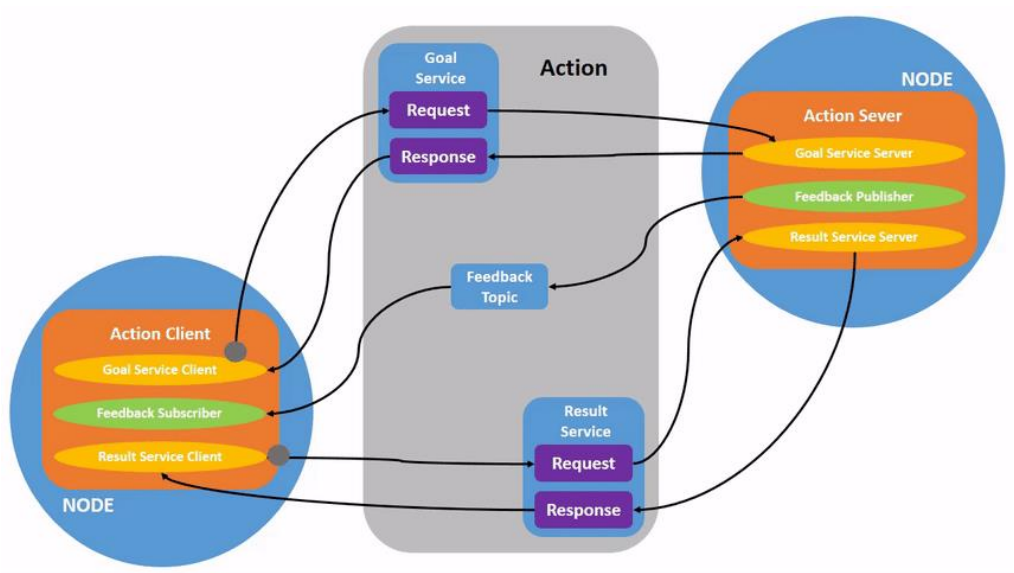

Figure 6 - ROS Action-Server communication schema

The high-level task requested by the robot is broken out into smaller pieces that can be managed individually by the robot. The list of the individual actions deployed by the Middleware contains information about the resources deployed in either Cloud or Edge machines. The robot is continuously updated in the specified time interval about the status of the resources deployed. With this information, the robot knows the path needed to execute the desired task and how many steps it takes to achieve it. When the robot finishes a specified

stage, it updates the Action Server that it has completed the execution of the specified task. When the Middleware receives the information from the robot, it can delete the resources used by the robot. The removal of resources is not conducted if these resources are needed in later stages of task execution or if the service is also utilised by another robot in the resource optimisation process.

The ROS Action-Server implementation differs from the implementation needed by the 5G-ERA Middleware. In native ROS, the Action Server provides information to the Action Client about the status of the task the robot is executing and informs the robot when the goal has been reached. For the needs of the communication between the Middleware and the vertical application, this order is inverted. In the communication between the robot and Middleware, only the robot can assess if the desired task has been achieved. The Middleware requires this approach, as the communication process between the robot and the NetApp omits the Middleware. Therefore, the communication between these components is direct. The Middleware neither knows about the action execution progress, nor the data exchanged between applications; only the robot can assess if the goal has been reached. The robot fulfils this protocol by issuing the goal update to the Action Client when the current action step is achieved. The update with a new goal has the same identifier as the initial goal. Using this identifier, Middleware recognises the need to update the existing goal and proceed to the following action in the sequence. After completing each element of the action sequence, the robot issues the new goal request with the reference value indicating that all steps have been finished. Therefore, the high-level task has been completed.

Additionally, Vertical Application Network sends the information to the Middleware with Heartbeat data. Heartbeat data is sent to the Middleware over REST API both from robots and from the NetApps. In each case, Heartbeat contains vital operational information. The robot sends the information about the current CPU, RAM and battery usage, while NetApp sends the information about the number of robots it currently works for and how many more it can handle.

4.3.3.1 NetApp and Robot Communication

When the vertical application requests the resources from the Middleware to conduct the specified task, the Middleware deploys the set of NetApps designed to work with the robot. Based on the robot's capabilities, the Middleware can deploy the NetApps working over the cloud-native communication protocols over the OSI Layer 7 (Application layer) like REST or RPC or can deploy the NetApps that communicate over the Layer 2 communication used by ROS. These NetApps communicate with other ROS applications by the set of topics that is a standardized communication protocol for robotic applications. Thanks to this approach, the Middleware does not require rebuilding existing vertical applications to work over the REST but instead keeps the backwards compatibility with the ROS-native applications.

4.3.4 Components description and functionality

As mentioned before, the Middleware consists of five components: Gateway, Redis Interface, Task Planner, Resource Planner and Orchestrator. This section of the chapter will cover each

component, its role in the overall system view and how these components address problems defined at the beginning of the document. The issues addressed are dedicated orchestration for third-party vertical applications, 5G enablement and resource optimisation in cloud-based environments.

4.3.5 Gateway

As the Middleware implements the Distributed Monolith architecture, the core services are physically distributed, each with a different IP address. Having many services running presents the problem of connecting directly to multiple microservices for the client. With each service playing a different role in the system, the client application must keep track of multiple service addresses and their ports. The Gateway pattern tries to solve this problem. It provides a single entry point for the groups of microservices as a part of a distributed system. API Gateway is placed between the microservices and the client applications, acting as the reverse proxy routing the requests to the appropriate microservices (Torre, et al., 2022).

Middleware uses the Ocelot API Gateway to aggregate the requests into a single domain and redistribute them between the appropriate services based on the route used. The Gateway also handles the authentication of the users and robots, restricting access to the specific parts of the Middleware APIs.

4.3.6 Redis Interface

Redis Interface is the backbone of the Middleware system as it provides access to real-time data about the robots' utilisation throughout the system and the knowledge graph needed for effectively planning the robot tasks and resource orchestration.

As its name suggests, Redis Interface uses Redis to store mission-critical data. Redis stores data ranging from the knowledge graph for the deployment and task competition to monitoring the robot's status via the heartbeat protocol. Primarily, Redis can be used as the database, cache, message broker and streaming engine (Redis Ltd, 2022). In addition, it can store other data formats like JSON, graphs, or time series when extended. With the JSON and graph extensions, Redis has all the functionality the Middleware needs. The JSON extension provides a way to store complex object data and provide data that is structured similarly to the RDBMS or the document database. The graph database plays a crucial role in the system as it allows associating services required to execute the task requested by the vertical application. The graph database integrated into Redis provides seamless and rapid data access that vastly improves the whole system's performance.

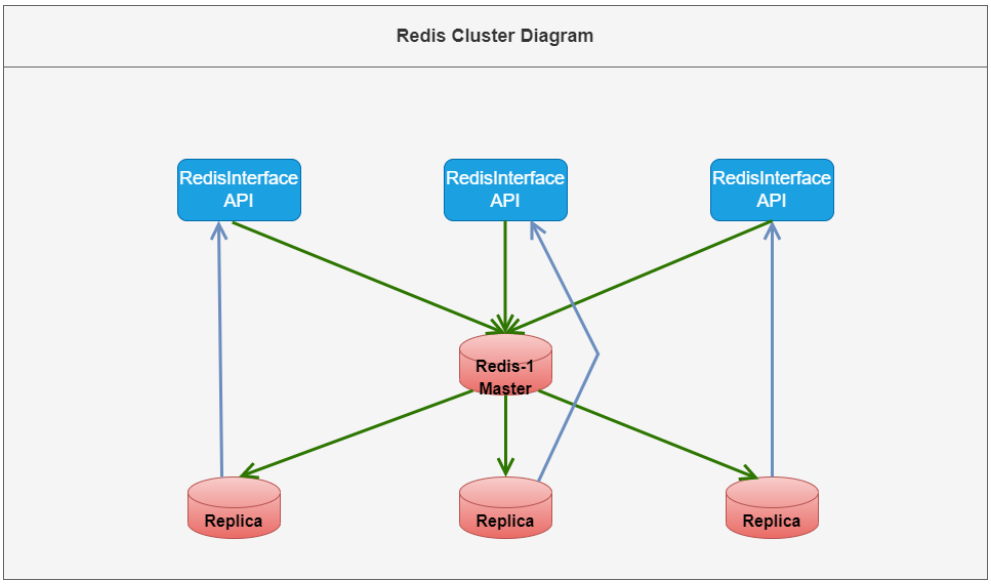

Figure 7 - Redis Cluster schema

On top of this, Redis allows clustering to provide even better performance by allowing the cluster to scale up to 100 nodes and more and ensure the highest possible availability (Redis Ltd, 2022). The Redis Interface utilises these principles behind the cluster to provide even quicker data access and integrity. The Middleware uses the Redis cluster as its data store with the replica of the cache placed in each Middleware location to enable instant access to the data without the added latency. This data is used by the Task Planner, Resource Planner and Orchestrator components throughout the work of the Middleware.

Figure 7 presents the Middleware communicates with the Redis cluster using multiple replicas for the reads and a single master replica used for writes. Each Middleware instance has the Redis replica in its localisation to improve the latency on reads. As the Middleware is a read-heavy system, the single-leader replication is sufficient for the current Middleware iteration. However, if the number of writes done by the Middleware increases, the Redis Cluster will be converted to support multi-leader replication.

4.3.7 Task Planner

The task planner is the entry point for the autonomous robot when one requests the plan and resource allocation from the Middleware to perform the specific task. The Task Planner is responsible for assessing the feasibility of the requested task and providing the sequence of the actions that will have to be conducted during the successful execution of the Task. The Task Planner also has the functionality to re-plan the already created action sequence in case of a resource failure or a robot failing to achieve the task for other reasons.

