MsBOT: An Open-Source Plug-and-Play Distributed Robotic Framework for Education

A thesis presented in partial fulfilment
of the requirements for the degree of

Master of Engineering
in
Mechatronics

at

Massey University Albany, New Zealand.

Siow Lim Chow

September 2022

TABLE OF CONTENTS

Abstract

Educational robots are inherently vertically integrated and stand-alone in nature. Despite, by comparison, being cost-effective and easy to use, the vertically integrated robots suffer greatly in terms of technology disagreement amongst manufacturers and limited choices for users. User choices are locked into the manufacturers' resolutions, including product maintenance, updates, etc., resulting in reduced competition among robotic manufacturers. In contrast, horizontally integrated robotics encourages technology openness resulting in interoperability amongst robot manufacturers and empowering users to make the best decision that suits them the most, be it quality, pricing, or both.

The fast-paced learning environment and increasingly complex robotic curriculum have prompted the need for resolutions that improve the quality of educational robotic systems, which can improve robotic interoperability by reducing human intervention, enabling easy components replacement and new component construction and testing, leaving teachers and students more time focusing on teaching and studying.

One potential solution for robotic interoperability is the plug-and-play (PnP) capability facilitating auto-detection in the presence of a part and auto-configuration of the part accordingly without human intervention. However, the PnP must be compliant with an integrative standard agreeable to robotic manufacturers and communities. A PnP that does not support a de-facto standard is another form of vertical integration.

This thesis implements a PnP system that uses CAN-Bus as the transport protocol for connecting micro-ROS clients and agents. This contrasts with usual TCP/UDP transports but gets inspiration from automotive sectors where CAN-Bus is reliably used in almost all modern vehicles. The implemented system is called MsBOT (Massey Robot). MsBOT aims to rectify the weaknesses of the inherent vertically integrated educational robotic systems and aspires to create a PnP framework for the education industry, if not robotic industries, and produce a functional PnP education robot for others to refer to.

In MsBOT components, the CAN-Bus is established through the ISR routines and the circular buffers. The MsBOT PnP leverages the open-source Micro-ROS robotics technology that provides the PnP feature of the micro XRCE-DDS agent and the APIs to implement micro XRCE-DDS Clients. The implementation is called MsBOT 2-step piggyback PnP operation, where another PnP feature is built on top of the other PnP.

The outcomes of the thesis are: 1) An understanding and identification of the weaknesses and strengths of the currently available educational robotic systems based on a comprehensive

literature review on the currently available robotics technologies, 2) Comprehension and matching up the robotics technologies for accomplishing the MsBOT, 3) Identification of the potential robotics technology gaps and outline the research interest for the MsBOT, 4) Development of the MsBOT, 5) Test, verification and reflection, and 6) MsBOT reflection and discussion, and conclusion.

1 Introduction

Robotics is a multi-disciplinary model of engineering, computer science, artificial intelligence, human interaction, behavioural science, and many other disciplines. Robotics has the great advantage of turning abstract concepts of real-world problems and solutions [1] into tangible objects that students can see, touch and play with their project [2]. Educational theorist Seymour Papert argued that the benefits of introducing tangible objects like educational robotics are to make the process of thinking and learning visible and to allow process-oriented experiential learning through an iterative construction and de-construction process [3]. In addition, the physical presence of robotics provides a vivid outcome allowing continual formative assessment and encouragement to students [4].

Much research has proven the effectiveness of using robotics as educational tools for teaching science, technology, engineering and mathematics (STEM) at a tertiary level [5]. Besides STEM, robotics is also suitable for non-STEM disciplines and university research projects [6]. From the pedagogical perspective, robotic systems also encourage deep and active learning [7] and collaborative learning for building social skills. In many cases, robotic systems often utilise knowledge in the real world or across time, space, and contexts [8]. Other benefits of robotic systems are their capabilities and adaptive nature to constant changes driven to meet fast-paced and complex multi-disciplinary requirements.

The Challenges

Nevertheless, nothing is perfect; the same applies to robotic systems, not forgetting educational robotic systems. Educational robots are inherently vertically integrated and stand-alone in nature. Users of vertically integrated robotic systems have no privileges to mix and match components of other manufacturers; instead, their options are locked into the manufacturer's discretion, be it maintenance and product updates, etc. Robotic systems are incompatible with among manufacturers resulting in reduced competition among robotic manufacturers. The vertically integrated systems are seemingly cost-effective and easy to use; however, it is at the expense of the advancement of the greater robotics technologies.

Meticulous and time-consuming configuration tasks

The growing recognition of the importance of educational robotic systems for effective learning has encouraged more education institutes to adopt robotic systems in STEM. In

contrast, many have attempted to explore robotic systems in multi-disciplinary curricula. However, the increasingly complex robotic curricula activities have increased the occurrence of setting up and configuring activities. However, configuration tasks of robotic systems are inherently meticulous and time-consuming. Only someone well-trained is allowed to perform it. In addition, it is a painful process to keep up system knowledge of the robotic systems and keep track of the detailed configurational knowledge by teachers and students; otherwise, teachers and students would focus on teaching and learning, respectively.

Repetitive Unmeaningful tasks

Education institutions deal with high-volume and complex curricula. The high-volume yet multi-disciplinary, fast-paced curricula implicate high occurrences of system configuration. Every curriculum is different, and it needs different settings for every chapter, resulting in setting up a robotic system in such an environment can be challenging. Teachers and students find it difficult to upkeep with the repetitive configuration and new changes in configurational details. It raises a concern for teachers and students to spend time on unmeaningful tasks which can otherwise use for teaching and learning.

Vendor lock-in *[9]* ***and*** **un-evolvable systems**

Robotics, especially educational robotic components, are predominantly vertically integrated systems. Vertical integration featuring proprietary design, limiting competition amongst manufacturers and locking in customers' choices [10] directly affect the robotic systems' ecological system development. New components of better performance purchased off the shelf are unlikely to work on different vendors, resulting in un-evolvable systems inhibiting performance and functionality upgrades. The vendor lock-in resulted in reduced competition among robotic manufacturers. In addition, at times when the system breakdown, the robot owners have no options to use alternative solutions to maintain the robotic in most cases. Instead, it needs technicians from the manufacturer. The approach is not only costly but time-consuming as waiting times and schedules can be long and inconvenient. In contrast, horizontally integrated robotics empowers users to make the best decision that suits them the most and encourages technology openness resulting in positive competition and interoperable parts amongst robot manufacturers.

The Resolution – MsBOT 2-Step Piggyback Plug-and-Play Implementation

The issues of the educational robotic systems have prompted the immediate need for an autodetecting and autoconfiguring Plug-and-Play (PnP) system. The Plug-and-Play feature aspires to help teachers and students ease up the configuration of robotic systems, including component replacement and new component development and testing with little effort, time, and knowledge, leaving teachers and students more time to focus on teaching and studying. Plug-and-Play of robotic systems is a convenient feature facilitating auto-detection in the presence of a part and auto-configuration without human intervention. Ideally, the Plug-and-Play is the ultimate resolution for educational robotic systems, if not all robotic systems. Besides saving time and cost, plug-and-play can also improve educational robotic systems quality, encourage educational ecosystem development, and improve the interoperation ability of robotic components. Finally, keep the operation and maintenance cost-effective producing a sustainable system. Notwithstanding the development of the MsBOT Plug-and-Play framework being at its infancy stage, the MsBOT Plug-and-Play framework sets to be a definitive reference to proliferate more research activities.

However, the Plug-and-Play is only meaningful if it is a product of an integrative standard pursued by most robotic manufacturers. The Plug-and-Play capability is a causal effect of the issues relating to the integrative standard management, not the cause of the disagreement among robotic communities and manufacturers. The Plug-and-Play feature will manifest itself only if the manufacturers and the communities come to an agreement of adhering to one standardised robotic infrastructure that forms the integrative standard. Consequently, the robotic technologies will soar to the next level.

Disproportionate Robotics Research Activities

Evidently, the benefits of robotic systems have attracted many interest groups, commercial and non-commercial communities to conduct robotics research. However, many robotic research activities disproportionately emphasise enriching the capabilities of robotic systems [11], overlooking the importance of standardisation and code reusability [10]. Despite benefiting the robotics technology, however, the disproportionate emphasis outweighs the benefits of having a universal integrative standard resembling PC Plug-and-Play. Educational robots, too, suffer from the same dilemma. Many institutions resort to paying higher prices for maintaining un-interoperable components and upkeeping with new technology. Conversely, if Plug-and-Play had more research activities on educational robotics, it would have created a whole new

robotic integration landscape. At least, education institutes would have greater autonomy and robotic technology.

PnP without a de-facto standard is another form of vertical integration

Plug-and-Play is only meaningful if it is a product of an integrative standard pursued by robotic communities and manufacturers. Conversely, Plug-and-Play without a de-facto standard is another form of vertical integration. The robotics community has recognized the importance of standardised robotic infrastructure for accomplishing the Plug-and-Play that enables interchangeable components across the board. Therefore, many communities have defined robotic ontologies and specifications to introduce a common integrated robotic infrastructure. Despite being invested in numerous and enormous efforts, Plug-and-Play robotic integrative standardisation efforts are work-in-progress.

Needs Support from All Quarters

Nonetheless, there are too many dissonances disagreeing with implementations and divergences resisting the formation of an integrative standard. It is not a simplistic matter of merely being technically optimal and meeting all parties' interests and opinions, but it is much more involved and complicated than it seems. A joined effort from the global technology firms, academics, and governments will help support and meet the industry's demand. Government initiatives such as providing various kinds of incentives to partakers may also help overcome the resistance and catalyse the formation of global integrative standards.

Robotic Middlewares

Despite acknowledging the benefits of having one integrative standard and attempting to create many integrative propositions and international agreements [12] and ontologies [13] in the robotics domain, the industry still has not accepted and adopted these models—nevertheless, many communities' approaches to accomplishing the integrative standard differently. Instead of providing Plug-and-Play products, many facilitate initiatives, ontologies, specifications, and implementations collectively called middleware, enabling manufacturers to develop robotic systems based on the predefined integrative standard software libraries of specifications implementations.

Contributions from Robotic Communities

One of the prominent robotics standard contributors is the Object Management Group (OMG) [14, 15], which has established Robotics Domain Task Force (Robotics-DTF) [16] to promote horizontal integration amongst robotic modular components. DTF not only facilitates the dynamic deployment but also allows auto-configuration of robotic technology components that help achieve the plug-and-play objective [10]. In addition, OMG also has established the robot technology function modules (RTC) [17, 18], called robot modules. The RTC is composed of software that realises a set of robotic functionalities. Furthermore, RTC facilitates communication among the modules through the distributed object layer.

OMG's Distributed Data Service (DDS) is another significant contribution to robotics advancement, which aspired to address real-time performance, dependable, interoperable, Quality of Service (QoS), security, resource-constrained devices and many more. ROS2 [19, 20] has adopted the DDS's machine-to-machine networking middleware for real-time distributed communication systems [21]. Besides OMG's DDS and its variants, Common Object Request Broker Architecture (CORBA) [22, 23], ROS2, Orca [24, 25], and Component-Based Software Engineering (CBSE) [26] are well-established software principles and middleware. They have widely accepted open-source software platforms for accomplishing Plug-and-Play robotic or distributed robotic systems. 1.2.14 DDS for eXtremely Resource Constrained Environments (DDS-XRCE).

Leveraging on the increasingly advanced Microcontroller and networking technology, the adoption of single-board computers (SBCs), which are made of Microcontrollers, is gaining acceptance in robotic systems. However, DDS is oversize Microcontrollers' Flash storage limits. Its DDS messaging protocol and processing bandwidth are beyond what Microcontrollers could normally handle. OMG overcomes the limitations by redefining the DDS to DDS-eXtremely Resource-Constraint Environment (DDS-XRCE) protocol [27], facilitating Microcontroller-based systems, including robotic systems to access to DDS Global Data Space [28, 29]. The DDS-XRCE protocol work by having at least a source-constrained device, namely XRCE-Clients and an XRCE Agent (server) [14]. The client-server architecture offloads the massive DDS processing load between the XRCE-Clients and the XRCE Agent(server), which acts on behalf of its clients in the DDS Global-Data-Space [29].

DDS-XRCE specifications exist in many different implementations by different vendors. Amongst the DDS-XRCE implementors, **eProsima** is the most prominent. eProsima's DDS-XRCE is called **Micro XRCE-DDS** [28, 29], which is an open-source middleware. The

middleware is also called Micro-ROS [30-32], and it is the official extension of ROS 2 for Microcontrollers (MCUs). Besides, Micro XRCE-DDS provides both a plug-and-play Micro XRCE-DDS Agent and an API layer which allows users to implement your Micro XRCE-DDS Clients [28].

The Transport for MsBOT

In the MsBOT project, development is not limited to the Plug-and-Play Applications through the XRCE-DDS middleware. In fact, it was also tasked to explore and charter the unchartered territories. CAN-Bus has gotten the attention. CAN-Bus is well-known for its resilience to noise, which has been widely accepted in the automotive industry but is yet adopted in the DDS, especially in the XRCE-DDS implementations.