4.3.7.1 Semantic planning

The semantic planning of the task allows Middleware to utilise the knowledge of the specific domain to create the high-level task. By introducing the domain knowledge prepared by the experts in the field, the robot will be able to dynamically work on most of the tasks given without the need for human intervention in the planning process or control over the vertical application.

By supplying the current state of the robot, like localisation, battery status and other robot specifications, the semantic planner can assess which actions can be executed by the system, allowing for the highest performance and accuracy from the delivered services. Based on this, multiple object recognition algorithms can be used for the same task, but each can be adjusted for different scenarios. For example, there could be a distinction between algorithms best suited for object recognition in low-light environments or bright environments like the outdoors. Furthermore, the planning process considers the sequence of actions that must be followed. When some action requires input from another action, the semantic planner will take this into account and plan the steps so that each action follows the actions needed to be completed and whose finish was required to handle the desired action. The example dependency is shown in Figure 8**Error! Reference source not found.**. When the action that another action depends on has failed, the task competition cannot be achieved, and the re-plan will be scheduled.

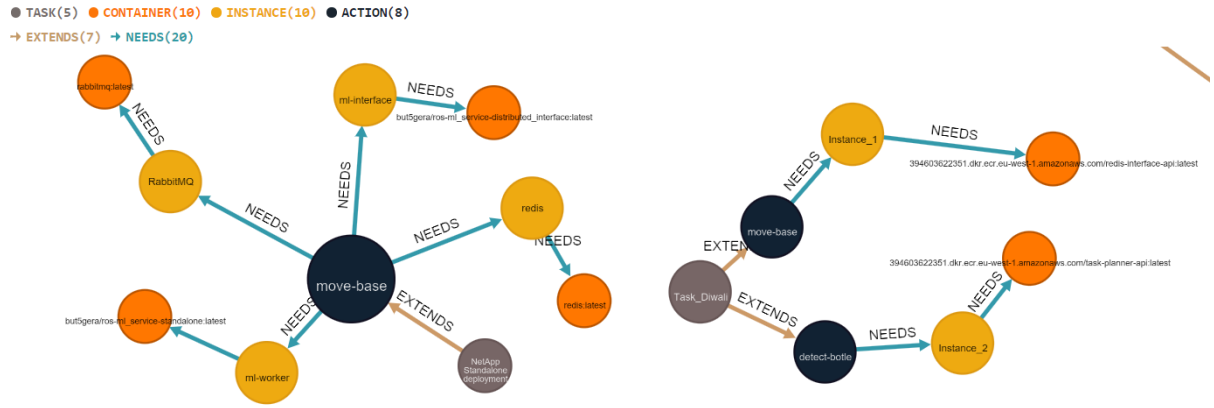

Figure 8 - Task Planner relations

4.3.7.2 Re-planning

The re-planning process is the diversion from the planning process, executed when some actions fail to be correctly completed. When the re-plan action is requested, it assesses what is needed to make the task available again for the vertical application. The re-planning process follows the same rules as the planning process.

The task re-plan can be called either by the robot or when the Orchestrator detects a failure in the application. The Task Planner allows for two forms of re-planning: partial and complete.

When the partial re-plan is scheduled, it usually means that some action in the sequence has failed its execution. For example, it can be the failure in the operation of the NetApp or the robot cannot achieve the desired goal with the resources provided. In this case, the Task Planner will search for the replacement for the specific failed action and possibly the actions following it, as it is the quickest way of allowing the robot to continue its task execution. If the task cannot be finished in the particular form, the full re-plan will be scheduled by exchanging the services.

During the full re-plan, the Task Planner tries to find an alternative way of achieving the desired task using a different set of resources from the ground up. The process is the same as the planning process. It will try to construct the sequence of actions allowing the vertical

application to reach the specified goal. The difference in this process is that the re-plan will not use the services included in the original plan.

When the plan has been successfully created, it is time to associate the resources needed for the task competition and the plan's current state is sent to the Resource Planner.

4.3.8 Resource Planner

The resource planner plays a similar role to the Task Planner but operates on the resource level. First, the Resource Planner assesses the resources required by the task for its completion. Then, when the task plan is received from the Task Planner, the Resource Planner uses semantic planning to associate the resources required. These resources, NetApps, will have the placement associated by the Resource Planner to eliminate the assignments in unsuitable locations for NetApp.

4.3.8.1 Semantic planning of the resources

The resources needed to execute the specified action are identified during semantic planning. Then, in the same way as the Task Planner, the Resource Planner utilises the semantic knowledge graph created by the field experts to recognise the best possible resources for the robot. Furthermore, by using the knowledge given by the robot during the task organisation, the Resource Planner can assess what kind of resources suit the current robot's situation best. Finally, the correctly configured instance type will be assigned to the robot to help with the task execution.

The example of the knowledge graph utilised by the Resource Planner is showcased in Figure 9. It showcases the relations named "NEEDS" between the defined Instance, like "ml-interface" and the container that needs to be deployed like "but5gera/ros-ml_service-distributed_interface:latest". A presented connection is required to correctly identify the resources that must be deployed for the NetApp to function correctly.

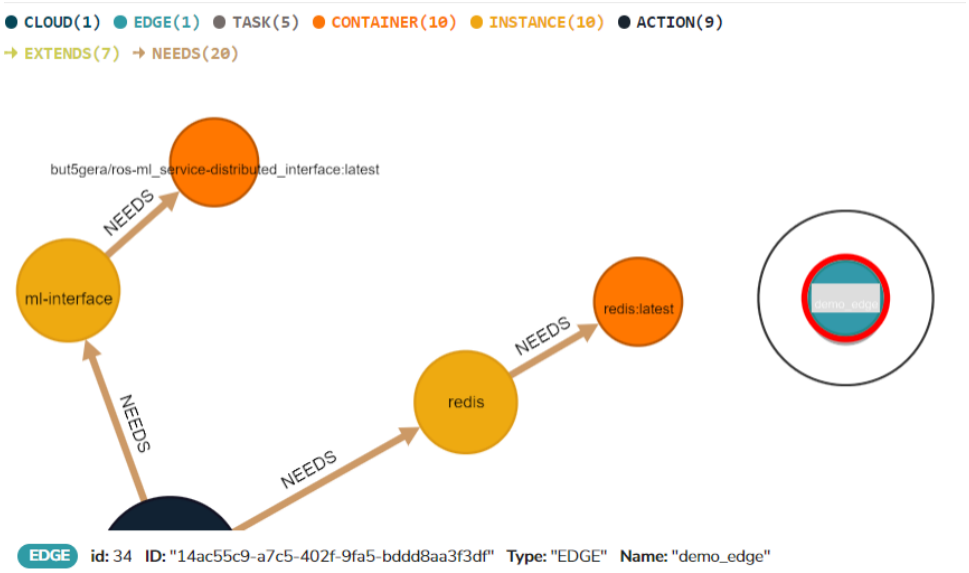

Figure 9 - Semantic resource planner graph

The Resource Planner checks the existing running tasks and their instances during resource identification. During this check, the Resource Planner queries for the information from the heartbeat protocol to establish the instances that could be reused to allow for better resource optimisation at scale.

When the desired NetApp is identified, the Resource Planner will try to identify the most suitable location. However, suppose there is no fitting location for the NetApp in the currently available platforms. In that case, the Resource Planner will locate a similar resource that can fit into available platforms like Edge or Cloud.

4.3.8.2 Re-planning

During the re-plan process, the Resource Planner identifies the action modified by the Task Planner to allocate new resources. Depending on the type of the re-plan, if it is a full re-plan or partial re-plan, the Resource Planner will request the specific number of the resources to be reorganised. When the new actions are identified, the Resource Planner, in the same manner as during the initial planning, uses the semantic knowledge to identify the most relevant resources.

With semantic resource planning, the NetApps are assigned to the robot in the given location that meets the hardware requirements. Unfortunately, meeting hardware specifications may not be enough to ensure that the NetApp will run correctly in the desired location. For example, with a few other NetApps already running in the same place, the Edge or Cloud VM could throttle and provide a decreased experience. For this and similar situations, machine learning resource prediction will be utilised.

4.3.8.3 Machine Learning Resource Prediction

The machine learning algorithms are in the planning phase, and their detailed specification is not available yet. The idea is that these algorithms will use reinforced learning to utilise the existing NetApp behaviour data and learn in real time. The advanced resource prediction will be crucial for the large-scale resource optimisation and prediction of how the specified NetApps behave so that they can be placed in the most suitable position.

In the end, when the Resource Planner has finished identifying the resources needed, it is time for their deployment. In this process, the Resource Planner responds to the Task Planner that the resources required have been identified. Afterwards, the complete plan is scheduled for the deployment that the Orchestrator will handle. Finally, the Task Planner can return the proposed plan to the robot without scheduling its instantiation when the robot has scheduled only the preview of the proposed plan.

4.3.9 Orchestrator

The Orchestrator is a centrepiece of the Middleware that allows for managing the resources in the Kubernetes cluster. The Orchestrator coordinates the deployment and manages the applications' lifecycle under the 5G-ERA Middleware. The Orchestrator responds to the requests from the Task Planner when the plan is created. When the Orchestrator receives the new plan to be instantiated, it starts the process of orchestrating all the resources needed.

4.3.9.1 Resource and Service Orchestration

The Orchestrator is responsible for the last step of the resource and service deployment process using Middleware. First, based on the plan prepared by the Task and Resource Planners, it schedules the deployment of the identified services for specified actions. Next, for the correct deployment, the Orchestrator identifies the deployment configurations for the resources identified by the Resource Planner. Finally, it connects to the Redis backing storage to access the appropriate configuration for each service to deploy later.

During the deployment, the configuration present in the Redis datastore is validated. The validation is based on the type of resource to be deployed. As the NetApps must be deployed in the Kubernetes cluster, the content is checked that it meets the schema requirements of the Kubernetes objects like services or deployments. The deployment will be deemed unsuccessful when the services do not have compliant configurations. However, with the correct configuration, the Orchestrator can proceed with the deployment procedure. The next step is to alter the deployment configuration to allow the NetApp to communicate with the Middleware through the Heartbeat protocol. The Heartbeat protocol will be described in more detail in the Heartbeat section of the chapter.

Figure 10 presents the communication order in which the request to deploy NetApp is processed. It starts with the robot requesting to instantiate the specific high-level task definition. It communicates through Action Client and Action Server to send correctly formatted requests to the Middleware over REST API. When Middleware receives the request, it processes it with multiple services. First, the Action Planner identifies the correct actions to be completed and their correct sequence. Second, the Resource Planner identifies the applications to be deployed and their placement within the list of available resources, like Clouds or Edges. Finally, the Orchestrator is responsible for the deployment of the applications themselves. When the NetApps are deployed, Orchestrator also receives the heartbeat information from the NetApps to collect the most up-to-date operational information.

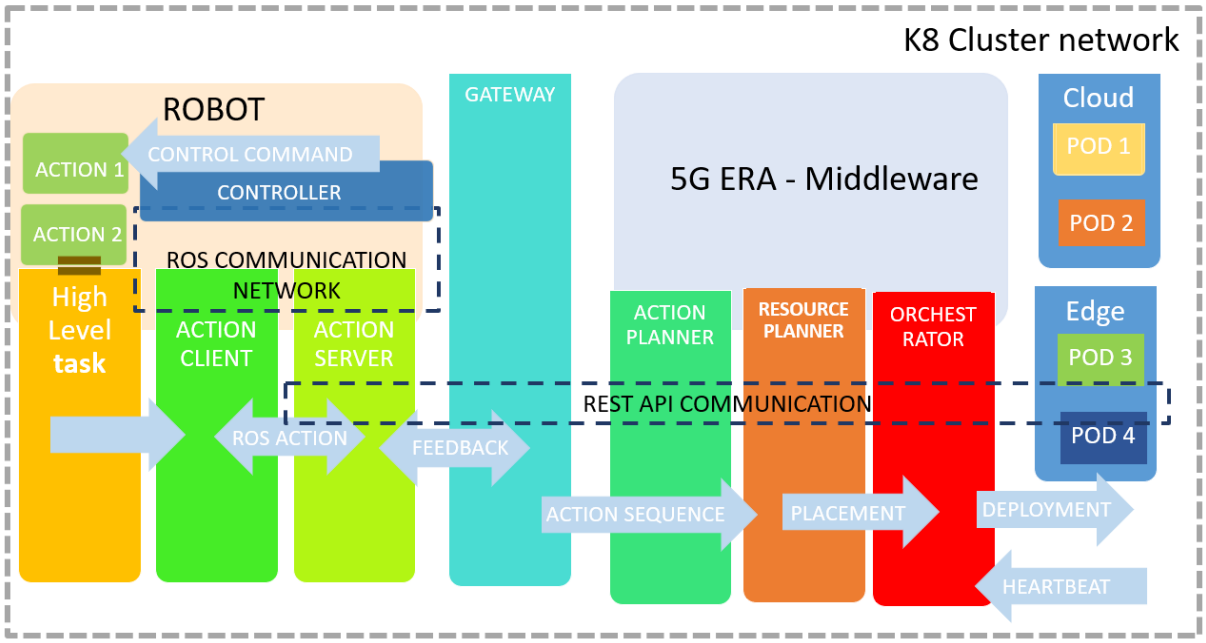

Figure 10 - Middleware end-to-end service creation and management

Depending on the state of the services to be deployed, the Orchestrator will have a different deployment procedure. As defined in the NetApps section of the Literature Review, NetApps can be stateless or stateful. Each of these types requires a different approach to deployment. In the case of the stateful NetApps, the data persistence layer must be provided on top of the standard deployment process. The stateful NetApp will operate the specific dataset that needs to be supplied and persisted after the service finishes its work. An example of the specified stateful NetApp could be the application performing the SLAM operations. During its operation, access to the persistent files has to be provided to read the existing map, complete the localisation, and expand the current map. To provide this data, Orchestrator assigns the Persistent Volumes to the pods to allow access to the data. Initially, the data attached to the Kubernetes pods can be located in cloud-designated storage like Amazon AWS EBS or local storage in Edge machines.

When all the variables needed for the deployment are associated, the Orchestrator can start the deployment process. For this, it contacts the Kubernetes APIs to deploy desired services. For example, suppose the Resource Planner has identified a service that can be reused. In that case, the Orchestrator will not deploy the specified service but will directly provide the details on connecting to the specified service. When the deployment process is realised, the Orchestrator reads the needed information to access the resources and gives them back to the Task Planner, which passes them further to the robot. Finally, with the desired services deployed, the Orchestrator moves to the process of monitoring and managing the lifecycle of the deployed services. The deployment of the services through Kubernetes enables the transparency of the resource deployment and the plug-and-play experience for the vertical application developers.

4.3.9.2 Lifecycle management

Lifecycle Management is one of the crucial responsibilities of the Orchestrator. It is responsible for monitoring the status of the NetApps deployed in the Kubernetes cluster by the Middleware. For the monitoring, Orchestrator periodically checks the quality of all services to ensure their availability and to apply the recovery procedures. After the Middleware receives the information that the robot has finished the work, the process of lifecycle management conducted by the Middleware schedules the removal of the resources no longer utilised by any robot. The lifecycle management is supported by communication between the NetApps and the Middleware through the NetApp protocol.

(a) Heartbeat

As the industry-standard mechanism, Heartbeat serves as the indication of the service's health. Using the heartbeat protocol is a standard procedure in mission-critical systems that provides the system's high availability and fault tolerance. The service is deemed failed if the master system does not receive the heartbeat information for a given time (Aguilera, et al., 1997).

The Heartbeat is the process in which the NetApp informs the Middleware about its status. The Heartbeat from NetApp serves two purposes. The main one is the assessment of the functionality of the NetApp, allowing it to detect when the NetApp does not function correctly. Another objective for Heartbeat is to help reuse the existing NetApps by the multiple robots.

The Heartbeat is a piece of information sent in regular periods and allows for the constant monitoring of the software. Due to the frequency of the updates, the eventual issues with the NetApp can be quickly detected. Rapid detection allows for a quick reaction to the problem and assessing the appropriate actions to take when issues with the NetApp are detected.

On the other hand, the heartbeat functionality of the Middleware and the NetApps can serve optimisation purposes. When the Middleware accepts the heartbeat information, it also receives the information on how many connections with the Robots the NetApp handles. With the prepared calculations, the NetApp should be able to assess the optimal and maximal number of Robots it can work for. With this information, the Middleware decides if the existing NetApp should be reused instead of deploying the new instance of the NetApp.

During the deployment, the 5G-ERA Middleware gives NetApp the required information on where to report the Heartbeat. This information is stored in the NetApp in the following environment variables:

- NETAPP_ID – Unique identifier in GUID format
- MIDDLEARE_ADDRESS – The URL to which NetApp must call the Middleware to report its status
- MIDDLEWARE_REPORT_INTERVAL – The interval in which the NetApp must report its status to the Middleware, expressed in seconds

The Middleware can identify each NetApp to monitor and manage the status with these settings. The Heartbeat is sent to the Middleware through designated REST API calls. This

allows for better resource planning, and identification of the status of the NetApp passed to the robot. The example Heartbeat received from NetApp is shown in Figure 11 below. It contains the following information:

- Id - unique identifier of the NetApp that has sent the request.
- Name – the name of the NetApp, in this case optional, as it is known from the Id.
- HardLimit – the maximum number of robots the NetApp can provide a service to.
- OptimalLimit – the optimal number of robots the NetApp can confidently provide service for
- CurrentRobotsCount – the number of robots the which NetApp is currently providing a service.


```
{
   "Id" : "1009941c-846f-46e0-9674-84d650876b4c"
   "Name" : null
   "HardLimit" : 10
   "OptimalLimit" : 5
   "CurrentRobotsCount" : 0
   "Timestamp" : "2022-06-22T10:20:42.2575197+00:00"
   "Relations": []
}
```


Figure 11 - Heartbeat data

4.3.9.3 Recovery process

With the distributed system always comes the need to ensure the quality of experience and maintain the system's high availability. The 5G-ERA Middleware faces the problems of every distributed system built from multiple services. On top of that, it must ensure the highest QoS for the robot utilising the system. The NetApps used by the robot may experience some unexpected issues, and the Middleware should propose a way to ensure the robot can successfully execute the desired action. To do so, the 5G-ERA Middleware system has taken a few approaches.

During the deployment of the NetApp, many defects can occur. For example, it could be an error when downloading the OCI image or its start-up. On top of this, errors could occur when NetApp works that would cause the NetApp to shut down unexpectedly. All these problems could occur and cause delays or even disallow the robot to execute the desired task, and therefore, Middleware must implement the recovery procedures.

The second approach to ensure the highest possible uptime of the services and quality of experience for the robot is to incorporate the recovery procedures into the lifecycle management of the NetApps.

There are two viable solutions for the implementation of the recovery procedures. Each of the approaches is dependent on the state of the NetApp. Based on the state, the complications of the recovery procedure will vary.

(a) Automatic restart of the services

As the stateless applications do not have the persisted data, they can be replaced quickly. For such a scenario, the Middleware incorporates the built-in Kubernetes mechanics. Each NetApp

is defined by a Kubernetes Deployment object that controls the status and availability of the NetApps and their backing services. Choosing the Kubernetes deployment could provide resource benefits over other methods of deploying them in Kubernetes. It allows users to quickly scale out the applications, perform the updates, and monitor and keep the desired number of replicas available. Using this approach, Kubernetes will automatically keep the removed pods alive. Figure 12 showcases the moment in which Orchestrator-managed Kubernetes restarts the failed service. The area selected with the red box shows one instance of the application being terminated, which has the status "Terminating", and the replacing NetApp being instantiated, with the status "ContainerCreating".