The OMG's DDS and its derivative DDS-XRCE are predominantly capitalised on advanced networking and computer technologies to meet the high-speed and large-size messaging communication protocols. The eProsima Micro-XRCE-DDS, which is derived from DDS, also predominantly supports wired Ethernet UDP/TCP/IP transport protocols. Seeing robotic systems resembling automotive, it is logical to apply CAN-Bus in XRCE-DDS. In CAN-Bus protocol, the CAN standard classic 2.0A characteristics of being an 8-byte payload and supporting bitrate up to 1 Mbps. Instead of using TCP/UDP transports, MsBOT would challenge to adopt CAN-Bus for connecting Micro-ROS clients and Micro-ROS agents.

Besides, CAN-Bus has ported and has gained tremendous success in PC systems, be it Microsoft Windows or Linux and MACOS. The well-established SocketCAN [33, 34] is a networking stack of a set of open-source CAN drivers formerly known as Low-Level CAN Framework (LLCF), contributed by Volkswagen Research to the Linux kernel. The SocketCAN protocol, namely PF_CAN, adopts the Berkeley sockets API in Linux, which coexists with PF_INET for the Internet Protocol. The SocketCAN implements the CAN device drivers as network interfaces which resemble the TCP/IP protocols with the intention of shortening the learning curve for programmers who are familiar with network programming.

The transport of the MsBOT, which implies it transports CAN messages amongst the MsBOT Components, also called Micro-ROS Clients and the MsBOT Master, also called Micro-ROS Agent. The onboard CAN controllers in MsBOT Components accomplish the CAN-Bus, the ISR routines, and the circular buffers. In contrast, the Micro-ROS Agent runs on Linux systems, which leverage in-system CAN Socket kernel device driver to establish the CAN-Bus.

The MsBOT aspiration

MsBOT project was inspired to rectify the weaknesses of the inherent vertically integrated educational robotic systems and aspires to create a Plug-and-Play framework for the education industry, if not robotic industries. The MsBOT project is not a non-empirical expedition, but rather a practical and real-world robotic system development. It is through a series of pragmatic activities to explore, analyse and finally reach the final decision. Despite being an academic project, it was determined to produce a real-life working Plug-and-Play education robot, which teachers and students can use for teaching and learning respectively. The MsBOT project accomplished the goals through a series of activities, which are organised in the following chapters in this thesis.

Chapter 1 – Introduction. It provides an overview of the MsBOT development strategies. Began addressing the weaknesses of the prevalent, educational robotic systems including, but not limited to, meticulous setting up and revealing the issues of the inherent vertically integrated robotic systems for being vendor lock-in hampering the progress of technology openness and the forming integrative standard. It also briefs prominent resolutions in the form of middleware that implement robotics specifications, ontologies, software libraries, and tools at developers' convenience. The MsBOT system design, Plug-and-Play resolution, and MsBOT CAN transport were discussed.

Chapter 2 - Literature Review. It discusses the literature review methodology and explores the existing robotics technologies relating to distributed robotic systems, component-based software engineering, and Plug-and-Play implementations. It reviews in-depth some of the prominent middleware for developing robotic systems, especially OMG's DDS, XRCE-DDS and its variant. Nonetheless, it reviews literature that implicates the shortcomings of vertical integration and advocates for the strengths of horizontal integration products.

Chapter 3 – Methodology. It details the MsBOT design and implementation, from the very high-level conceptual MsBOT system architecture to the module's coding. It provides the conceptual design of the MsBOT software system architecture, the MsBOT hardware design and the MsBOT user application system design. The middleware and the Plug-and-Play feature of the MsBOT are discussed in conjunction with the corresponding design and implementation. Finally, it reviews the design and implementation of the customed debugging tools. MsBOT adopts bottom-up stepwise development principles where it develops the bottom modules prior to the upper modules. Hence, the MsBOT began development with the CAN-bus development

for both the MsBOT Master and the MsBOT Components, followed by the circular buffer and then the user applications on both MsBOT Master and MsBOT Components.

Chapter 4 – Test and Results. It provides an in-depth application of the test and verification principles adopted in the MsBOT development. The principle, namely Plan-Do-Check-Act (PDCA), was applied to every MsBOT module. It begins with the first CAN-Bus trialling module and ends at the relevant Plug-and-Play modules. At the end of the test, a reflection is performed to reflect on the strengths and shortcomings, future improvements, or new design of the module-under-test. Although the reflection is not an obvious process in the PDCA test regimes, in MsBOT development, the reflection is included to reveal issues and potential resolutions in relation to the module-under-test.

Chapter 5 – Discussion and Conclusion. It provides module and system-level reflection of the MsBOT design and implementation, including but not limited to the modules of CAN-Bus design and implementation and the modules of the Plug-and-Play design and implementation. No design is perfect; the issues, the root cause, and potential resolution relating to the CAN-Bus as the transport protocol for the MsBOT are discussed. It ends with a proposition for future research activities and derives the conclusion.

The source code of the MsBOT study is stored at the author-created GitHub repository [35] https://github.com/vibnwis. This repository is divided into three directories, as described below.

- Micro-XRCE-DDS – It contains the source code of the MsBOT user applications, namely, the Health-Inventory system, the MsBOT-Master first-step authentication system, the call-back timer system, USBGPIO system, uclient_AA, uclient_BB, uclient_CC, and more many
- Micro-XRCE-DDS-Agent – It contains the source code of the Micro-XRCE-DDS Agent, also called uAgent, which has established the CAN-Bus transport using the Linux built-in SocketCAN APIs.
- Micro-XRCE-DDS-Apps – It contains the source code of the Micro-XRCE-DDS Client, also called uClient, which are the STM32E407F software development, namely, the CAN-Bus interrupt ISR routine, the circular buffer routines, and app.c.

2 Literature Review

The MsBOT project performed a systematic and thematic review of the existing literature on robotic systems, particularly educational robotic systems with auto-configurational Plug-and-Play (PnP) algorithms. This review investigates

- the definition, the technologies, and the algorithms in relation to auto-configuration and Plug-and-Play, relating to educational robotic systems.
- critical thematic attributes of sustainable educational robotic systems.
- theoretical and methodological traits of ROS-2 compatible DDS protocols and implementations for resource-constrained Microcontrollers.
- benefits, limitations, resolutions, and implementation of Controller Area Network (CAN) Bus for Data Distributed Services (DDS).

Eventually, it attempts to identify the gap and outline a research interest.

2.1 Introduction

Exploring the Existing Methodologies

This thesis performed a systematic and thematic review of the literature on the existing robotics technologies, including the robotics for education, particularly in relation to auto-configuration and plug-and-play (PnP) algorithms. It also reviews the common attributes of sustainable robotics, the technology supporting PnP development, and the issues and resolutions relating to the technology, e.g., Robotic Operating System (ROS) 2 architecture for resource constraint Microcontrollers. Instead of standalone robotic technology, this review focuses on technologies that accomplish distributed collaborative robotic systems, particularly open-source solutions.

Many robotic technologies are presented in the form of middleware for better incorporation of the robotic ontologies and convenient and friendly development platforms. Common Object Request Broker Architecture (CORBA), ROS2, Orca and Distributed Data Services (DDS), and many DDS variants are widely accepted open-source middleware for distributed robotics. Reviewing various DDS implementations, particularly OMG's Distributed Data Services (DDS) implementation in resource constraint devices, attempts to identify the gap and outline the research interest in this study.

2.2 Literature Research Methodology

Like other research activities, to achieve the objectives, the Systematic Literature Research (SLR) [36] is adopted to collect the strengths and weaknesses of the existing empirical evidence of Plug-and-Play (PnP) design and implementation in software development, particularly robotic systems. Besides, it reveals any gaps in the PnP research that potentially led to producing a framework for the new research activities.

Under the SLR guidelines [36], the methodology activities are broadly categorised into three phases: planning, processing and analysis, and synthesis [36]. Following the phases and stages flow diagram [37], the planning phase, research questions were established, and developed research protocols. The details will be presented below.

Many Academic Search Engines (ASE) were employed during the processing phase [36-38], such as Google Scholar, ResearchGate, Elsevier, ACM Digital Library (ADL), Google Scholar (GS), IEEE Explore (IEEE), Science Direct (SD), Scopus, Springer Link (SL) and Web of Knowledge (WOS). However, Google Scholar has been used the most for research literature, journals, and conferences. The search results were then filtered using the exclusion and inclusion criteria defined in the research protocols. The following items show the snapshot processes of the SLR for the PnP robotic research activities.

2.2.1 Objective

The primary research question (PRQ)

What are the most useful features of robotic systems for increasingly complex education?

2.2.2 Secondary Objectives

The PRQ further leads to the following secondary questions (SQ):

Q1. What are the contemporary educational robotic capabilities and technologies?

Q2. What are the PnP technologies of educational robotic systems?

Q3. What are the technologies of communication protocols in open-source robotic systems?

Q4. What are the essential attributes educational robotic systems must have to be sustainable?

2.2.3 Inclusion Criteria (IC)

Studies include the following elements:

• Works relate to the design and development of robotic systems for education.

• Works relate to designing and developing PnP features in software development, particularly for robotic systems.

• Studies on open-source robotic middlewares (e.g., ROS2), its application (e.g., DDS) on educational robots and its influence on the features of the educational robots.

• Studies on the communication protocols of open-source robotic middlewares, particularly CAN-bus.

2.2.4 Exclusion Criteria (EC)

Studies exclude the following characteristics:

• Works that do not use open-source robotic middlewares, particularly ROS-2 or compatible ROS2 construct.

• Works using outdated open source middlewares no longer have or lack community support.

• Reports that detail the technicality of standalone robotic system design and development.

• Conceptual design that does not have any implementation evidence.

2.2.5 Search and Selection Strategies of Primary Studies

For an SLR data collection and analysis, each paper of interest will apply the selection criteria that consist of the following three filtering stages:

Stage 1. Applying the EC by parsing the title, the abstract, and the articles' keywords.

Stage 2. Applying the EC by parsing the introduction and conclusion of the first filter.

Stage 3. Applying the EC with the last filtering step through reading the remaining papers.

2.2.6 Search String Sources

The search strings were established by logically grouping the keywords defined in the PQS, SQs, IC and EC. The keywords are Robotic Systems, Robots, Educational robots, Education,

Training, Teaching, Learning, ROS-2, Data Distributed Services (DDS), Auto-configuration, and Plug-and-Play (PnP).

2.2.6.1 Search String formation

Boolean operator ORs, ANDs, and parentheses form a search expression.

The first-string group makes up of two parts, i.e., (“open-source” AND Robot*) and (Education*), which is meant to return papers related to educational robotic literature. However, the (Education*) is intentional so that it can be discarded when in need.

The second-string group, i.e., (“Plug-and-Play” or “PnP” “auto-configuration” “self-configuration”) intends to slimline the search to papers that discuss the topic of interest.

The third-string group is (“Robotic middleware” “ROS-2” “Distributed Data Services” OR DDS), which restricts the search to papers discussing those topics.

The fourth string is (“Transport Protocols” AND (CAN OR “CAN-Bus”)) restricts the search to papers that discuss Transport protocols, in particular CAN-bus.

All these groups of strings were derived from the research questions and the inclusion criteria, while the results were parsed using the exclusion criteria.

By combining these search string groups, resulting in the research expressions are:

1. ("Open-source" AND Robot* AND Education*) AND ("Plug-and-Play" PnP "Autoconfiguration" "Self-configuration").
2. ("Open-source") AND ("robotic middleware" "ROS-2" or "Distributed Data Services" OR DDS) AND ("Transport Protocols" or CAN or "CAN-Bus").

The first search string expression intends to return papers discussing open-source robotics for education, which have involved Plug-and-Play or PnP or have features of auto-configuration or self-configuration.

The second search string expression intends to return papers discussing the open-source robotic middleware, which involves transport-protocols development.

2.2.7 Definition of Plug-and-Play

It is rational to begin reviewing the definition of PnP in various systems out there. The definition of PnP is not the same across the board, and each has its unique features and capabilities. The PnP will have different capabilities depending on the technology adopted and

the system's architecture. Nevertheless, the PnP feature is not strictly confined to robotic systems but is generally applicable to all computer systems.

To make a start writing a review of PnP, I suggest posing the following questions:

- What is PnP?
- What is Plug-and-Play for Robotic Systems?
- What are the essential attributes of PnP?
- Why does robotic PnP aspire?
- What technologies involve in PnP construction?

Look for one to two sources that answer each of these questions. Also, the topics don't necessarily, need to be limited to robots, e.g., look at how PnP has been used with PC peripherals and other devices. Avoid posing questions that result in yes or no answers - discussing the answers in the literature review will be hard.

2.2.7.1 What is Plug-and-Play (PnP)?

In recent years, emerging robot technologies have given rise to various robotic applications in our daily lives: educational robots [8, 38-41], home automation robots [42], medical robots [43], and many more. It fits the newly released concept of the Ubiquitous Robotic Companion(URC) [44], which provides ubiquitous services anytime and anywhere at users' convenience. However, these robots are not easy to fix when breaking down and maintaining when needing an upgrade, not to mention the rise of their numbers.

Unlike personal computers (PCs), robot component manufacturers are yet to have de-facto industry standards, like the add-on interface of PC, which simplify board replacement by plug and play without user intervention. Despite the drawback, robot component manufacturers have yet to adopt horizontal integration, which catalyses rapid, cost-effective mass-market robotic products and empowers consumers [45].

On the contrary, vertical integration dominates in robotic systems, featuring proprietary design, limiting competition amongst manufacturers, locking in customers' choices [10], which hinders the implementation of Plug-and-Play, interchangeable components across the board. The thesis investigates the existing robotic technologies and produces a framework manifesting PnP in educational robots.