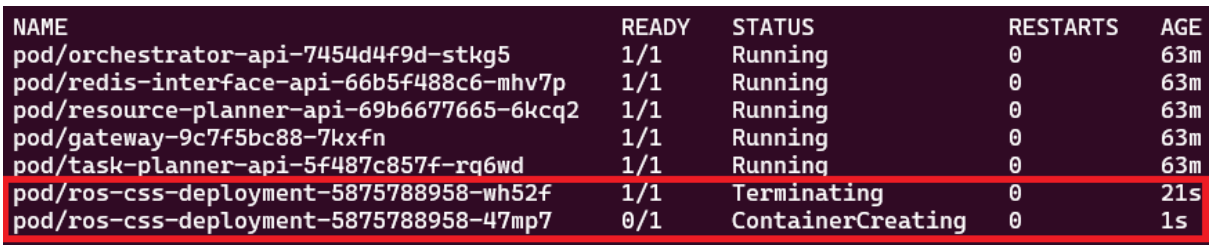

Figure 12 - Automatic service restart managed by Kubernetes

The other approach must be taken for the stateful NetApps or for the NetApps that failed during the deployment. These applications, by assumption, store data in themselves while working. When the application fails, the data is lost, and the NetApp can no longer provide the robot with critical data like SLAM. Using the Heartbeat allows for quickly detecting the eventual problems with the services. Heartbeat protocol enables the system to swiftly respond to the issue and provide a quick update to the robot.

(b) Re-plan on failure

Another situation in which the NetApp cannot provide the data to the robot is when the deployment of the NetApp fails on the Kubernetes level. While the NetApps cannot work for the robot, the issue must be mitigated.

The mitigation for the described scenarios is the re-plan. With the semantic planning of the task and the resources, alternatives could be proposed for a robot, and replacement services could be used. The re-plan could provide a robot with services that can conduct the same job but not be precisely adjusted to the robot's environment, as further described in the Task Planner section of this chapter. With the lower accuracy, the robot will be able to conduct the task, and the system will ensure the highest possible quality of experience and uptime. Independent semantic planning in the future should be able to fulfil the presented scenario.

4.3.9.4 Edge Switchover

Another functionality supported by the orchestrator is the Edge switchover. The Edge switchover is the functionality allowing the transfer of the resources utilised by the vertical application into a different localisation. For example, in the scenario that the autonomous robot conducts the work in the designated area, but during the execution of the work, it has changed the operation area to the site with a better connection with a different Edge than the one it initially connected. Due to this, the new Edge can now provide a better QoE to the robot due to shorter latency. The Action Server can request the Edge switchover when the described

situation happens. In this process, the Orchestrator deploys the services identified in the Task Plan assigned to the robot in the new location and updates the robot's plan. After the update, the robot gets the information about the change of the resource location and can adjust itself to the new localisation of the services. When the robot switches the resources utilised, and the Middleware receives the heartbeat information from the NetApps that the number of robots the NetApp is working for has increased, the Orchestrator confirms the switchover was successful. With the successful switchover conducted, the resources in the old Edge can be disposed of if they are unused by other robots.

By providing the functionality of dynamic location switching, the Middleware provides the backup solution in case of abrupt connection issues to the Edge or a power outage. The Action Server detecting that the current location is unavailable can request the switchover to continue the work in another site available. The Edge switchover enables the constant realisation of the

4.4 Summary

To summarise, the 5G-ERA Middleware is the Orchestration software responsible for linking vertical applications managed by ROS (Robot Operating System), a 5G infrastructure managed by Open Source MANO (OSM) and the cloud-native network applications (NetApps). The core principles and functionality of the Middleware cover the lifecycle management, recovery and error handling of the applications, and integration of the semantic planning into the orchestration process. With these capabilities, the Middleware serves as a way to bridge the domain knowledge with the orchestration process for third-party applications.

5 Evaluation

The Evaluation chapter covers the evolution of the 5G-ERA Middleware, validates whether the system meets the requirements, and addresses the challenges stated in the Problem definition. On top of this, the chapter covers the process of validating and integrating the Middleware with the NetApps for the on-demand resource orchestration and integrating it with the robot to ensure the pipeline with the desired scenario is fully functional in the real-world use case.

The first section of the chapter will cover the current version of the Middleware that the basic functionality was tested on. Then, the existing functionalities will be explained, what role the Middleware plays in the deployment process and on-demand resource request, and how it supports the autonomous robot in realising the desired objective.

In the following section, the chapter will describe the differences and the improvements that the 5G-ERA Middleware provides over the existing solutions. Next, the differences and improvements of using the OSM and Kubernetes for the resource orchestration will be described as how using the Middleware builds on the already solid foundations and combines the solutions into a single platform. Finally, it will showcase what benefits it brings to the scalability of ROS-native applications.

In the end, the chapter will summarize if Middleware meets the requirements specified at the beginning of the document.

5.1 Current Middleware Edition

The 5G-ERA Middleware is a complex software that requires much work to be put in to achieve the desired functionality. By the time of writing the thesis, Middleware had reached its initial version but still did not implement all the features it was designed to. The source code for the current Middleware version can be found under the link in the 5G-ERA Middleware GitHub repository section of the Appendix.

The main functionality assumed is supported by the Middleware. At the moment of writing, the Middleware supports on-demand deployment of the NetApps requested by the autonomous robots. The Middleware supports the communication process with the Vertical Application Layer using the ROS Action Server to provide a constant communication channel between the vertical application and the Middleware. The Middleware supports the entire pipeline used for the deployment of the NetApps, which utilizes the communication with the Kubernetes Resource Network Layer. The Middleware can deploy the NetApps in the plan requested by the autonomous robot. Currently, Middleware implements a simplified version of semantic planning. The plans are prepared beforehand and are selected directly by the robot to be utilised.

The critical feature currently missing in the Middleware is full semantic planning. The simplified version of the semantic task and resource planning is usable but is far from the ideal solution and does not meet the requested criteria to provide intelligent planning for third-party applications. The feature is in active development and should become part of the 5G-ERA Middleware soon. Another missing feature is integration between the Middleware and

the OSM for implementing the Resource Enablement Network Layer cooperation. The Middleware lacks communication with the OSM to dynamically enable the SDN to enhance the QoS from the NetApps to the vertical application using the 5G network.

Even when the Middleware does not implement all the necessary features yet, it already has the functionality extensive enough that implement most of the initial requirements. Through its work, Middleware already provides the bridge between the ROS-native vertical applications and the cloud-based NetApps. With the dedicated orchestration supported by the basic planning capabilities, the Middleware enhances the vertical applications enablement and improves the resource orchestration. In addition, the Middleware has the functionality needed for the evaluation process to be conducted to assess the system's viability.

5.1.1 Middleware example

The example workflow will be showcased as the working Middleware example. It will include the example robot registering with Middleware and requesting the resources for the specified task. The task definition is placed in the Middleware by the administrator beforehand.

1. The user starts with the Middleware deployed in the Kubernetes cluster with all the available services. Figure 13 shows the list of the Deployments that are considered Deployed Middleware.

```
Every 2.0s: kubectl get all -n middleware                                                    5GERA3: Tue Jun 21 15:27:12 2022

NAME                                          READY   STATUS    RESTARTS   AGE
pod/orchestrator-api-6cd4d44f9f-pjjq8         1/1     Running   0          62s
pod/redis-interface-api-66b5f488c6-ndqmk      1/1     Running   0          57s
pod/gateway-9c7f5bc88-nxnzt                   1/1     Running   0          57s
pod/resource-planner-api-69b6677665-dm6lw     1/1     Running   0          56s
pod/task-planner-api-5f487c857f-28cx4         1/1     Running   0          56s

NAME                          TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)                     AGE
service/orchestrator-api      ClusterIP      10.152.183.93    <none>         80/TCP,433/TCP              62s
service/gateway               LoadBalancer   10.152.183.95    10.64.140.43   80:32395/TCP,443:31829/TCP  57s
service/redis-interface-api   ClusterIP      10.152.183.193   <none>         80/TCP,443/TCP              57s
service/resource-planner-api  ClusterIP      10.152.183.192   <none>         80/TCP,443/TCP              56s
service/task-planner-api      ClusterIP      10.152.183.137   <none>         80/TCP,443/TCP              56s

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/orchestrator-api         1/1     1            1           62s
deployment.apps/redis-interface-api      1/1     1            1           57s
deployment.apps/gateway                  1/1     1            1           57s
deployment.apps/resource-planner-api     1/1     1            1           57s
deployment.apps/task-planner-api         1/1     1            1           56s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/orchestrator-api-6cd4d44f9f     1         1         1       62s
replicaset.apps/redis-interface-api-66b5f488c6  1         1         1       57s
replicaset.apps/gateway-9c7f5bc88               1         1         1       57s
replicaset.apps/resource-planner-api-69b6677665 1         1         1       56s
replicaset.apps/task-planner-api-5f487c857f     1         1         1       56s
```

Figure 13 - Available Middleware services in the K8s cluster

2. Next, the administrator logs in and receives the bearer token from the gateway that will be later used for the authentication. After receiving the Token, the administrator uses it to create the task definition that will deploy the image recognition NetApp for

the robot. Figure 14 shows the endpoint and JSON structure of an object passed in the HTTP request body, to create a new task definition.