2.2.7.2 What is Plug-and-Play for Robotic Systems?

According to the online Collin dictionary, Plug-and-Play, also called PnP, refers to computer hardware capable of self-detection and auto-configuration without user intervention. TechTerms [46] describes Plug-and-Play as the ability of peripherals to work with a computer system as soon as they are connected without user intervention in installing drivers. Instead, the computer can auto-detect the peripherals and subsequently auto-configure the computer systems to work with the newly connected device.

Robotic systems are predominantly heterogeneous systems comprising computer systems with different electro-mechanical parts built for specific tasks. Robotic systems are dominantly vertically integrated; however, there have been many research and ontologies for horizontally integrated systems. One manifestation of horizontally integrated systems is plug-and-play. Despite there being no integrative standard for robotic systems, Plug-and-Play refers to the ability of robotic systems and their components to integrate without needing direct user intervention. It entails those robotic systems can auto-detect the presence of the newly plugged-in components and then auto-configure through communication and negotiation with the robotic components to get the component up and running.

However, robotic systems are not districted to standalone systems. Distributed collaborative robotic systems are increasingly common due to the advancement of networking and computer technologies. According to [11], the Plug-and-Play is a communication layer framework that enables robotic hardware composition over networks. When plugging in new or removing old hardware, the robotic systems can auto-detect and auto-configure without users' intervention. In addition, being a distributed collaborative robotic system, it uses databases to keep track of new and old components over the network.

To exemplify the Plug-and-Play application, consider a robotic educational system where a teacher requires upgrading robotic components to improve effective learning. Instead of using the original components, alternative components with greater features are employed. Without much effort and help from the technicians, the teacher succeeded in deploying the components by connecting the components to the robotic system and letting the robotic system auto-detect and auto-configure the new components.

2.2.7.3 What are the essential attributes of PnP?

PnP is the ultimate transformative feature of vertical integration to horizontal integration that catalyses the de-facto integrative standard [13]. In addition, it needs to be sustainable for

system design, implementation, and reconfiguration [47, 48], which are the three major areas of research in component-based software engineering.

The research [49] shows that four attributes of Plug-and-Play are essential for transitioning to component-based software :

- Context independence. When software components are developed, they should not define how they are connected. In short, it should state the assumptions made about their context.
- Location transparency. When software components are developed, they should not define where, a specific location, or what configuration they are connected to. Components can be in different configurations or assigned to different process threads distributed over networks.
- Heterogeneous connection. The technical architectures of the run-time environments: programming languages, communication protocols, hardware platforms and operating systems demand the support of the composition of heterogeneous components.
- Evolvability. Components should evolve easily by addition, removal, or replacement after deployment.

2.2.7.4 Why does robotic PnP aspire?

In 1982, IBM released an open de facto standard architecture for PC transforming vertical integration into horizontal integration in the PC industry. However, the robotics industry of today is still in the era resembling the computer industry before the 1980s. Without an established integrative standard, despite the advancement of computer technology which produces smaller form factors and cost-effective products, robotic systems which take advantage of computer technologies are still prevailing incompatible, cost-ineffective, locked-in and limited choices [13].

In order to have sustainable systems that promote user empowerment, encourage component interoperability and easy maintenance and installation, Plug-and-Play is the ultimate feature, an instrumental in transforming the industry's structure [10, 13]. Besides catalysing de facto industry standards that enable a simplified and seamless integration and rapidly developing cost-effective mass-market products, it also empowers consumers, increasing robotic adoption in all domains of modern society [10, 13].

Attempts have been made to create integrative standards and international agreements [12]; despite several ontologies [13] in the robotics domain, the industry still has not accepted and adopted these models.

Amongst the Plug-and-Play standardisation initiatives, they are a few open-source frameworks relevant to the topic of discussion. The following sections provide a review of the frameworks.

2.2.8 Object Management Group (OMG)

By capitalising on the advancement of computer and networking technologies, robotic systems nowadays have become much more versatile, smaller form-factor, intelligent, and have evaded our daily life. Besides industrial applications, robotic systems are increasingly common in non-industrial applications, e.g., education, household, welfare, and commercial services, resulting in increased complexity and size in robotic software.

The Object Management Group (OMG) [15, 16], an international technology standard consortium [15], aims to develop technology standards for vertical industries [15]. Its members include end-users, vendors, government agencies, universities and research institutions, vertical markets [15] and well-established technological firms.

To be efficient and effective in rolling out the robotic initiatives and specifications, OMG had established many task groups. Amongst prominent groups are:

- Robotics Domain Task Force (DTF)

In December 2005, OMG established Robotics Domain Task Force (Robotics-DTF) to promote interoperability, compatibility, and reusability between the various robotic modular components by adopting OMG standards [14-16] at the distributed object layer. DTF develops a standard for dynamic deployment and configuration of robotic technology components that help to achieve the plug-and-play objective [10].

- Robotic Technology Component (RTC)

The robot technology function modules [16], also called robot modules, refers to an entity composed of software that realises a set of functionalities—for instance, an actuator module is an entity that operates the actuator hardware based on its input. Other entities are controller modules, sensor modules and many more. It is easy and effective to build robots by integrating these robot technology function modules. To ease the reusability and interoperability of the modules, OMG standardises the communication protocols between modules, namely the

distributed object layer. These robot modules are referred to as Robot Technology Components (RTCs). RTCs have the following characteristics:

- Every RTC has functional tasks with which other RTCs can work.
- Data and commands are exchanged amongst RTCs.
- A new RTC can be created from a combination of RTCs.

RTCs that include hardware devices are called Super Distributed Objects (SDO).
Many institutions, both private and public, support OMG initiatives, including the Japan Robot Association (JARA) and the (Japanese) National Institute of Advanced Industrial Science and Technology (AIST) [50]. In 2006, AIST released the Middleware Robotic Technology (RT) standardisation [50], and in 2008 released the OMG Robotic Technology Component Specification v1.0 [18], which defines a component model and the infrastructure services for robotics software development, which include Platform-Independent Model (PIM) and Platform-Specific Models (PSMs) expressed in Unified Modeling Language (UML) and OMG Interface Definition Language (IDL) [14] respectively. The platform-specific model defines direct component communication, while a distributed model defines *component communication* through CORBA as its communication platform.

2.2.9 Robot Operating System (ROS)

The Robot Operating System (ROS) [51, 52] is an open-source robotic framework that emerged in 2007 from Stanford research driven by Willow Garage. It is not an operating system, not as its name implies. However, a middleware suite consists of tools, libraries, services, and conventions to simplify the complex and large-scale development of robotic tasks. Furthermore, promising robust robot behaviour for a wide variety of computing platforms. ROS consists of three levels of concepts, namely the Filesystem level, the basic Computation Graph level, and the Community level. ROS Filesystem facilitates ROS resources management; ROS Computation Graph facilitates communication amongst peers and computational tasks; ROS Community manages ROS resources from external communities.

ROS has been reviewed as the most promising emerging standard in recent years. It is a UNIX-based system; however, ROS supports non-UNIX-based systems like Microsoft Windows and Real-Time Operating-systems. The other reasons are broader support by the communities, especially the academia.

ROS has recently upgraded to the next version, ROS2, which adopts new technologies such as OMG's DDS (Data Distribution Service), DDS-RPC, DDS-RTPS and many more defined by OMG as underlying communication standards. Some prominent features of ROS are:

a. A Peer-to-Peer Network Protocol

   The peer-to-peer network is the plumbing [52] of ROS processes. ROS processes are also known as nodes that are the centre of computational units. ROS is a natively distributed robotic framework that supports collaborative robotics that allows robotic components to operate through several processes in a node or different nodes connected in a peer-to-peer topology. The peer-to-peer communication system connects multiple nodes of heterogeneous computer clusters agnostically to form a collaborative system. ROS data concepts are represented in many forms, and each carries different and distinct terminology, such as nodes, Master, Parameter Server, messages, services, topics, and bags. *However*, in different forms, all these present data to the Computational Graph for processing. For example, the Master node facilitates lookup mechanisms [52] to register nodes, discover nodes and topics, and control parameter server updates [52].

b. Numerous yet Versatile Tools

   Tools are integral parts of ROS. It enriches ROS core functionality. It lets developers visualise and record data, traverse the entire ROS package structure at ease, and automate complex configuration and setup processes. Tools are organised in packages and verified as robot-agnostic to simplify installation and resolve development issues. The existing tools are **rviz**: a three-dimensional visualiser **rosbag**: a command-line tool for recording and playback ROS messages, **catkin** [52] is the ROS build system, **rosbash** [52]: a suite of tools that augment the functionality of the bash shell, **roslaunch** [52]: for launching multiple ROS nodes both locally and remotely and setting parameters on the ROS parameter server, and many more.

c. A multilingual system

   Multilingual is one of the ROS design philosophies [51] that emphasises supporting various programming languages to meet individual preferences. ROS is a language-neutral [51] which currently supports C++, Python, Octave, LISP, and many more language ports in pipelines. The ROS peer-to-peer communication uses

XML-RPC at the messaging layer, of which implementations exist in most major languages. To support multilingual in the peer-to-peer network, ROS messages use common yet simple text files to compose fields of each message. This very short text-based language-neutral messaging protocol is named Interface Definition Language (IDL) [14].

d. A Thin Architecture

ROS architecture is thin [52], with the least redundant drivers and algorithms for encouraging efficient code reuse. For that matter, achieving the "thin" ideology, ROS encourages all driver and algorithm development to use of standalone libraries with no dependencies on ROS. It emphasises the complexity hiding in libraries and encourages a simple library interface that allows users to use and reuse efficiently. In addition, complying with the "thin" and multilingual philosophies, ROS uses CMake [53], a cross-platform build system generator, to generate a build system description for each directory of a source tree and is natively compatible with the development platform.

e. Free and Open-Source

Open-source software is well-known for having many benefits. Open-source software is not free; it is affordable for many. High affordability attracts academia, enthusiasts, and hobbyists to learn and contribute to the development and debugging at all levels of the software in a parallel manner.

ROS is licensed under the standard three-clause BSD license, a permissive open license that allows for reuse in commercial and closed-source products. The modular nature of ROS allows module owners to use fine-grain licensing policies to incorporate software protection of licenses ranging from GPL to BSD, despite being proprietary.

Ever since the existence of ROS, it has inspired much development of ROS related new technologies and improvement. The prominent ROS derivatives are reviewed in the following sections.

2.2.10 New Technologies - ROS Derivatives

2.2.10.1 Robot Operating System 2 (ROS2)

ROS2 [8] is next-generation ROS derived from ROS. At its inception, ROS's primary goal was to improve performance and provide the software tools for the PR2 development project. It has been a very successful project adopted by a wide variety of robots besides PR2. However, ROS has overgrown, and the increasing complexity has stretched ROS beyond its initial capacity. A new ROS version is sought to address new issues that ROS does not while leveraging its greatness [54]. The new version of ROS should be intrinsically capable of

- Building teams of multiple robots.
- Handling resource constraint embedded platforms.
- Supporting real-time systems.
- Handling poor connectivity conditions.
- Supporting migration of ROS lab-based and researching products to the real world.
- Supplying clear patterns and supporting tools for life cycle management and static configurations for deployment.
- Supporting new APIs that incorporate the community's collective experience learned from the first-generation APIs without losing the key concepts of ROS.
- Taking advantage of ROS-based new technologies such as Zeroconf, Protocol Buffers, ZeroMQ, WebSockets, and DDS (Data Distribution Service) for their code reusability, expertise, full documentation, and high-quality and production-ready.

2.2.10.2 Data Distribution Service (DDS)

Many ROS-based new technologies have emerged since its inception in the year 2007. DDS is one of the most prominent software technologies which complements what ROS is not. The new DDS publish-subscribe messaging protocol was aspired to address issues such as real-time system, high-performance, dependability, interoperable, Quality of Service (QoS), security, resource-constrained devices, scalable data exchanges, machine-to-machine communication and many more. DDS is transport-independent: It supports UDP/IP, TCP/IP, Shared-Memory or other transports [55].

Besides being transport independent [55], DDS's communications are decoupled [21], where DDS transparently handles messages delivery without knowing who and where its recipients are and what happens when messages are undelivered.

The DDS is a machine-to-machine networking middleware for real-time distributed communication systems [21], a next-generation communication standard adopted in ROS2 [54]. In order to optimise data delivery and ease integration to applications, the DDS specification specifies the adoption of a lower data-centric publish-subscribe (DCPS) and an optional higher data local reconstruction layer (DLRL) [27], respectively.

Unlike ROS's publish-subscribe transport, DDS provides publish-subscribe transport of real-time capability with enhanced flexibility, scalability and interoperability [21]. In addition, with the Quality of Service (QoS) facilities in its data distribution, DDS can control its data availability, resource usage, reliability, and timing [19]. Its messaging system uses the text-based OMG's IDL for message composition and serialisation DDS discovery is a process of finding the participants and endpoints of the DDS network. DDS relies on a multicast mechanism in discovering DDS participants [21]. DDS is a distributed discovery system [19] that allows two DDS programs to communicate without needing a medium such as a ROS master, making the system more fault-tolerant and flexible. Despite many similarities with CORBA in its distributed services, they are two independent standards; DDS is not a replacement for CORBA [23, 55].