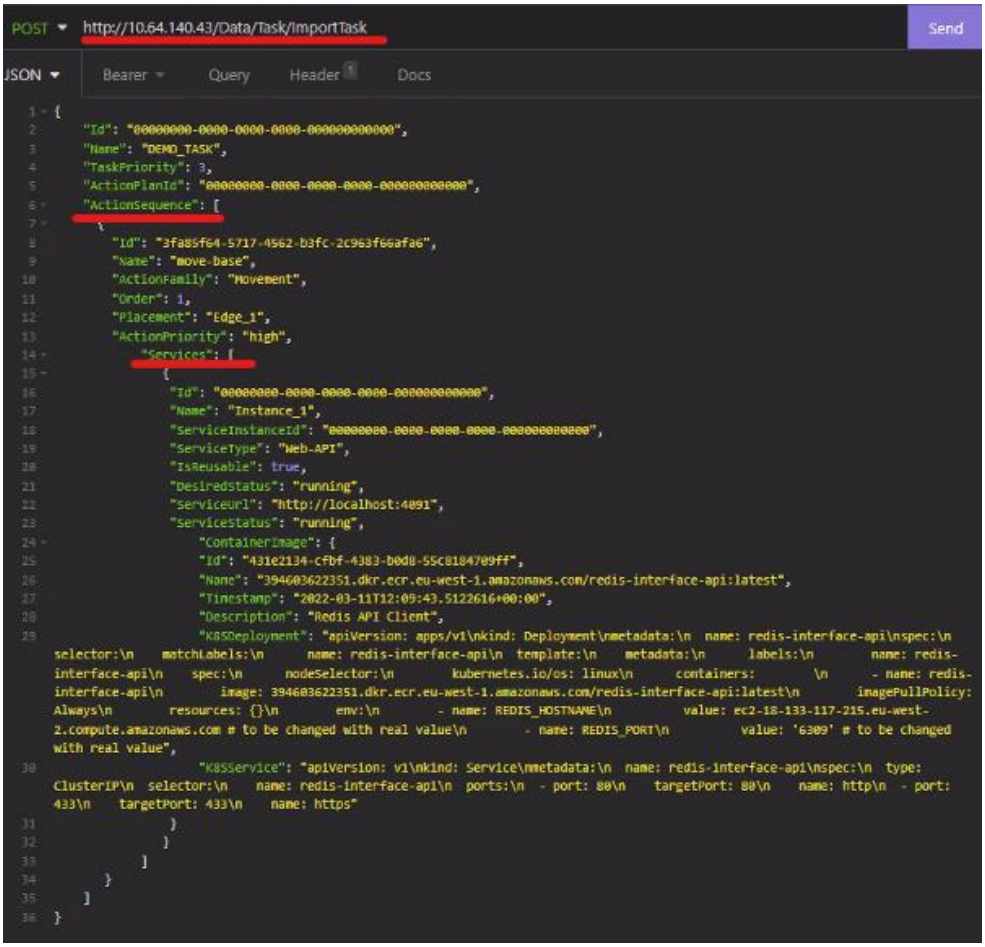

Figure 14 - Import task endpoint call

3. The autonomous robot can use the new task available through the action server. Before starting, the autonomous robot connects through the VPN to the local Edge. The vertical application starts the action server and requests the Middleware for the action plan and resources to conduct the specified task. It receives the task definition. Figure 15 shows the moment the Action Client accepts the goal, contacts the

Middleware and receives a response in the form of the Task definition that was identified to deploy an object detection algorithm.

```
os@5GMiddleware:~/ros2_galactic$ ros2 run era_5g_action_client ActionClientNode
INFO] [1655893225.916420624] [actionClient]: Waiting for action server
INFO] [1655893225.919105400] [actionClient]: Sending goal request
INFO] [1655893230.276675765] [actionClient]: Goal accepted!
INFO] [1655893230.300835042] [actionClient]: Received feedback:
INFO] [1655893230.302533032] [actionClient]: Action plan: ['fe813ad8-0df5-4d03-9872-43b70c95dfaa
]
INFO] [1655893231.362153269] [actionClient]: Received feedback: {'Id': 'b2e7d88a-88c4-4add-adab-
a07cb7574b2', 'Name': 'NetApp Standalone deployment', 'ActionSequence': [{'Id': 'fe813ad8-0df5-4
03-9872-43b70c95dfaa', 'Name': 'move-base', 'ActionFamily': 'MAchine Learning Image Recognition'
 'Order': 1, 'Placement': 'Edge_1', 'ActionPriority': 'high', 'Services': [{'Id': 'db7fcc94-3dad
4102-a82f-feb3d814308c', 'Name': 'ros-css-deployment', 'ServiceInstanceId': '1009941c-846f-46e0-
674-84d650876b4c', 'ServiceType': 'Web-API', 'IsReusable': True, 'DesiredStatus': 'running', 'Se
viceUrl': None, 'ServiceStatus': 'Idle', 'relations': []}], 'relations': []}], 'relations': []}
INFO] [1655893231.364080592] [actionClient]: Action plan: ['fe813ad8-0df5-4d03-9872-43b70c95dfaa
]
INFO] [1655893233.419020893] [actionClient]: Received feedback: {'Id': 'b2e7d88a-88c4-4add-adab-
a07cb7574b2', 'Name': 'NetApp Standalone deployment', 'ActionSequence': [{'Id': 'fe813ad8-0df5-4
03-9872-43b70c95dfaa', 'Name': 'move-base', 'ActionFamily': 'MAchine Learning Image Recognition'
 'Order': 1, 'Placement': 'Edge_1', 'ActionPriority': 'high', 'Services': [{'Id': 'db7fcc94-3dad
4102-a82f-feb3d814308c', 'Name': 'ros-css-deployment', 'ServiceInstanceId': '1009941c-846f-46e0-
674-84d650876b4c', 'ServiceType': 'Web-API', 'IsReusable': True, 'DesiredStatus': 'running', 'Se
viceUrl': None, 'ServiceStatus': 'Idle', 'relations': []}], 'relations': []}], 'relations': []}
INFO] [1655893233.421013716] [actionClient]: Action plan: ['fe813ad8-0df5-4d03-9872-43b70c95dfaa
```

Figure 15 - starting the Action Server from the robot

4. After the resources are requested from the Middleware, the new service is deployed in the Kubernetes cluster. Figure 16 presents the successfully deployed object detection application by the Middleware.

```
ros@5GMiddleware: ~
Every 2,0s: kubectl get all -n middleware                5GMiddleware: Wed Jun 22 12:21:03 2022

NAME                                        READY   STATUS    RESTARTS   AGE
pod/orchestrator-api-5f4897c99c-cnvpw       1/1     Running   0          22m
pod/gateway-5bc57dd64b-dqd56                1/1     Running   0          22m
pod/redis-interface-api-7ffb94f698-f8jrx    1/1     Running   0          22m
pod/resource-planner-api-7d686f85cf-594w7   1/1     Running   0          22m
pod/task-planner-api-79895f6b45-5xqzj       1/1     Running   0          22m
pod/ros-css-deployment-545584fc86-mzfcx     1/1     Running   0          33s

NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)
          AGE
service/orchestrator-api       ClusterIP      10.152.183.156   <none>         80/TCP,433/TCP
          23m
service/gateway                LoadBalancer   10.152.183.209   10.64.140.43   80:32408/TCP,443:30
893/TCP   22m
service/redis-interface-api    ClusterIP      10.152.183.203   <none>         80/TCP,443/TCP
          22m
service/resource-planner-api   ClusterIP      10.152.183.11    <none>         80/TCP,443/TCP
          22m
service/task-planner-api       ClusterIP      10.152.183.198   <none>         80/TCP,443/TCP
          22m

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/orchestrator-api       1/1     1            1           22m
deployment.apps/gateway                1/1     1            1           22m
deployment.apps/redis-interface-api    1/1     1            1           22m
deployment.apps/resource-planner-api   1/1     1            1           22m
deployment.apps/task-planner-api       1/1     1            1           22m
deployment.apps/ros-css-deployment     1/1     1            1           34s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/orchestrator-api-5f4897c99c       1         1         1       22m
replicaset.apps/gateway-5bc57dd64b                1         1         1       22m
replicaset.apps/redis-interface-api-7ffb94f698    1         1         1       22m
replicaset.apps/resource-planner-api-7d686f85cf   1         1         1       22m
replicaset.apps/task-planner-api-79895f6b45       1         1         1       22m
replicaset.apps/ros-css-deployment-545584fc86     1         1         1       34s
```

Figure 16 - NetApp deployed on-demand by the robot

5. After the NetApp is deployed, the robot can use the ROS-native communication to utilise the NetApp resources. In Figure 17 the topic from the NetApp is visible and can be accessed to communicate the results.

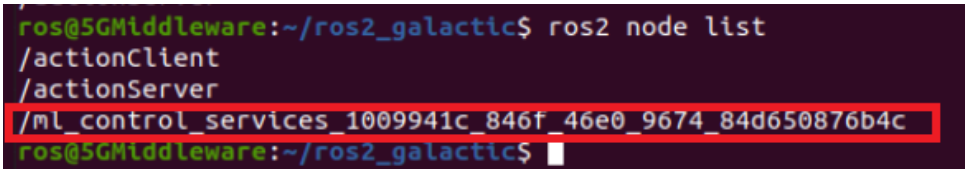

Figure 17 - NetApp node visible by ROS

6. During the work of the NetApp, the heartbeat is sent to the Middleware. The heartbeat sent from the exact NetApp deployed in this scenario is visible in Figure 11 - Heartbeat data.
7. The image below shows that the robot can access image recognition using the remote NetApp. Figure 18 shows the visual output of the object detection over the video. Detected people are shown with blue rectangles rendered on a video.

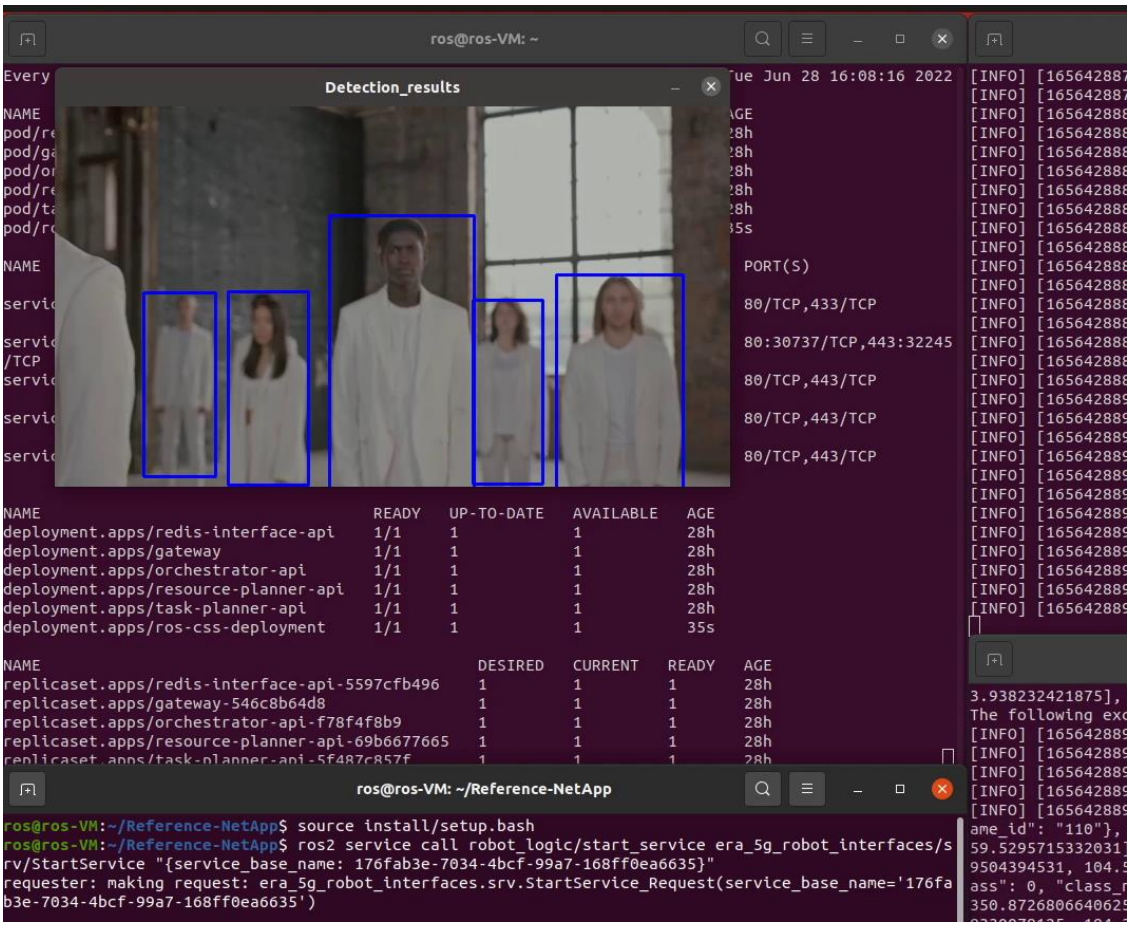

Figure 18 - Image recognition using the NetApp

8. After the robot finishes, it informs the action server and notifies the Middleware about the need to terminate the instance. After the instance is removed, the Action Server finishes its work. Finally, Figure 19 shows the Action Server starting the removal process of the deployed action sequence after receiving the goal with the

value “-1”. In the other terminal window, we can see the object detection application being terminated by the Middleware.

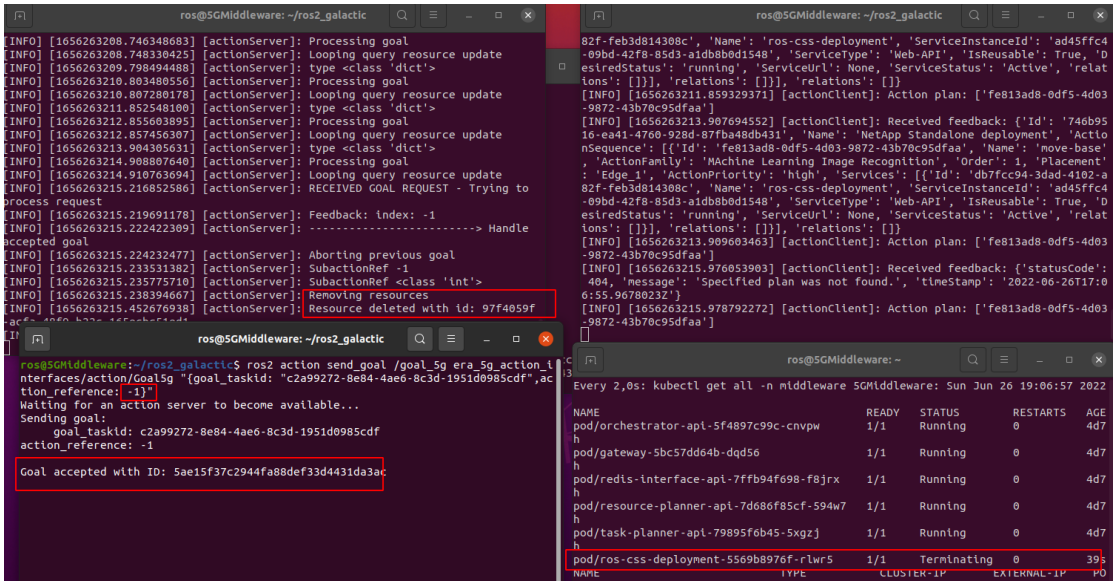

Figure 19 - Termination of the NetApp

As shown in the sequence above, the Middleware supports the resource management process for the vertical application. Starting from the definition of the task for the robot through the lifecycle management, the Middleware manages the NetApp. Middleware having the report from the Heartbeat protocol can control the state of the NetApps and react to unexpected situations that may happen to the deployed applications. At the end of the lifecycle management, the Middleware disposes of the deployed objects when they are no longer used.

5.2 Middleware improvements

The 5G-Era Middleware utilizes many state-of-the-art technologies to deliver the desired functionality. It uses Kubernetes, OSM and ROS to bridge the robotic applications with cloud-native resources. Even though the Middleware is based heavily on the functionality of these systems, they are significantly different, and the Middleware provides additional functionality on top of them.

5.2.1 Middleware and OSM

Starting with the differences between the Middleware and the OSM, these solutions provide cloud-based services under different models. The OSM uses the IaaS model to provide the infrastructure. At the same time, the Middleware works under the PaaS model to offer an extensible platform for the various use cases utilising the existing infrastructure. OSM operates on high-level functionalities like VNFs or KNFs on top of the orchestrated infrastructure under connected cloud providers like OpenStack. On the other hand, the Middleware works on the resources already deployed by the OSM, like the Kubernetes cluster. While the OSM can also provide the orchestration functionality similar to the Middleware, it does not have the dedicated functionality that manages the orchestration process according to the third-party specifications.

The crucial feature that the Middleware builds upon the OSM is the ability to directly communicate with the vertical applications to achieve better communication and provide better QoS and QoE. Thanks to the direct contact and the integration of the heartbeat protocol into the NetApps, the Middleware has a better knowledge of the operation status of the deployed resources, which results in quicker reactions to possible service failures. Furthermore, the Middleware provides a way to define the orchestration processes and the task and resource planning capabilities that incorporate domain knowledge. The Middleware utilises the knowledge graphs to store the domain knowledge needed to realize the planning on the appropriate level. This ensures the operability of the robotic verticals without the need for human intervention in the process.

Contrary to Middleware, OSM requires human intervention for resource deployment. Even if the process on the OSM side can be automated using the API, it requires creating a bespoke solution to handle this API. Thanks to its native integration with the OSM API, the Middleware can automatically schedule the resources needed when deemed necessary. In addition, OSM partially implements the PaaS model by allowing the provision and essential monitoring of the deployed services.

By extending the OSM functionality, the Middleware provides a complete experience by realising the vertical-oriented PaaS model instead of the resource-oriented PaaS model that OSM provides. The Middleware primarily focuses on delivering the end-to-end platform experience for vertical applications. It extends the already rich functionality the OSM provides to provide cooperation between vertical applications and the infrastructure and a better QoE. As a platform, the Middleware offers more benefits to the end users of the application. On top of this, the implementation of the domain resource planning and orchestration allows the Middleware to shift from the operator-based deployment to dynamic PaaS that supports on-demand deployment of the network resources and adjusts the topology to the SaaS requirements. The OSM will allow Middleware to provide the 5G network resources to the vertical applications to be utilised as a communication medium.

5.2.2 Middleware and Kubernetes

Similar to the OSM, Middleware and Kubernetes have different primary use cases. While Kubernetes is the top-of-a-class system that automates the deployment of containerised applications, the Middleware can be considered a sort of add-on to the already complete functionality. Kubernetes already has an extensive set of measures to allow the highest possible availability of the deployed resources and their provision and enablement. The Middleware focuses on providing the integration layer between the orchestration process and the vertical application. As a result, middleware offers a simplified process of deploying resources on demand by focusing on communication between the vertical application and the deployment platform. As with the OSM case, Kubernetes does not provide a deployment scheme that can be adjusted to the needs of the third-party applications. Instead, the Middleware provides the logic layer that the domain specialists can supply with the required processes for the custom vertical application services deployment.

While working in the Kubernetes environment, the Middleware provides a set of additional functionalities to the vertical applications. First, the Middleware is aware of the architecture

it is working on. Thanks to the graph representation of the architecture, Middleware can assess the best possible configuration of services for the autonomous robot. Furthermore, with domain-specific planning capabilities, Middleware can provide more optimized services tailored especially for autonomous applications. On top of this, by using the heartbeat protocol, Middleware offers closer integration with the deployed applications. This functionality allows the Middleware to have an insight into the processing capabilities of the NetApp to optimize resource usage. Heartbeat also provides the service monitoring capabilities to the Middleware, allowing it to better respond to the failures of the application execution and provide recovery plans. For recovery planning, the Middleware utilizes the exact mechanisms that use the domain knowledge for the initial planning.