2.2.10.2.1 Remote Procedural Calls (RPC) over DDS (RPC-DDS)

Multiple middlewares are required for implementing different communication styles in a large distributed system because one middleware does not satisfy all the requirements [21, 27, 55]. Despite being expensive, it is not an uncommon approach compared to a single middleware construct. As a resolution for being cost-effective, least maintenance overhead and clean, Remote Procedural Calls(RPC) over DDS, abbreviated as DDS_RPC, was aspired to provide request/replay semantics [56]. RPC-DDS simplify complex networking programming [27, 57]. In addition, RPC is an evolvable type system, QoS, and fine-grained security [55], which makes RPC superior for its scalable data exchange communication. DDS-RPC make use of the basic building blocks of DDS.

2.2.11 DDS Implementations

The Object Management Group (OMG) [15, 16], an international technology standard consortium [15] where the DDS originated, has inspired many organisations to implement middleware in accordance with OMG's DDS specifications. The products of the development

are called DDS implementation. The following reviews some of those which are prominent and have gained popularity in the industry.

2.2.11.1 RTi Connext DDS

Real-Time Innovations (RTI) [58] is the largest software framework company for autonomous systems. RTi Connext DDS [58] is an implementation based on OMG DDS specifications. RTi DDS is proud to be the first software framework designed to address companies building remotely operated autonomous systems. It enhances the development of real-time systems from disparate network types. Despite being a commercial DDS implementation, it supports many platforms that target Linux, Windows or macOS. Due to its size, it does not fit well in embedded platforms. However, RTI offers commercially available Micro-DDS, which is meant for Microcontrollers.

2.2.11.2 eProsima Fast DDS

eProsima Fast DDS [28, 29, 59] is striking fast, outperforming ZeroMQ and other DDS middleware solutions in Windows and Linux. Taking advantage of being component-based software, the middleware facilitates Publish/Subscribe code using an Interface Definition Language (IDL). A useful tool allowing the developer to focus on his application logic without bothering about the networking details.

2.2.12 Resolution for Miscommunication Among Different DDS Vendors

Like any other protocol, several DDS implementations exist from different vendors, despite only one DDS standard specification [14]. The differences among DDS implementations may create miscommunication among vendors.

2.2.12.1 Interface Data Language

A resolution for bridging the differences is by implementing a middle layer in between different programming languages and translating the received messages to the format that is correctly formatted for the wanted programming language. The resolution is achieved through the IDL language, which translates the general data types between different programming languages.

In the DDS applications, a message is described using IDL. With mapping platform-specific to IDL of DDS, regarded as an application portability [14], an application can easily switch to other DDS implementations by recompilation.

2.2.12.2 Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSi-RTPS)

At runtime, differences in different DDS implementations present another challenge for the community. DDS implementations cannot exchange messages over transports such as TCP/UDP/IP [14] unless the vendors set up a specific ‘‘bridge” [14]. For DDS implementations to be interoperable, DDS needs to standardise communication protocols among DDS implementations.

DDS is increasingly prevalent in large distributed systems, which need interoperability amongst DDS implementations from multiple vendors. OMG standardises communication protocol, namely DDS Interoperability Wire Protocol, also regarded as DDS Interoperability-Real-Time Publish-Subscribe Protocol(DDSI-RTPS) [14]. DDSI-RTPS has aspired, as outlined in the design philosophies, to achieve high performance and QoS-enabled, fault tolerance, extensibility, plug-and-play enabled, load-balance enabled, modularity, scalability, and type-safety [14].

In March 2009, based on DDSI-RTPS version 2.1, three vendors demonstrated interoperability by discovering each other’s publishers and subscribers despite different OS Platforms and different network communications [27]. Besides achieving the design philosophies, DDSi-RTPS can handle multiple topics and instances of topics, multicast and UDP/IP connectivity, delay-tolerance, content-filtering, and many more [14, 27].

2.2.13 DDSi-RTPS variants

Like DDS implementation, the DDSi-RTPS messaging protocol specifications have been implemented by many vendors. Different vendors design and implement DDSi-RTPS to meet their needs resolving various issues to it system-related or social-related. In the following sections, it provides a review of a few prominent DDSi-RTPS variants in the community.

2.2.13.1 embeddedRTPS [60]

embeddedRTPS is a portable, open-source implementation of the Real-Time Publish-Subscribe Protocol (RTPS) for Ethernet-capable embedded devices. Despite targeting resource-constrained embedded platforms, it should not be confused with the DDS-XRCE protocol implementation; rather, embeddedRTPS is an independent first-class DDS participant without needing an Agent. For being portable, it employs the strategies of using lightweight FreeRTOS [61] and lwIP [62], discarding dynamic memory allocation after establishing endpoints, implementing rudimentary QoS policies and employing the static buffer eProsima Micro-CDR [63]. Common Data Representation (CDR) [64] serialises primitive and complex types. CDR [64] is part of OMG DDS implementations, which ROS2 adopts. It comes with many features which benefit the resource constraint Microcontrollers. There are:

- Discovery - Simple Participant Discovery Protocol (SPDP), as well as Simple Endpoint Discovery Protocol (SEDP), provide the participant with embeddedRTPS to discover other participants and endpoints in the network.
- Interoperability – It has successfully tested the interoperability with Fast DDS 2.3.1.
- QoS Policies – it supports both reliable as well as best-effort endpoints.
- UDP Multicast – With the support of multicast locators. It is useful for a large network with many participants, where the server only needs to send each message once and will reach all the participants.
- Message Size – the buffer size of the lwIP controller of the system determines the message size. In addition, messages are not split up among multiple buffers.

2.2.13.2 eProsima FastRTPS [59]

eProsima FastRTPS is a DDS-RTPS implementation, a standalone C-PlusPlus (C++) middleware of ROS 2. It employs IDL to generate the Publish/Subscribe code, removing developers from the daunting networking details. eProsima Fast DDS is open-source software released under Apache License 2.0. FastRTPS has many advantages:

- High Performance - The performance benchmarks show that the FastRTPS is low latency and high throughput despite being in low bandwidth environments.

- Easy Multi-Platform Integration - It has been ported not only to Windows, Linux and MacOS but also to many real-time OSes like QNX, VxWorks, iOS, Android, and Raspbian/Debian Buster.
- Free and Open Source is licensed under Apache License 2.0, where the party of interest can use the source code or software for any purpose. To distribute it, to modify it, and to distribute modified versions of the software under the terms of the license without concern for royalties.
- Standard-based and Interoperable DDS compliance: OMG DDS 1.4 Compliant, which ensures efficient and robust delivery of the right information to the right place at the right time [65].
- Full RTPS compliance: OMG RTPS 2.2 Compliant, which ensures that applications based on different vendors' implementations of DDS can interoperate.
- Personalised Services & Commercial Support – eProsima is a networking middleware expert committed to providing personalised, high-quality development assessments and service support to fulfil customer needs.
- Well-documented & easy to use: Full documentations are available online. Users can download it for free. Most importantly, the documents are well-structured and easy to read.
- Well suited for robotics, IoT, automotive, and critical applications - The performance benchmarks show that the FastRTPS is low latency and high throughput despite being in low bandwidth environments, which are the right ingredients for life-critical robotic applications and equally suitable for IoT and automation that require the real-time ability to boost their product performance.

2.2.13.3 FreeRTPS [60, 66]

FreeRTPS is a free, portable, minimalist RTPS implementation. It was intended to provide an alternative option for deeply embedded ROS2 applications where RAM/ROM size is a critical factor. Despite being extremely experimental, it aims to facilitate embedded devices where resources are critical to partake in DDS distributed systems under the ROS 2 framework. FreeRTPS is designed for portability; hence it does not have any runtime dependencies. FreeRTPS can be used on itself without the ROS2 abstraction layers. The only way to compile and run the system on a Microcontroller (MCU) is with a cross-compiler in a Linux terminal.

It can generate images for the STM32F4 and STM32F7 “discovery” evaluation boards. However, only a minimal number of Microcontrollers it supports for time being. At the time this is written, STM32 MCUs are supported.

2.2.14 DDS for eXtremely Resource Constrained Environments (DDS-XRCE)

With the increasing advancement of computer and networking technology, Microcontrollers are getting smaller, but processing power is getting higher, and features are getting richer, e.g., Ethernet network-capable is not uncommon. Adopting single-board computers (SBCs), made of Microcontrollers, is increasingly common in robotic systems. However, DDS’s processing footprint is oversize Microcontrollers could. In order to overcome the limitation, OMG defines the XRCE protocol [27] to extend DDS to resource-limited devices for communicating with larger systems [28, 29]. The XRCE protocol is for many resource-constrained devices (clients) and an XRCE Agent (server) [14]. The client-server architecture splits the entire, massive DDS processing load between resource-limited devices(clients) and the XRCE Agent(server), which acts on behalf of its clients in the DDS Global-Data-Space [29].

The DDS-XRCE protocol is defined as a set of logical messages exchanged between the XRCE Client and the DDS-XRCE Agent. The resource-constrained clients make publishing and subscribing to topics via an intermediate service handled by an agent in a DDS domain [18, 29], resulting in the resource-constrained devices accessing the DDS Global Data Space [14].

2.2.15 eXtremely Resource Constrained Environments (XRCE) Variants

Like DDS, DDS-XRCE specifications exist in many different implementations by different vendors.

2.2.15.1 eProsima Micro XRCE-DDS

eProsima Micro XRCE-DDS [28, 29] is an open-source middleware product that implements the OMG DDS-XRCE. eProsima Micro XRCE-DDS is the default middleware of Micro-ROS [30-32], the official extension of ROS 2 for Microcontrollers (MCUs).

eProsima Micro XRCE-DDS is a software solution that enables eXtremely Resource Constrained Environments (XRCEs) devices to act as DDS entities by allowing them to

communicate with an existing DDS network. In addition, eProsima Micro XRCE-DDS implementation complies with the Object Management Group (OMG) consortium XRCE-DDS specifications.

For the resource-constrained devices (clients) to communicate with the existing DDS Global Data Space, the eProsima Micro XRCE-DDS library facilitates a client-server protocol which makes use of a PC or higher performance computer system to act as an agent, also called eProsima Micro XRCE-DDS Agent (server). Through the agent (server), which acts as a bridge, the resource-constrained devices (clients) access the DDS Global Space. The agent acts on behalf of the Micro XRCE-DDS Clients by enabling them to participate in the DDS Global Data Space. In addition, the Micro XRCE-DDS library facilitates Fast RTPS protocols allowing the XRCE clients to communicate with other DDS entities as DDS publishers and/or subscribers. Besides, the eProsima Micro XRCE-DDS also support Remote Procedure Calls, as defined by the DDS-RPC standard, which implements a request/reply communication pattern. To ease the development of eProsima Micro XRCE-DDS Clients, eProsima Micro XRCE-DDS Agent is set to operate by auto-detection and auto-configuration Plug-and-Play system. It comes with many features. Some prominent are:

- High performance – Serialisation and deserialisation are integral operations in eProsima Micro XRCE- DDS Client, which uses a static low-level serialisation library, namely eProsima Micro CDR that serialises in XCDR.
- Low resources - The Client library does not consume any dynamic or static memory. In contrast, its memory footprint is due to the stack growth. However, the related publisher/subscriber operate use less than 2 kB of RAM. Besides, the Client functionalities are customizable in accordance with the user's profiles concept, resulting in changes in the binary size.
- Multi-platform – The Client library supports both the standard Operating Systems Linux and Windows and the Real-Time Operating Systems Nuttx, FreeRTOS and Zephyr.
- Pluggable modules - users can easily implement their platform-specific modules for the eProsima Micro XRCE-DDS Client library.
- Compiler dependencies free – The Client library uses no C compiler extensions but pure C99 standard instead.

- Free and Open Source – The eProsima Micro XRCE-DDS in general and the Client library, the Agent executable, the generator tool, Micro CDR, and eProsima Fast DDS are all free and open-source.
- Easy to use – Easy to use and full documentation, step-by-step guides with several examples, namely, how to create a publisher/subscriber, a requester/replier and a Peer-to-Peer publisher/subscriber Client. In addition, interactive real-life demos are available Online, e.g., the ShapesDemo application. Through that, users understand the DDS-XRCE protocol and the Client API.
- OMG XRCE-DDS compliant – It implements the standard communication protocol defined by the OMG consortium, which allows eXtremely Resource-Constrained Environments devices to communicate with the DDS world.
- QoS - eProsima Micro XRCE-DDS supports both best-effort and reliable stream of communication. The former focuses on providing fast and light communication, while the latter emphasises a reliable independent transport layer.
- Customed transport layer – Besides the existing Serial and TCP/UPD transport protocol, eProsima Micro XRCE-DDS support customed transport protocols. It is designed to be transport agnostic, empowering users to develop customed transport easily.
- Commercial support – Despite it is free and open-source, eProsima provides consultant services at commercial rates.

2.2.15.2 RTi Connect Nano

Like eProsima Micro XRCE_DDS, RTI Connext Nano [67] is an implementation of the OMG DDS-XRCE released under the open-source Apache 2.0 license. It supports Micro-ROS [30-32], the official extension of ROS 2 for Microcontrollers (MCUs). Compared with the eProsima Micro XRCE, the client supports more transport technologies such as serial links, Bluetooth, cellular networks and TCP/IP. The agent supports many platforms which are supported by RTI Connext DDS, but it needs to pair with RTI Connext DDS [58].

Like eProsima Micro XRCE-DDS, RTI Connext Nano [67] adopts a client/server protocol that offloads all management of DDS entities from applications to an external Agent process. User applications of the clients connect to the XRCE Agent. RTI Connext Nano client library provides the Client APIs to:

- Create and configure DDS entities on the Agent.
- Write DDS samples using a DataWriter on the Agent.
- Read DDS samples received by a DataReader on the Agent.

In addition, RTI Connext Nano comes with many features:

- Reduced size transport - The XRCE-DDS protocol works on any transport that has a Maximum Transmission Unit (MTU) of at least 24 bytes.
- QoS - The XRCE-DDS support reliable stream communication, which allows a reliable protocol for the reliable delivery of messages over unreliable transports.
- fragmentation – The XRCE-DDS XRCE supports fragmentation and reconstruction of large data payloads, allowing XRCE to send data larger than the transport's MTU.

2.2.15.3 Micro-ROS [30-32]

With the advancement of networking technologies, robots are no longer standalone systems but a composition of many components distributed over various spaces connected through different data links. ROS 2 supports higher processing power and larger memory space, such as PCs. Despite today Microcontrollers are equipped with high computational power.

eProsima Micro XRCE-DDS [28, 29] is an open-source middleware product that implements the OMG DDS-XRCE. eProsima Micro XRCE-DDS is the default middleware of Micro-ROS [30-32], the official extension of ROS 2 for Microcontrollers (MCUs). For details, please refer to section 1.2.15.1.

2.3 Non-Distributed Data Service (DDS) Robotic Systems

By far, the focus has been on Distributed Data Service (DDS) robotic systems. However, many other well-accepted versatile robotic systems do not use DDS as their transport protocols despite the systems being capable of distributed components. The following sections provide a review of non-DDS robotic systems.

2.3.1 Orca [24, 25, 68]

The closest acronym of Orca is cross-platform interfaces and data objects. Orca is an open-source framework for developing component-based robotic systems. It facilitates the definition

and development of the building blocks which can be pieced together to form arbitrarily complex robotic systems.

Like other machinery software, Robotic software is massively large and complex due to its conformance to the timeliness, precision, health and safety requirements, and many operational standards. Robotic software development is often complicated and time-consuming, which hinders the progress of robotics. A common resolution is code reuse [69] because it quickly integrates functionality under limited resources in terms of time, skills, and cost. Orca is characterised by being:

a. Open-source framework – *Orca* [24], an open-source framework, aims to promote efficient code reuse in writing robotic software. Orca's open-source project is hosted on SourceForge.net and is distributed under a combination of LGPL and GPL licenses. The software units, i.e., components, including their repositories, can be independently distributed, deployed, and configured.
b. Component-based Design - [26] compared and contrasted a software object and software component by describing an object as a unit of instantiation while a component is a unit of independent deployment and has no persistent state.

A *software component* consists of two broad elements: an executable unit of code and a set of services accessed, by other components, through specified interfaces. According to [70], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only [70]. A software component [70] can be deployed independently and is subject to third-party composition. To improve clarity when explaining software components, the term software unit, which refers to an executable unit of code, is being used.

Orca applies Component-Based Software Engineering (CBSE) [26] principles, promotes code reusability, emphasises low coupling, high cohesion, and the balance of number and size (i.e., granularity [24]) of components are important in robotic system design. The component-based design encourages users to define and develop quality software units [26] of the least redundant that can work together cohesively to form arbitrarily complex robotic systems.

To promote efficient reusability, Orca adheres to, also guided by CBSE, the three following principles:

a. Arbitrary Component Granularity - Component granularity refers to the number and size of the software unit [26]. A monolithic software unit comprises numerous functionalities, which are resources and operation efficient but impractical for

interfacing with other modules to access its functionalities. On the contrary, the CBSE approach defines and develops software units of low-coupling [26], high-cohesion [26] and a well-balanced number and size of components, which promote code reuse. Despite being guided by CBSE, Orca allows developers, for efficient reusability, to conceal the detailed elements within a single component [24].

b. Agnostic Architecture - Orca is a component-based robotic software model which emphasizes code reuse through software components. Being a software component that inherits the behaviours of low-coupled and high-cohesion, Orca does not impose system architecture. Instead, Orca allows system developers to compose a system from any components arranged to form arbitrary architectures. Not tightly coupled with any system architectures, Orca is agnostic of programming language and operating systems. To date, Linux, Windows and macOS support Orca. For instance, two Orca components of different programming languages executing on different operating systems communicate through libOrcaIce, an ICE facility distributed with Orca.

   Orca was implemented using Common Object Request Broker Architecture (CORBA). However, to mitigate the complexity faced with CORBA [23], it has migrated to Internet Communications Engine (ICE) [71]. ICE is a transport layer protocol for managing communication amongst components, setting up a registry for service discovery, and managing components.

c. Arbitrary Interfaces and Component Architecture-less - Orca imposes no rules on defining and implementing the interface of components. Component developers are free to implement components and define the new interfaces [70].

2.3.2 Miro [72, 73]

Middleware for mobile robots (Miro). a CORBA-based robot middleware leverages modern software engineering technologies, which fosters rapid development of reliable and safe software that supports heterogeneous computer networks and multiple programming languages [72]. It is an object-oriented robot middleware which eases the development of mobile robots and fosters the portability and maintainability of the robotic software [73]. Miro is constructed using three layers: the device, the service, and the class framework.

- The device layer

  This layer is platform-dependent and matches layer-2 of the OSI model [74]. It provides object-oriented interface abstraction for all hardware devices.

- The service layer

  It provides device abstraction services via CORBA interface definition language (IDL). In addition, its CORBA-based adaptive communication environment (ACE) [75] provides object-oriented abstraction layers for many operating systems and communication primitives, and the CORBA Notification Services [22] is used for providing event-based communication functionality.

- The class framework

  It facilitates and eases the development of multi-platform robotic applications. It is a toolbox that consists of several often-needed services such as mapping, self-localization, behaviour generation, path planning, logging, and visualisation facilities. It uses multiplatform libraries to ease portability amongst different platforms and to support heterogeneous networks. Besides being an object-oriented system, it also adopts the layered architecture making it very flexible and expandable to new devices and new services for new robot applications [76].

2.3.3 Universal Plug-and-Play Middleware(UPnP) [17, 77]

Unlike CORBA, which was developed for distributed environments, UPnP middleware, developed based on UPnP technologies, is meant for ubiquitous robot control and simplifying network implementation in the home and corporate environments. UPnP is a technology for implementing dynamic ubiquitous service robots, which shares the same concept of the Ubiquitous Robotic Companion (URC) [77] that provides services at any time and any place in ubiquitous computing environments.

It features dynamic robot software integrations and pervasive peer-to-peer network connectivity PCs, wireless devices, and intelligent appliances [78]. By the dynamic characteristics, it allows ubiquitous computing devices to connect and disconnect seamlessly. Evidently, [77] argues that UPnP architecture is appropriate for both internal integration and external control of the robot in ubiquitous computing environments. The Open Connectivity Foundation (OCF) [78] and its predecessor UPnP Forum are tasked to manage and disseminate UPnP device control protocols released under open-source licenses.

The UPnP, a distributed, open networking architecture, leverages TCP/IP and Web technologies. Not only it enables seamless proximity networking, but it also enables control and transfer of data among networked devices [78]. UPnP technology has been adopted as the base protocol for device discovery and control for the home networked device interoperability architecture of the Digital Living Network Alliance (DLNA) [79].

The Three Components [77]

The UPnP architecture comprises three board components, namely Devices, Services and Control Points [77].

- The Devices - is an entity that provides Services, which can have zero or many Services. Some Services are attached to the Device. A Device can be nested.
- The Services – is a unit functionality associated with a Device. A Service exposes actions and state conditions for other Devices to access.
- The Control Point – is a service requester which can discover and control other devices.

Please see Figure 1 for the basic communication and relationship among the three components.

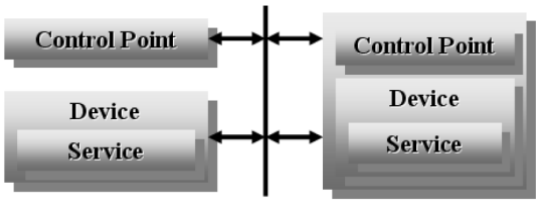

Figure 1 The relation and the communication amongst the three UPnP components [77].

The Protocol Stack [77]

The protocol stack of UPnP adopts the layered architecture in which each layer is tasked with specific functions to describe devices and the services it provides. Besides normal network connectivity, it also provides connectivity to ad-hoc and unmanaged networks. The bottom three layers are adopted to TCP/IP stack. While the top three layers, namely UPnP vendors, UPnP Forum Working Committees and the UPnP Device Architecture, define the highest layer protocols used to implement UPnP. Depending on the device architecture, the working committees define information globally to specific device types. Figure 2 shows the architecture of the UPnP protocol stack.

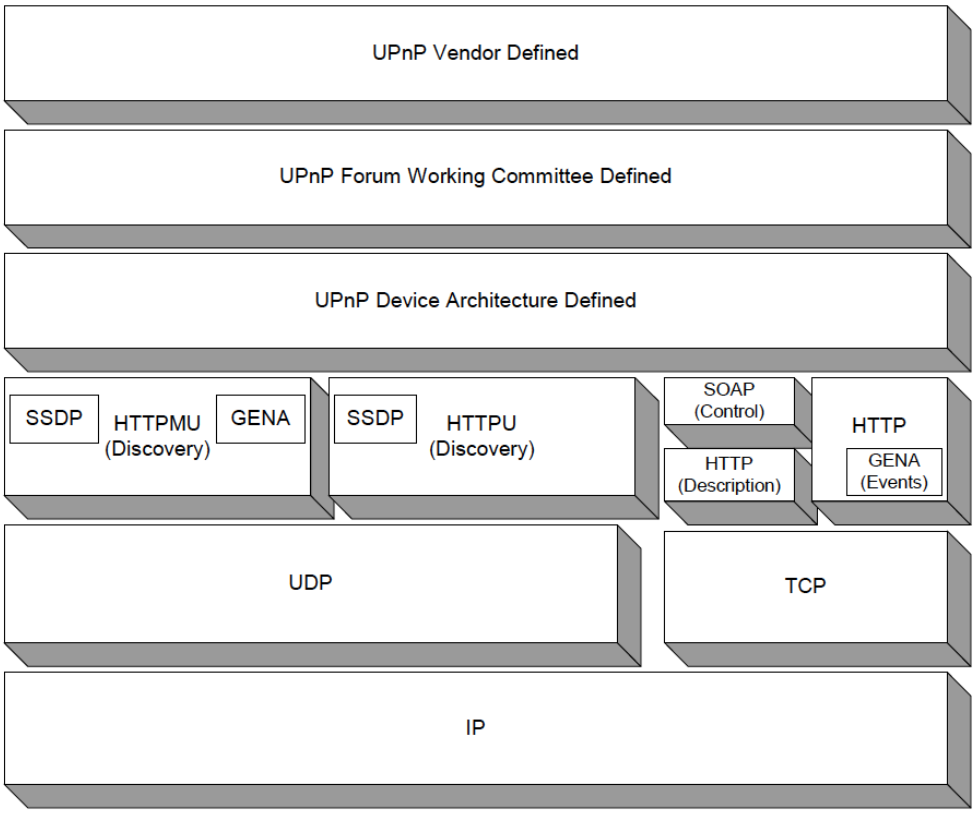

Figure 2 The architecture of the protocol stack of the UPnP [77].

UPnP Networking [77]

For Control Points and Devices to communicate with one another, the UPnP provides support for establishing networking operations that are based on open standard, Internet-based protocols. The UPnP networking is accomplished through 6 steps: Addressing, Discovery, Description, Control, Eventing, and Presentation. Table 1 explains the 6 steps of UPnP networking.

Table 1 Six steps of establishing a UPnP network

Steps	**Description**
Addressing	Control point and device get addresses
Discovery	Control point finds interesting device
Description	Control point leans about device capabilities
Control	Control point invokes action on device
Eventing	Control point listens to state changes of device
Presentation	Control point controls device or views device status using HTML UI

Scope of UPnP Technology

UPnP technology is coherent with the concept URC, of which robots provide ubiquitous robotic services 24 hours round the clock at users' convenience of home, small business, and proximity networks. Any devices on the UPnP-compliant networks can initiate communication with another device on the network. The great advantages of UPnP technology are being independent of any particular operating system, programming language, or network technology [78]. In addition, the great features of UPnP are Zero Configuration and Automatic Discovery where a device can [77]:

- Dynamically join and leave a network
- Support DHCP IP address allocation
- Device name service
- Request and response protocols for device identification and capabilities information
- Auto-discovery of the presence and capabilities of other devices
- Robust auto-configuration of a network seamlessly without leaving any unwanted state information behind
- Optionally support many TCP/IP compliance protocols such as DHCP and DNS servers.

Technology Benefits

UPnP inherits many benefits from its technology.

- Network media independence. It seamlessly works with any network technology: Ethernet, Wi-Fi, coax and many more.
- Platform independence. It is very interoperable amongst all the currently available operating systems.
- Programming language agnostic and programmatic control. It supports any programming language and conventional application programmatic control.
- TCP/IP compliance. It supports IP, TCP, UDP, HTTP, and XML, many more.
- Vendor-Specific UI Control. Vendors decide the device user interface (UI) and user experience (UX).
- Common base protocols. Vendors agree on common base protocol sets on a per-device basis.

- Vendor-customized Control. Vendors can value-add services layered on top of the basic device architecture.