Kubernetes has extensive capabilities that allow it to manage and scale an almost infinite number of applications. While the management of these applications is simple on a small scale, control of the fleet of the vertical applications and their services can become troublesome. The 5G-ERA Middleware improves upon Kubernetes by providing the integration for the fleet of autonomous robots. Managing the services and resources running in Kubernetes enhances the scalability and automates the service management, deployment and configuration of vertical applications. With the cooperation of Middleware and OSM, the Middleware can integrate the vertical applications with the 5G connectivity for end-to-end resource scalability.

The extension of the Kubernetes functionality provided by the Middleware enables the better utilization of the PaaS by integrating the SaaS applications deployed in the K8s. Furthermore, with the inclusion of the bridging of the ROS-native applications into the cloud-native domain, Middleware provides the easy transition phase for the robotic software to utilise the cloud infrastructure. In the end, the Middleware enhances Kubernetes to become a better integration platform for autonomous robots and other vertical applications. Furthermore, with the Middleware designed to work in the Kubernetes cluster, it provides behaviour transparency regardless of the location of the Kubernetes cluster, either Edge machine or public cloud.

5.2.3 Combining OSM and Kubernetes

The 5G-ERA Middleware combines the benefits of OSM and Kubernetes to enable increased scaling capabilities for the fleet of autonomous robots. Furthermore, by integrating the transparency of Kubernetes deployments and the management of the digital infrastructure by OSM, the Middleware can provide the Edge Switchover functionality.

The Edge switchover is the functionality allowing the transfer of the resources and services utilised by the vertical application into a different localisation. When working with distributed systems, some parts of the system will be closer than others. For example, with autonomous robots working in the field, there may be a situation in which the robot changes the operation area during the task execution and comes under the jurisdiction of a different Edge. In this case, a new Edge may provide a better experience, connection stability and lower latency for the robot.

The coordination of the Middleware deployment by OSM in the multiple sites allows the Middleware to integrate the autonomous robots into the large-scale distributed environment. This solution is possible for the Middleware thanks to the shared knowledge between the Middleware instances synchronised through Redis. Furthermore, the transparency provided by Kubernetes allows a seamless synchronisation and transfer of resources and services between the Middleware instances.

The Edge switchover provides significant flexibility in the distributed architectures by giving the ability to dynamically change the operation place to increase the performance of the vertical applications. The Edge switchover allows consistent resource distribution between the Middleware instances running in the different locations based on the robot distribution in the system. The described functionality allows for the increased reliability and resiliency of the system. By providing the functionality of dynamic location switching, the Middleware provides the backup solution in case of abrupt connection issues to the Edge or a power outage. The Action Server detecting that the current location is unavailable can request the switchover to continue the work in another site available.

Implementing the switchover makes the Middleware more resilient by allowing the change of the availability zones like in Kubernetes. Furthermore, the Edge switchover provides better resource optimisation and increases service availability by automatically changing the placement of the resources to be closer to the autonomous robot.

5.2.4 Benefits for ROS applications

The last benefit of the 5G-ERA Middleware is the support for the robots utilising the ROS-nave software. The Middleware supports the NetApps using cloud-native communication methods like REST or RPC to provide the highest compatibility with state-of-the-art network applications. On the other hand, most of the robotic software uses the Robot Operating System with its communication that does not work outside the Local Area Network. Therefore, the Middleware takes several steps to counter this issue and enable ROS applications to use the Middleware. First, the Middleware proposes the NetApp variance that can work using the native ROS communication through the nodes and topics. Then, to make the application work outside the LAN, the Middleware exposes the VPN endpoint for the robot to connect. After the connection is established, the robot behaves and utilises the remote NetApps like they were in the local network.

By allowing the ROS applications to use their native methods of communication, the Middleware provides enhanced scalability. As a result, Middleware designs a transition path that will allow for an accelerated movement of ROS-based technologies to the cloud by maintaining backward compatibility. Furthermore, the robot onboarding process is simplified by offloading the robotic vertical applications to the cloud. With the Middleware, making the robot operational will require less time spent configuring the robot and its sensors to work with the operation services like object detection or SLAM. In the end, Middleware enhances the robotic community by craving the path to improving applications' resource utilisation and scalability.

In the end, Middleware also provides simplified management of the robot fleet and its services. The simplified deployment of robotic services enables the easier management and deployment of robotic applications through an automated service provision process.

5.2.5 Expected optimizations

While the Middleware has not yet been tested in a real-world scenario, including robots and NetApps it could utilise, many improvements are expected. Middleware should provide the robot with the NetApps running in the local Edge or remote Clouds with the results and total latencies close to or slightly lower compared to local processing. On top of this, the offloading of the applications to the cloud should release most of the CPU usage and part of the RAM. More demand will be put on the network interfaces of the robot, but due to the lower power consumption of the network cards compared to the CPU, a reduction in battery usage is also expected. While the network traffic can increase, it is not expected to put a strain on the current generation network card capabilities.

To summarize. Usage of the Middleware in the real-life scenarios should bring the following results:

1. Decreased CPU usage
2. Decreased RAM usage
3. Higher NIC utilisation
4. Decreased battery usage
5. Robot operating longer on a single battery charging

5.3 KPI assessment

In this section, the KPIs defined in the introduction are assessed and verified if the Middleware implements the desired functionalities and meets the KPIs. The results are presented in the table below. The table divides the evaluation metrics into three sections: viable performance assurance, predictable performance assurance and plug-and-play performance. The three sections focus on providing reliable networks for sustained high throughput and close to real-time communication, behaviour of vertical applications within their specific and expected performance levels as well as to enable plug-and-play to expect interoperability and guarantee the performance levels.

Functionality	OSM	Kubernetes	Middleware
Viable performance assurance			
Realises native digital resource orchestration by connecting cloud-based virtual resources	✓	X	Through communication with OSM
Enables the 5G network over the testbeds	✓	X	Through communication with OSM
Flexible placement of the services in the available digital infrastructure	✓	✓	Through communication with OSM and Kubernetes
Increases QoE for vertical applications by providing increased resource availability and network connectivity with the best performance available	X	X	✓
Predictable performance assurance			
Middleware realises semantic resource planning with the inclusion of third-party domain knowledge	X	X	✓
Allows the robotic fleet management at a large scale by providing end-to-end connectivity between the vertical applications and online services	X	✓	✓
Provides simple service orchestration management	X	✓	Through communication with Kubernetes
Middleware enables automatic service scalability	✓	✓	Through communication with Kubernetes

Table 2 - KPIs assessment through OSM and Kubernetes comparison

Predictable performance assurance			
Integrates vertical-level applications with the resource provision under a unified platform	X	X	✓
QoE realised by providing a seamless deployment service provision through a dedicated orchestration process	X	X	✓
Acts as a bridge between existing ROS applications to provide an easier transition to utilise the cloud-based services	X	X	✓

Table 3 - KPIs assessment through OSM and Kubernetes comparison

5.4 Summary

The Evaluation chapter has covered various aspects of Middleware and how it compares to existing solutions like OSM and Kubernetes. The chapter has presented the current state of the Middleware and the existing functionality in action to showcase the improvements already present, even if the software is not fully ready. First, the core functionality was showcased, including the whole pipeline from the robot requesting the resource plan until the deletion of the resources when the robot finished its work. The later parts of the chapter described the improvements coming from the Edge switchover functionality and how the Middleware combines the functionality of OSM and Kubernetes to achieve it. In the end, the Middleware has explained how it improves ROS-native communication and creates a path for adopting the ROS applications in cloud-native environments.

First, the current Middleware development progress has been presented. The Middleware allows the deployment of the desired resources in the Kubernetes environment and monitors them throughout their lifecycle. The delivered functionality has been described in a scenario when the robot has requested the Middleware to deploy the imported action plans to conduct the image recognition task. The Middleware provides an additional functionality layer for the OSM to realise the PaaS model on top of the IaaS model provided by the OSM. The improvements from the semantic planning of the resources improve the infrastructure provision using the OSM and the cloud-native resource deployment in Kubernetes by providing more detailed orchestration plans for the desired domain use case. Furthermore, the Middleware improves Kubernetes by providing tighter integration between the NetApps playing the role of the SaaS model with the management in the form of the PaaS.

In addition, Middleware provides an easier transition for the robotic industry in moving resources to the cloud. Therefore, Middleware improves the ease of deployment of robotic applications and their scalability. The final form will allow the robots to be an integral part of the distributed system.

In the end, the KPIs described in the Introduction were assessed against the assurances described in the Time and Material methodologies. As a result, the 5G-ERA Middleware platform enables seamless cooperation between the robotic applications and bridges the domain knowledge utilisation with the resource orchestration to provide better usability and availability of services for the vertical applications.

6 Conclusion

The 5G-ERA Middleware is expected to close the gap between robotic applications and 5G using the cloud-native design. Handling the orchestration process of the complex services allows connecting the robots to the cloud-native systems to maximise the capabilities of the 5G networks. Middleware provides a flexible and intelligent selection of the needed actions and resources to integrate with the semantic knowledge base (I3M, 2021). Furthermore, the lifecycle management of the Net Apps combined with the recovery procedures managed by the Middleware offers the reliability of the system and the QoE for the autonomous robot. By shifting the deployment process from the networking aspect and the IaaS paradigm to the PaaS model, the Middleware offers enhanced flexibility in dynamically providing cloud-based resources. In the end, the 5G-ERA Middleware is supposed to provide a paradigm of bridging the domain knowledge with the orchestration process.