2.4 Robotic technologies and PnP construction

Plug-and-Play emerges to facilitate easy and flexible [49] system maintenance: adding, replacing, upgrading and removing old components with the least effort. With upgrading a new version of components, a system improves its functionality; Plug-and-Play allows it to evolve to another system because of its easy replacement with variant-compatible components [49]. The facility applies to software and hardware systems alike.

2.4.1 What technologies involve in PnP construction?

Robotic systems are predominantly information technologies of software constructs and ideas extended to the physical domain [10]. For many years software engineering has been researching higher quality and performance and lower costs plug-and-play resembling the PC style of software construction [49]. Despite a sounded simplistic feature, the PnP facility entails complex, enormous, evolutionary software ideas and technologies, which require many software modules. Research of PnP has shifted from architectural-driven toward component-based systems [49].

Component-Based Software Engineering (CBSE) [26] is a research project that aims to develop technologies enabling software plug-and-play. Although it is principally a component-based system, it still manifests architecture-driven [49] software principles, such as layering, modularizations, abstraction, and information hiding, emphasizing programming-in-the-large and developing distributed systems [49].

2.4.2 PnP construction - A component model

As software is increasingly large and complex, it needs a pragmatic approach to manage complexity. The component model aims to manage and resolve complexity, address all architectural requirements, and enable developers to understand the big picture of the system design.

According to [49], the component model is a set of modelling concepts, semantics and rules for constructing a system architecture abstraction. It is achieved using the Architecture Specification Language (ASL), which the CBSE project developed, to state component-based architecture specifications. A component-based software model comprises four integral attributes, which are essential for manifesting PnP. The attributes are components, component interfaces, bindings, and configurations.

- Components

  A component is a functional unit of a system, which can be anything from high-level applications to low-level technical tasks. It is also a reusable and sharable unit to which other components connect. Components connect to form composite components. However, a component can be either atomic or composite. Therefore, a complete system is a hierarchical organisation of atomic and composite components.

- Component interfaces

  A component functionality defines its interfaces. Therefore, a component of multiple functionalities will have multiple interfaces. Literally, provided interfaces expose components' services or capabilities, while required interfaces specify those services that a component expects from other components. It is the required interface that software plug-and-play instruments invocation of a component to other components. In a client-server analogy, a server is a component with a provided interface, and a client is a required interface. Components are not fixed or “hard-wired” to one another; instead, a component of required services can attach any components that offer the matching services.

- Bindings

  Components connect to form composite components. Through binding, a required-interface component connects to another provided-interface component. In the modelling of large software systems, it needs to define component bindings explicitly. A complete system is a hierarchical organisation of components. Conversely, a component composition represents the architecture of a system.

- Configurations

  Despite components combined to form composite components to realise functionalities for resolving issues, at the application level, these components need configuration. Besides, low-level component implementations depend on local development properties and execution environment at run-time, collectively named component configuration

properties. The configuration refers to facilitating the information enabling integration of heterogeneous component instances into an executable distributed system.

2.5 Component-based System Key Concepts

Despite being a component-based system, it still goes through the life-cycle of normal software: design, implementation, and deployment [47], which software engineering disciplines have respectively identified and dealt with these activities: software architecture, configuration management, and configurable distributed systems. The software architectures outline the high-level design, system blueprint [49] and components composition [47, 49]. While configuration management handles implementing a software system, the configurable distributed systems manage released systems in the fields [47].

2.5.1 Middleware

The increasingly large and complex software systems demand reusable and flexible software architecture to leverage the quality, time-to-market, productivity, adaptivity, and sustainability of systems. By comparison, despite modular design can achieve the above requirement; however, it has integration issues [80] such as communication, interoperability, and configuration, resulting in the adoption of component-based architecture [49] emphasising low coupling, high cohesion, and granularity as the ultimate solution. Component models differ from object models as it resolves integration issues such as communication, interoperability, and configuration [49]. For many years, there have existed many ontologies for building horizontally integrated robots. However, none of the ontologies has been adopted as the common de facto standard by robotic manufacturers.

In order to promote the adoption of the standards and foster a component-based model, a middle layer [80] is devised to resolve the issues. The middle layer is also called middleware [80], a layer of software that resides in-between application software and system software and other proprietary properties relating to the local system. For instance, object middleware for robotic provides the technology component-based architectures and robotic services, libraries, and tools that foster robotic ontologies and component-based software development and connect various software components of different robotic manufacturers. The middleware

services offer many possibilities and enhance robotic applications development, which operating systems could not. Besides, middleware aims to accomplish [80]:

- Simplified development processes - providing higher-level abstractions and simplified interfaces, which ease software integration and reuse,
- Easy communication and interoperable - providing a mechanism to ease interoperability and communication,
- Efficient resource utilisation- providing high-performance and responsive processing power through efficiently utilising robot components and resources,
- Supporting heterogeneity abstraction - acting as a collaboration software layer, which hides the low-level details enabling communication amongst heterogeneity of the modules.
- Supporting integration with other systems - providing abstraction and real-time interaction services, enabling new robot types to integrate with other systems.
- Efficient reuse of often-needed robot services - to avoid redundancy and save time and resources, providing services to reuse to the commonly used robot services,
- Providing self-adaptive and automatic recourse discovery and self-configuration - Providing self-adaptability, self-configuration, self-optimization and automatic and dynamic resource discovery, preparing for unexpected situations.
- Supporting embedded low-resource devices -Extending services to resource-constrained devices.

Due to the intrinsic nature and benefits of middleware, middleware is increasingly popular among robotic manufacturers and researchers, especially distributed robotic systems. Not only does it ease development for distributed robotic systems, but it also serves as a potential candidate to integrate robot software. To date, many researchers are working on middleware for robotic development, both open-source and commercial, using different technologies and ideas, and implementing different ontologies and specifications. Like many software implementations, despite the above challenges being universal for many middlewares; however, the implementation approaches vary. However, there are a few middlewares that are worth discussing in this review.

The Object Management Group (OMG) has produced a series of distributed robotic specifications; each serves a different purpose; some have been implemented in middleware, others are still in conceptual models and development frameworks.

2.5.2 Common Object Request Broker Architecture (CORBA) [23]

CORBA, a middleware adopted in the robotic field [23]. CORBA's existence can be traced back to early 1991 when the first specifications were released [23]. It is one of the earliest distributed technologies middleware that underpins component-based system architectures [49]. CORBA facilitates communications between computer systems of different operating systems and different development, implementation, and software architectures. CORBA employs Interface Definition Language (IDL) to specify the object interfaces to interface with other objects. Through an Object Request Broker (ORB), objects send requests to and receive responses from other objects on the same system or across networks. Objects compliant to CORBA-compliant Object Request Brokers (ORBs) can interface with one another regardless of their network type, architectures, and operating systems.

However, in recent years, OMG has introduced DDS, which has many similarities with CORBA in its distributed services. They are two independent standards; DDS is not a replacement for CORBA [55]. Besides DDS, OMG has also introduced other distributed technologies and services, such as Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS), DDS for eXtremely Resource Constrained Environments (DDS-XRCE), Remote Procedural Calls (RPC) over DDS (RPC_DDS) and many more.

2.5.3 Microsoft's OLE [81] [82]

Object Linking & Embedding (OLE) is a proprietary technology developed by Microsoft that focuses on the reusability of assets. With its underlying Component Object Model (COM) [49, 83], OLE defines communication infrastructure and protocols allowing connected objects to achieve functionalities. Depending on the functionalities, OLE objects are implemented using the OLE Object interface, with a wide range of other interfaces, and dynamically linked to another OLE application. OLE Control Extension (OCX) to develop and use custom user interface elements.

2.6 Reflection

Having studied the relevant works of literature above, it is obvious that:

- The robotics community has recognised the importance of standardisation for creating sustainable robotic infrastructure; not only does it catalyses cost-effective mass-market robotic products and empower consumers [45] but also promotes establishing a de facto standard for robotics, consequently accomplishing the implementation of Plug-and-Play enabling interchangeable components across the board.
- The robotics community has long been involved in creating robotic ontologies and specifications with the intention of standardising horizontally integrated robotic systems. However, the efforts of Plug-and-play robotic integrative standardisation is work-in-progress. Nonetheless, there are too many dissonances disagreeing with implementations and divergences resisting the formation of an integrative standard. It is not a simplistic matter of merely being technically optimal and meeting all parties' interests and opinions, but it is much more involved and complicated than it seemed. A joined effort from global technology firms, academics, and governments will help support and meet the industry's demand. Government initiatives such as providing various kinds of incentives to partakers may also help overcome the resistance and catalyse the formation of global integrative standards.
- Despite the presence of many plug-and-play robotic specifications and ontologies; however, by far, none of the robotic ontologies, specifications and implementations has been adopted as an exemplary candidate for a universal plug-and-play standard. Microsoft Robotic Development Studio (MRDS) [84] featured Concurrency and Coordination (CCR) for managing asynchronous parallel tasks and Decentralized Software Services(DSS) manifesting complex behaviours, has discontinued in 2014 and only offered limited support available by the forum community. Princeton Autonomous Vehicle Engineering claimed complications and overhead required to run Microsoft Robotic Development Studio (MRDS). Sony's OPEN-R architecture [85] is a proprietary system. Despite its highly modularised open architecture, which can reduce programming efforts through the APIs and enable users to alter the robot functionality by hardware modules, it did not get wide attention.
- In comparison, open-source ROS and ROS2, Orca, and many OMG robotic specifications and implementations have produced many products. Despite the early stage of open-source implementations attracting mostly academics, however, recently, it has gradually drawn attention from commercial robotic industries. Capitalising on the advancement of network and computer technologies, OMG has introduced many distributed robotic specifications, all focusing on interoperability amongst different DDS vendors and for different complexity of devices.

- Among the emerging specifications, OMG's approaches and specifications have been well-accepted in commercial and academic communities. Besides being robust, the other reason is that the implementations of OMG's specifications are open-source licenses like Apache, GPL, BDS, and others, which attract academics, hobbyists, and commercial players to contribute to its proliferation.
- Plug-and-Play is the ultimate manifestation of horizontal integrative standardisation, which benefits users and manufacturers. Besides self-detection and self-configuration, Plug-and-Play should possess context independence, location transparency, and evolvability attributes to influence wider acceptance. However, the development of Plug-and-Play entails complex algorithms and huge resources and costs. Fortunately, the Component-based Software Engineering (CBSE) component model, which features the reuse of components attributing low-coupling and high-cohesion, balanced component granularity, is a good candidate for leveraging Plug-and-Play. The model defines properties of system architecture modelling abstractions and a set of rules of components' compositions.
- Middleware resides under the application layer and above system software with underlying robotic ontologies and philosophies; providing development tools, and leveraging the DDS technology is increasingly popular in robotic development. Besides being compliant to component models, it also benefits the developers by being cost-effective, easing development, and accomplishing interoperability, resource utilisation reusability, and heterogeneity abstraction. More importantly, it fosters robotic ontologies and component-based software development and connects various software components of different robotic manufacturers running at different platforms. Amongst the middleware, ROS2 and OMG's DDSi-RTPS, DDS_RTC, and DDS-XRCE are well-received by distributed robotic communities.
- In recent years, the open-source ROS2 middleware that supports OMG's DDSi-RTPS has been increasingly versatile in many aspects. Education institutions can leverage ROS2 in general and specifically the DDS technologies for the increasingly demanding curriculum. With it, Plug-and-Play educational robots for the multi-disciplinary curriculum is now possible and highly achievable with reasonable efforts. As an incentive of leveraging on DDS, development can adopt many other DDSi-RTPS variants in accordance with their conditions. For instance, robots built with resource-limited Microcontrollers can use Micro-ROS or DDS-RTi Connect Nano, which conforms to DDS_XRCE specifications.

3 Methodology

3.1 Introduction

The fast-paced learning environment and increasingly complex multidisciplinary robotic curriculum have increased the occurrence and time needed for setting up the different robotic systems, which reduces the time teachers and students can use to focus on knowledge delivery and learning, respectively. In addition, constructing and configuring robots is inherently meticulous and, to some extent, hazardous.

Despite the advancement of robotic technology, commercially available educational robotics are inherently limited in functionalities and rigid for redevelopment and repurpose, resulting in education institutes facing uphill battles in upkeeping facilities to meet the increasingly complex disciplinary curriculum and diversity research activities.

Like other machinery, maintenance of robotic systems can run into support issues if it relies solely on the manufacturer's support. Out-of-date models, poor management and stiff competition may drive robotic manufacturers to rationalise their priorities. It is not uncommon for manufacturers to discontinue their support for the old models, forcing users to upgrade despite the system being very productive. Otherwise, users are left with no support.

The remainder of this chapter is organised as follows: In Section 2.2 describes the issues faced by the educational robotic community and the resolutions the MsBOT project adopts; Section 2.3 provides an overview of the underlying technology adopted in MsBOT; Section 2.4 reviews the framework of MsBOT system; Section 2.5 describes the MsBOT transport protocols; Section 2.6 provides the conceptual design of the MsBOT software system architecture; Section provides the MsBOT hardware design; Section 2.7 reviews the MsBOT user application system design; Section 2.8 describe the MsBOT development system; Section 2.9 provide a systematic review of the MsBOT Plug-and-Play feature; Section 2.10 provides the development and the implementation of the MsBOT Plug-and-Play feature; Section 2.11 reviews the design and implementation of the customed debugging tools.

3.2 System Overview

For decades, the Plug-and-Play feature has been well-accepted in PC industries that auto-detects and auto-configures when new or replacements are plugged into PCs. Like PCs, the

Plug-and-Play robotic system for education inherently features auto-detection and auto-configuration. It is intended to ease teachers and students configuring robotic systems with little effort, time, and knowledge. The direct implication of plug-and-play is the reduction of human intervention enabling easy component replacement and new component construction and testing. With plug-and-play, the configurational details are concealed from the students and teachers, leaving teachers and students more time to focus on teaching and studying, respectively. The benefits of plug-and-play can induce the development of a de-facto standard of robotic components, enabling horizontal integration amongst robotic manufacturers.

Among many robotic solution providers, ingenious Object Management Group (OMG) introduces the Data Distribution Service (DDS) for real-time systems, a machine-to-machine connectivity framework that aims to enable dependable, high-performance, interoperable, real-time, scalable data exchanges using publish-subscribe communication protocols. The DDS is developed by and for an open-source platform; it reduces the overall maintenance and development cost and improves the sustainability of the robotic systems. Besides, it allows the learning/teaching a series of authoritative distributed protocols such as DDS-RPC, DDS_RTPC, XRCE-DDS and ROS2. Still, it also enables further exploration of research resource-constrained robotic systems. The increasing advancement of networking technology, especially in Microcontrollers, has revolutionised the construction of robotic systems, transforming robotic systems from conventional standalone to remotely distributed robotic systems. Leveraging on the OMG's Extreme Resource Constraint DDS specifications (XRCE-DDS), eProsima developed Micro-ROS for supporting Microcontrollers to act as DDS entities despite being resource-limited.

In addition, Micro-ROS offers many advantages; by facilitating XRCE-DDS ROS2-compliant robotic system development and Microcontroller-optimised client APIs, and supporting various RTOSes, permissive license, active community, long-term maintainability, and interoperability. Unlike other XRCE-DDS implementations, the Micro-ROS, compliant with XRCE-DDS specifications, adopts FastRTPS publish-subscribe protocols, which adopt the unique client-server architecture which enables the low-resource devices acting as XRCE-DDS Clients to communicate to a server called XRCE Agent, which acts on behalf of its clients in the DDS Global Data Space.

3.3 The Underlying Technology

The increasingly versatile ROS2 has gained acceptance from many robotic communities. However, despite the advancement of Microcontroller technology, it was still challenging to implement ROS2 on Microcontrollers. For that reason, eProsima initiated the Micro-ROS project, whose goal is to put ROS 2 onto the Microcontroller. It is achieved by adopting Micro XRCE-DDS as its default middleware layer. eProsima Micro XRCE-DDS is an open-source wire protocol that implements the OMG DDS for eXtremely Resource-Constrained Environment (XRCE) standard. These devices are unique in their resources, i.e., minimum real-time operating system or no operating system, battery-powered, wired or wireless low bandwidth connectivity, and intermittent processing with periodic sleep mode. The aims of Micro-ROS are to access the DDS Global Data Space from resource-constrained devices, i.e., Microcontrollers. Complying with OMG's XRCE-DDS specifications, eProsima adopts the client-server architecture, where low-resource devices, called XRCE Clients, are connected to a server, called XRCE Agent, which acts on behalf of its clients in the DDS Global Data Space. The client-server architecture splits processing and memory load between the server and the client, allowing low-resource devices access to the DDS world through the server.

The MsBOT resolution is leveraging the advancement of network and computer technologies and the success of middleware ROS2 and OMG's DDS, particularly DDS-RTPS and DDS-XRCE specifications, which have shown their effectiveness in distributed robotic systems. To be interoperable amongst extremely resource-constraints devices in the DDS-compliant distributed systems, eProsima has developed FastRTPS and Micro XRCE-DDS implementation based on DDS-RTPS and DDS-XRCE specifications, respectively. eProsima Micro XRCE-DDS [28] is the default open-source middleware of Micro-ROS [30] and has been proclaimed as the official extension of ROS 2 for Microcontrollers (MCUs).

3.4 MsBOT System Descriptions

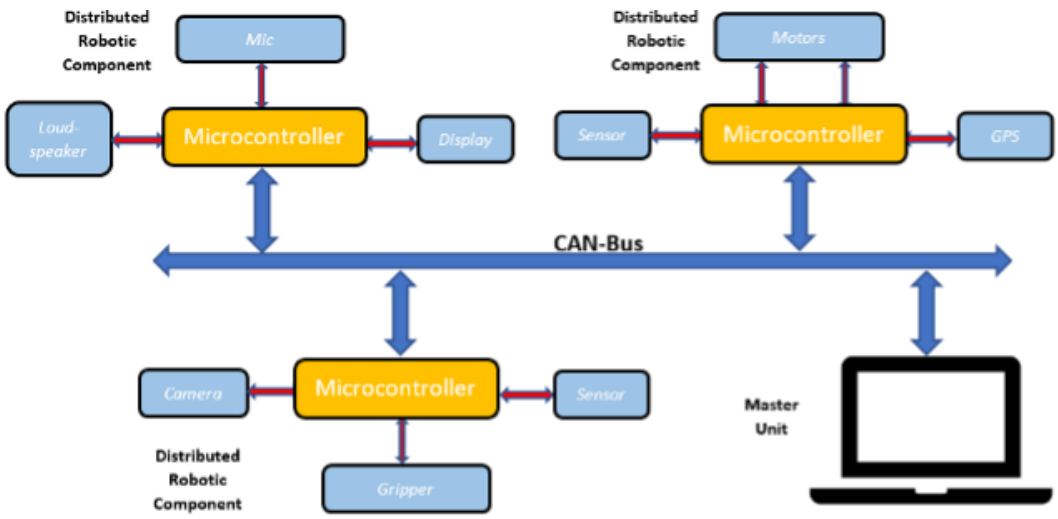

Figure 3 The illustration of the MsBOT distributed robotic system comprises self-sufficient robotic components connected through CAN-Bus.

The MsBOT is the abbreviation of **M**as**s**ey ro**BOT**, which is based on the DDS-RTPS technology of eProsima, i.e., Micro-ROS. MsBOT facilitates a Plug-and-Play facility for resolving the issues faced by teachers and students, enabling teachers and students to focus on their dedicated tasks efficiently rather than configuring and maintaining the robotic systems. In hardware aspects, MsBOT consists of at least a unit of Master component, which is also called Micro-ROS Agent and several components, namely Micro-Clients. The Micro-Clients are connected to the Micro-ROS Agent through the transport of the CAN-Bus. Figure 3 above shows a typical MsBOT physical system consisting of a Master component and multiple components connected through a CAN-Bus. The Plug-and-Play facility enables adding new components and removing or replacing old components on-the-fly without involving convoluted human configurations.

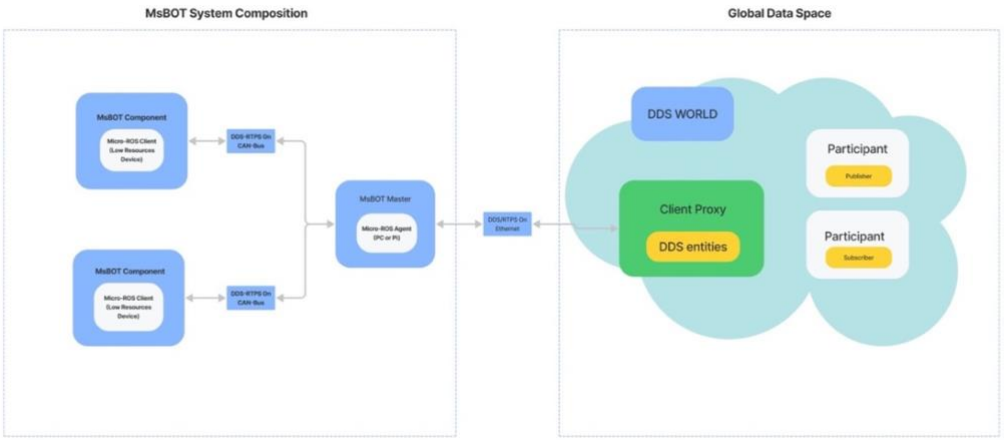

Figure 4 The DDS composition of MsBOT and the communications with the DDS world (Global Data Space)

MsBOT is not only a distributed robotic system but also inherits attributes of Micro-ROS, being modular, interoperable, scalable, cost-sensitive, practical, and sustainable. Every robotic component of MsBOT is a self-sufficient system made of a CAN-Bus-enabled Microcontroller and peripherals depending on its functionalities. The components are connected through CAN-Bus to form a comprehensive robotic system. The MsBOT components, also depicted as Micro-ROS Clients, are connected through the CAN-Bus.

The communication among components is handled by the Micro-ROS agent residing in MsBOT Master. For simplicity, the terminology of the MsBOT component is interchangeably used as Micro-ROS Client; the same applies to MsBOT Master with the Micro-ROS Agent. The Micro-ROS Clients are resource-constraint entities tailored for operating in the eXtremely Resource Constrained Environments (XRCE). Hence, the Micro-ROS Clients are also called XRCE-DDS Clients or devices.

The XRCE-DDS Clients communicate with the DDS world through the Micro XRCE-DDS Agent, which acts as a middleman translating messages from the XRCE-DDS Clients to messages understood by the DDS world, and vice-versa. In addition, the Micro-ROS Agent is also responsible for handling communication among the Micro-ROS clients. Figure 4 above shows the DDS composition of MsBOT and the communication of the XRCE-DDS Clients with the DDS world.

The Micro-ROS Clients perform request/reply/subscribe/publish/operations amongst the Micro-ROS Clients and the Agent, not forgetting the DDS Global Data Space. With the support of Remote Procedure Calls (RPC), as defined by the DDS-RPC standard, the Clients can communicate in the forms of request/reply messages with one another in the DDS Data

Space through the Micro-ROS Agent. The Agent processes these requests and responds with the reply messages of the operation status together with the requested data.

The Agent keeps track of the Clients by the corresponding dedicated ProxyClient entity created when the Client requests connections. The ProxyClient entity in the Agent represents the XRCE-DDS Client. Only after creating the ProxyClient can the XRCE-DDS proceed further with other operations such as creating Participants, Topics, Requester, Replier, Publishers, Subscribers, etc.

3.5 MsBOT Transport Protocols

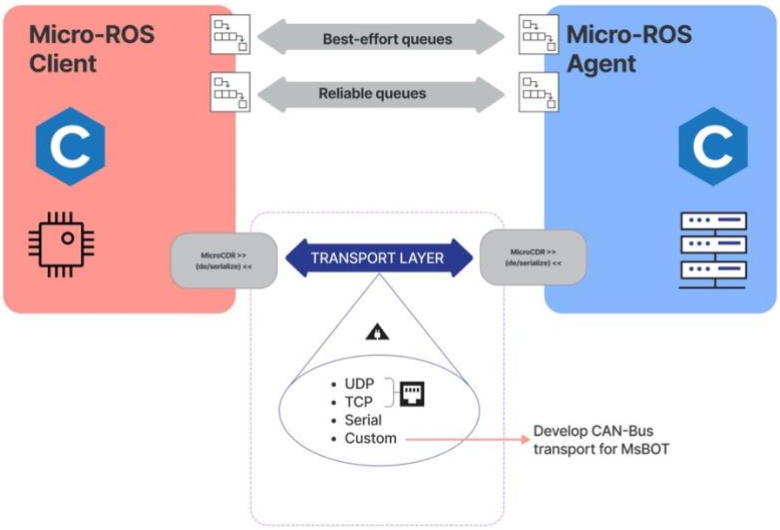

Figure 5 The transport choices of the Micro-ROS (Adapted from eProsima Micro XRCE-DDS[28])

When MsBOT development was in progress, Micro-ROS supported only UART serial and wired Ethernet UDP/TCP transport protocols, as shown in Figure 5 above. However, Micro-ROS allows customised transports which need to be developed by the intended party.

CAN-Bus was chosen for the MsBOT's transport protocols because it is well-known for its resilience to noise, which has long been widely accepted in the automotive industry. For the same reason, robotic systems resembling automotive would find CAN-Bus appropriate. From the bitrate consideration, the CAN standard classic 2.0A supports up to 1 Mbps, faster than the existing Micro-ROS UART bandwidth model. Instead of using the existing serial UART or

IP/UDP transports, MsBOT would develop a new transport, i.e., CAN-Bus for MsBOT connecting Micro-ROS Clients and Micro-ROS Agent.

The Micro-ROS Agent runs on Linux/Win/macOS systems, on which the CAN-Bus interface is achieved using a USB-CAN adaptor. However, the CAN-Bus transport protocol is leveraged by Linux CAN Socket. Meanwhile, the CAN interface is realised using the onboard CAN controller for the Micro-ROS Clients. The low-level CAN-Bus drivers responsible for CAN-Bus transport implementation would be developed for the Micro-ROS Clients.