The Literature Review has described a number of the existing orchestration technologies, software architecture paradigms and cloud service models. Starting with the software architecture description, the comparison has been made between monolithic and microservices architecture and how they fit into the cloud-native solutions. The hybrid of these two, the distributed monolith, has been included in the equation to combine the benefits of both previous types and decide which architecture best fits the Middleware. Following is the description of the cloud service models like Platform as a Service and Infrastructure as a Service. The IaaS and PaaS were compared and assessed for what kind of functionality they provide. Finally, the orchestration technologies like Kubernetes, OSM and Infrastructure as Code were compared. The orchestration capabilities of Kubernetes and OSM have been evaluated, and the benefits they bring to the different resource layers were described. Finally, the Robot Operating System and its role in creating robotic vertical applications have been described. The Network Applications have been identified as a set of network services designed to provide a specific functionality over the network. In the end, the 5G Network has been described as a medium that is supposed to connect these applications.

The discoveries made in the literature review have led to the design of the 5G-ERA Middleware, the platform that enables the communication and on-demand orchestration of the NetApps for vertical applications, including the custom third-party planning processes. The Middleware was designed as a distributed monolith that connects three network layers: the Resource Enablement Network managed by OSM, the Resource Layer Network controlled by Kubernetes and the Vertical Application Layer Network that hosts ROS vertical applications and the NetApps. The communication between layers has been achieved by combining the solutions native to each layer, mostly the REST APIs and the ROS Action Server. Furthermore, the Middleware has divided its functionality of managing the lifecycle of the NetApps, customized tasks and resource planning into separate components to mimic the microservices architecture. The division of the application into several smaller parts has allowed Middleware to achieve the separation of concerns required for the easy maintenance and development of the system. In the end, the Middleware was designed as the cloud-native application to run in the Kubernetes cluster to provide high transparency.

The 5G-ERA Middleware provides a set of functionalities that allow the third-party domain knowledge in the planning and orchestration process for deploying and managing the cloud-based resources for the vertical applications. The Middleware offers the complete lifecycle management of the deployed applications connected with monitoring processes like Heartbeat and the recovery functionality that allows recovery to occur when the application malfunction is detected. On top of this, if the application cannot be recovered, it is scheduled for the replanning process. The distribution of the services allows Middleware to create a distributed system composed of multiple Middleware instances. Such an approach allows for the edge switchover functionality that enables the vertical application to dynamically change the operation area to the one that provides better network capabilities. Such an approach allows for the increased scalability and resiliency of the whole system.

Finally, the outputs were evaluated. The latest version of the system has been presented with its current functionality. Further on, the Middleware has been compared to the OSM and Kubernetes. The main benefits of the Middleware are the dedicated orchestration for the third-party applications and the increased integration with the deployed applications due to the Heartbeat protocol. The Middleware has showcased its upper hand on the scalability of robotic applications by providing the bridging solution that enables the easy transformation of the existing monolithic vertical applications into cloud-native solutions.

6.1 Future Directions

The Middleware is a ready-to-use product. After the development stage, there is now a time to start testing. The testing should include how the Middleware behaves and how it serves the robots. The results will be compared to the expected optimizations, whether are they valid and to what extent they improve the robot's operation in the unstructured network cloud environments.

Finally, for Middleware to be fully utilised and to achieve its full potential, a wider user group is needed. The Middleware needs to be promoted as a solution to transfer the existing solutions from the robots to the cloud using the presented framework. How the wider public adopts it will be a determinant of how well Middleware fits the current or the future market for robotic software solutions.

7 Appendix

7.1 5G-ERA Middleware GitHub repository

https://github.com/5G-ERA/middleware

8 References

I3M, 2021. *International Multidisciplinary.* [Online] Available at: https://www.msc-les.org/i3m2021/ [Accessed 30 06 2022].

Redis Ltd, 2022. *Redis cluster specification.* [Online] Available at: https://redis.io/docs/reference/cluster-spec/ [Accessed 27 10 2022].

5GASP, 2021. *5GASP definition of a NetApp.* [Online] Available at: https://community.5gasp.eu/index.php/2021/07/12/what-is-a-netapp/ [Accessed 30 09 2022].

5G-PPP Software Network Working Group, 2022. *NetApp: Opening up 5G and beyond networks.* [Online] Available at: https://5g-ppp.eu/wp-content/uploads/2022/10/5G-PPP-Software-Network-White-Paper-V9.pdf [Accessed 25 10 2022].

Aguilera, M. K., Chen, W. & Toueg, S., 1997. *Heartbeat: A timeout-free failure detector for quiescent reliable communication.* Saarbrücken, Springer Berlin, Heidelberg, p. 126–140.

Artac, M. et al., 2017. DevOps: Introducing Infrastructure-as-Code. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C).* s.l.:IEEE, pp. 497-498.

Benj Edwards, C. H., 2022. *What is 5G, and how fast will it be?.* [Online] Available at: https://www.howtogeek.com/340002/what-is-5g-and-how-fast-will-it-be/ [Accessed 22 09 2022].

Butler, B., 2013. *networkworld.com.* [Online] Available at: https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html [Accessed 07 10 2022].

Canonical Ltd, 2022. *Onboarding Network Functions.* [Online] Available at: https://ubuntu.com/telco/osm/onboarding-network-functions [Accessed 30 06 2022].

Cynthia Rudin, C. C. Z. C. H. H. L. S. C. Z., 2021. *Interpretable Machine Learning: Fundamental.* s.l., s.n.

Earl, T., 2005. *Service-Oriented Architecture: Concepts, Technology & Design.* s.l.:Prentice Hall PTR.

Edwards, B., 2022. *howtogeek.com.* [Online] Available at: https://www.howtogeek.com/340002/what-is-5g-and-how-fast-will-it-be/ [Accessed 07 10 2022].

Elgan, M., 2018. *securityintelligence.com.* [Online] Available at: https://securityintelligence.com/how-to-prepare-for-the-coming-5g-security-threats/
[Accessed 07 10 2022].

Etheredge, J., 2018. *simplethread.com.* [Online] Available at: https://www.simplethread.com/youre-not-actually-building-microservices/ [Accessed 06 10 2022].

ETSI OSM, 2020. *OSM Usage.* [Online] Available at: https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html#deploying-your-first-network-service
[Accessed 28 10 2022].

ETSI OSM, 2022. *OSM.* [Online] Available at: https://osm.etsi.org/
[Accessed 20 05 2022].

Fiedler, M., Hossfeld, T. & Tran-Gia, P., 2010. A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, March-April, 24(2), pp. 36-41.

Forest Interactive, 2020. *forest-interactive.com.* [Online] Available at: https://www.forest-interactive.com/newsroom/positive-5g-outlook-post-covid-19-what-does-it-mean-for-avid-gamers/
[Accessed 07 10 2022].

Fowler, M., 2015. *martinfowler.com.* [Online] Available at: https://martinfowler.com/bliki/MicroservicePremium.html
[Accessed 06 10 2022].

Harris, C., 2022. *atlassian.com.* [Online] Available at: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith
[Accessed 05 10 2022].

Horwitz, J., 2019. *venturebeat.com.* [Online] Available at: https://venturebeat.com/mobile/the-definitive-guide-to-5g-low-mid-and-high-band-speeds/
[Accessed 07 10 2022].

Lauretis, L. D., 2019. *From Monolithic Architecture to Microservices Architecture.* Berlin, Germany, IEEE.

Lueth, K. L., 2018. *iot-analytics.com.* [Online] Available at: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/
[Accessed 07 10 2022].

Manuela-Ioana Csoma, B. K. R. B. I.-A. I. A. K., 2020. *Management and Orchestration for Network Function Virtualization: An Open Source MANO Approach.* Bucharest, Romania, IEEE.

Mell, P. & Grance, T., 2011. *The NIST Definition of Cloud Computing,* Gaithersburg: U.S. Department of Commerce.

Michael Wittig, A. W., 2015. *Amazon Web Services in Action.* s.l.:MANNING.

Microsoft, 2022. *azure.microsoft.com.* [Online] Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas/#products
[Accessed 07 10 2022].

Open Robotics, 2021. *ROS: Home.* [Online] Available at: https://www.ros.org/
[Accessed 20 05 2022].

Qiu, R. et al., 2022. Intent-Based Deployment for Robot Applications in the 5g-Enabled Non-Public-Network. *ITU Journal on Future and Evolving Technologies,* Issue Accepted.

Quigley, M. et al., 2009. ROS: an open-source Robot Operating System. *ICRA workshop on open source software,* 3(3.2), p. 5.

Rash, W., 2018. *IT Needs to Start Thinking About 5G and Edge Cloud Computing.* [Online] Available at: https://au.pcmag.com/it-management/51666/it-needs-to-start-thinking-about-5g-and-edge-cloud-computing
[Accessed 30 06 2022].

Richardson, C., 2018. *Microservices Patterns.* s.l.:Manning.

Senapathi, M., Buchan, J. & Osman, H., 2018. DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018,* p. 57–67.

Stack Exchange, 2022. *2022 Developer Survey.* [Online] Available at: https://survey.stackoverflow.co/2022/#overview
[Accessed 30 06 2022].

Tellez, R., 2019. *The Robot Report.* [Online] Available at: https://www.therobotreport.com/speed-robotics-development-cloud-ros/
[Accessed 22 09 2022].

The Linux Foundation, 2022. *What is Kubernetes?.* [Online] Available at: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
[Accessed 30 06 2022].

Torre, C. d. l., Wanger, B. & Rousos, M., 2022. *.NET Microservices: Architecture for Containarized .NET Applications.* 6.0 ed. Redmond: Microsoft Developer Division, .NET and Visual Studio product teams.

Verizon, 2020. *verizon.com.* [Online] Available at: https://www.verizon.com/about/our-company/5g/5g-latency [Accessed 07 10 2022].

Victor Medel, O. F. R. J. A. B. U. A., 2016. Modelling Performance & Resource Management in Kubernetes. In: *Proceedings of the 9th International Conference on Utility and Cloud Computing.* Shanghai, China: Association for Computing Machinery, p. 257–262.