3.6 MsBOT Software System Architecture

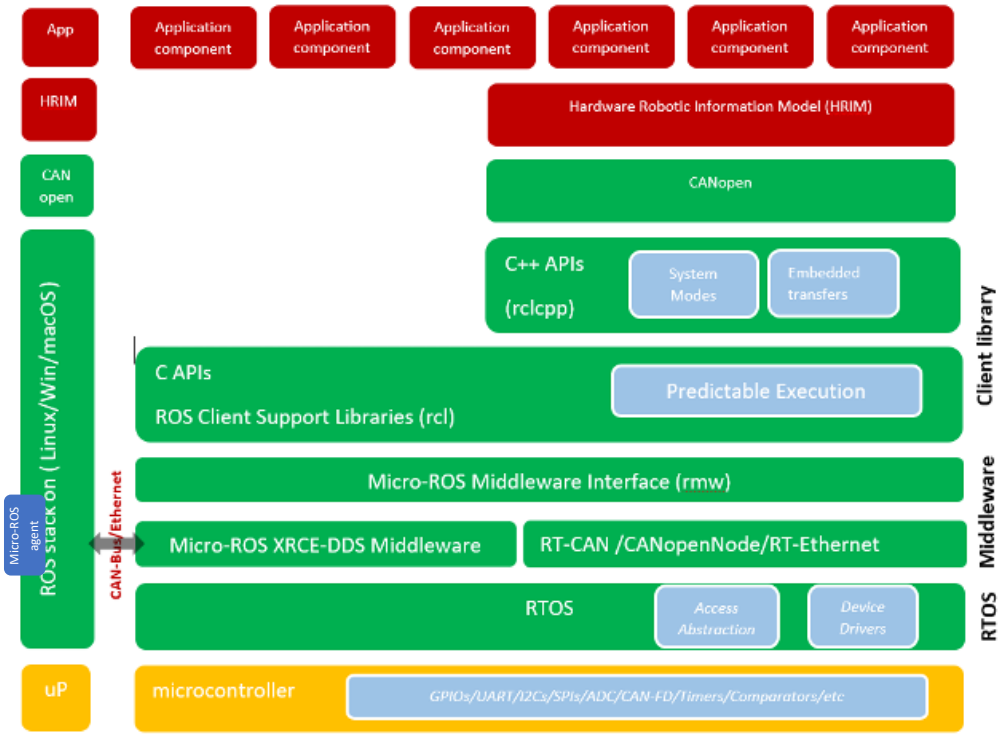

Figure 6 The illustration of the MsBOT layered system architecture based on eProsima Micro-ROS [30]

The robotic components of MsBOT, be it the Micro-ROS Client or Micro-ROS Agent, are built in accordance with the layered architecture, as shown in Figure 6 above. Micro-ROS is derived from the standard ROS 2 architecture and is compliant with DDS-XRCE specifications.

The Micro-ROS Agent is a ROS 2 node that wraps the Micro XRCE-DDS Agent [28]. The Micro-ROS Client is a scaled-down of ROS2 that enables the XRCE-DDS Clients, e.g., MsBOT Component, to access the DDS Global-Data Space through the Micro-ROS Agent.

The right-hand side of Figure 6 above shows the stacks of software modules for building Micro-ROS clients that are optimised for extremely constrained Microcontrollers.

To achieve real-time performance set by DDS-RTPS specifications, the Micro-ROS client includes support for various RTOSes such as FreeRTOS, Zephyr, or NuttX. The real-time performance is one of the significant milestones of ROS2 that is compliant with DDS-RTPS specifications. The MsBOT Components employ FreeRTOS as the Real-Time Executor. The adoption of real-time executives helps robot developers to achieve real-time performance at ease and leveraging on its capabilities of being

a. Deterministic execution that is real-time guaranteed
b. Single integrated platform of real-time and non-real-time functionalities
c. Support RTOS for small memory footprint and limited processing power Microcontrollers.

Micro-ROS Clients rely on the Client Library to ease robot development, which consists of various built-in routines and tools. It facilitates the programming of Micro-ROS Clients applications to achieve DDS-RTPS protocols requiring access to a publisher/subscriber architecture. In addition, Micro XRCE-DDS Agent employs a C++11. In contrast, Micro XRCE-DDS Clients employs a C99 library for the Client functionality, e.g., Micro CDR, a de/serialisation built-in routines of transport protocols used in the Client library and other tools such as Micro XRCE-DDS Gen, a code generator used for generating Micro CDR de/serialisation code and the Client applications. The library profile can be configured at compile-time through a set of CMake flags to achieve intended functionalities and image footprints.

As shown in Figure 6 above, the middleware is responsible for communicating between the low resources Microcontrollers and the DDS world. The underlying library of XRCE-DDS protocol implementation eases the DDS-RTPS application development on Microcontrollers because the details on the RTPS accessing the publisher/subscriber architecture are hidden from the developer. The Micro XRCE-DDS as a middleware solution has been adopted by many companies, such as Renesas and ROBOTIS. Besides, the middleware also features low resource consumption, multi-transport, multi-platform, QoS, and many more.

3.7 MsBOT User Applications Software Design

MsBOT system consists of multiple units that are categorised into two broad systems. Firstly, the Personal Computer (PC)-based system has a full-fledged operating system, complete ROS 2 stacks and Micro-ROS agent, CAN-Bus interface and other device drivers, and Micro-ROS applications. The Micro-ROS applications are called MsBOT Master and MsBOT Health-Inventory system. In addition, the Micro-ROS Agent takes up the higher processing capability, allowing the low-resource Micro-ROS clients to access the DDS world. Secondly, the low-resource Microcontroller-based systems are also called Micro-ROS clients. It comprises software modules adopted from eProsima and many other open-source resources, application-specific system-dependent modules, and Micro-ROS Participant applications. The right-hand side vertical stacks and the left-hand side vertical stacks of Figure 6 illustrate the PC-based MsBOT Master component architecture and the Microcontroller-based MsBOT Component architectures, respectively.

3.7.1 MsBOT Master Software Design

MsBOT Master runs on full-fledged, widely available operating systems, such as Microsoft Windows, Linux and macOS. At the time when this thesis was being written, the full ROS-2 stacks and Micro-ROS agent module had fully ported to Raspberry Pi. In other words, the Micro-ROS agent can now run on standalone PC-based systems and the tiny form factor Raspberry Pi.

In normal situations, off-the-shelve PC and Raspberry Pi do not equip with any CAN-Bus interfaces. However, the CAN-Socket has been well-established in Linux, Wins and macOS. A CAN-Bus interface can be established by connecting a CAN-Socket-compliant USB-to-CAN-Bus adapter on the PC or Raspberry Pi.

The PC-based system acts as Micro-ROS Agent, designed as a DDS-RTPS server, allowing the Micro-ROS clients to access the DDS world, including the other MsBOT components. Being compliant with ROS 2 architecture, also an OMG's DDS implementation [19], MsBOT Master features

A more powerful device to undertake a larger memory footprint and networking processing binaries of the standard DDS, which are not needed to be on MsBOT components, i.e.,

Micro-ROS Clients. Otherwise, it will bog down the operation of the extremely resource-constraint Microcontrollers.

Act as the XRCE Agent, which acts on behalf of its MsBOT components in the MsBOT in specific and the DDS Global Data Space [28] in general. The resource-constrained MsBOT components publish and subscribe to topics via an intermediate service handled by an agent in a DDS domain [28], resulting in the resource-constrained MsBOT components accessing the DDS Global Data Space.

3.7.2 MsBOT Components

MsBOT Components are built on resource-constrained Microcontrollers. MsBOT Components adopt eProsima DDS-XRCE implementation, which complies with OMG's DDS-XRCE specifications. From a software perspective, MsBOT Components are Micro-ROS Clients optimised for Microcontrollers. Micro-ROS Clients are still sufficient and efficient in handling the control and monitoring operation and low bandwidth communications despite their low resources and computational power. It is through the adoption of the client-server approach defined by OMG.

OMG defines the XRCE-DDS protocol [86] of the resource-limited devices, i.e., clients for communicating with larger systems [21, 27] through an XRCE Agent (server) [28]. To promise a small footprint and efficient execution in the resource-constraints Microcontroller, Micro-ROS splits the standard ROS-2 architecture amongst the Micro-ROS Agent and the Micro-ROS clients. The larger footprint, networking and computation modules reside on XRCE-agent, leaving only the low bandwidth dedicated Microcontroller-specific executives on the MsBOT Components. Resulting in the MsBOT Components, the DDS-XRCE clients adequately administrate the DDS-RTPS messages in a DDS domain [28, 86]. The software stacks of MsBOT Components are shown on the right-hand side of Figure 6 above.

3.8 MsBOT Hardware System Design

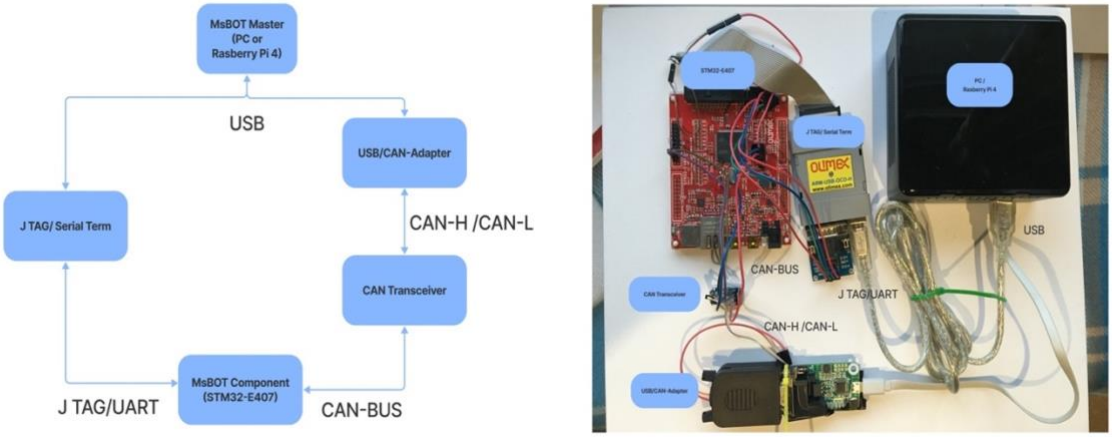

Figure 7 MsBOT hardware composition block diagrams (Left-Hand side) and the MsBOT composition in real-life (right-hand photo)

The construction of MsBOT is based on eProsima DDS-XRCE implementation, which aims to enable low resources devices, i.e., Microcontrollers, to access the DDS global data space. Micro-ROS is the product of eProsima DDS-XRCE implementation, which is derived from ROS 2 architecture in many aspects. Besides the ability to access the DDS world from low resources devices, Micro-ROS features low resource consumption, multi-transport, multi-platform, QoS, and many more.

Figure 7 shows the composition of the MsBOT components. The left-hand block diagram depicts the MsBOT's major hardware composition and the CAN-Bus transport connecting the devices, while the right-hand photo shows the MsBOT's actual hardware and the CAN-Bus wirings. To ease understanding of the relationship between the MsBOT and the Micro-ROS, MsBOT is analogous to a PC, while Micro-ROS is analogous to Microsoft Windows.

3.9 MsBOT Development Platforms

3.9.1 The Development Platform of MsBOT Master

The MsBOT Master consists of ROS 2 stack and Micro-ROS Agent packages, as represented in Figure 6 above. As the name implies, it acts as a broker, bridging the MsBOT components with the DDS world. In addition, MsBOT Master is also responsible for communication among MsBOT components.

The transport between Micro-ROS Agent and the Micro-ROS Clients are Serial UART and UPD/TCP. Besides multi-transport, Micro-ROS also supports multi-platform, QoS and many more. For academic reasons, instead of using the existing available transports, i.e., Serial UART and UDP/TCP, MsBOT decided to develop CAN-Bus transport for Micro-ROS. However, PCs or Raspberry Pi 4s do not carry CAN-Bus interfaces by default. MsBOT is adopting a USB-CAN-Bus adapter for Micro-ROS Agent to establish a CAN-BUS. The left-hand block diagram of Figure 5 shows the MsBOT hardware composition, which consists of MsBOT Master connected to MsBOT components by CAN-Bus.

To date, a PC or a Raspberry Pi 4 can build a MsBOT Master unit. The right-hand photo of Figure 5 shows that the MsBOT Master is built with an Intel NUC PC that uses a USB-CAN-Bus adapter. Nevertheless, a Raspberry Pi 4 with the CAN-Bus shield controller module [87] has been set up and is ready for further development and testing. The USB-CAN-Bus adapter and the CAN-Bus shield for Raspberry Pi are built-in with a CAN transceiver. Figure 5 shows no CAN transceiver at the CAN-Bus adapter's end.

Both have advantages and disadvantages with either a PC or a Raspberry Pi4. Empirically, a PC can handle multiple tasks as a full development platform, developing programs for MsBOT Master and MsBOT components. Instead of having one development station for a MsBOT component and one for a MsBOT Master, a PC can accommodate both development platforms. The other advantage of using PC over Raspberry Pi4 is that it is efficient due to its high-performance architecture. It can efficiently handle JTAG debugging and serial UART protocol for status monitoring on the MsBOT components while running as MsBOT Master. However, its high power, size, and weight make it unsuitable for building small form-factor robots.

Conversely, Raspberry Pi4, due to its small footprint, lightweight and low power advantage, is suitable for building small form-factor robots. It is rational to develop and test the robotic system on a PC and get transferred to Pi4 once development is completed. Figure

7 above shows that the PC is connected to the JTag and the serial term through a USB. The Olimex JTag dongle has built-in two functionalities, i.e., JTag Debugger and Serial Terminal, but only through one USB interface.

3.9.2 The Development Platform of MsBOT Component

MsBOT Components are the generic name referring to MsBOT robotic peripherals. Each of the MsBOT Components carries specific tasks. However, the focal point of the research activities is implementing the Plug-and-Play feature of MsBOT. All MsBOT Components use the same hardware design, i.e., using Olimex's STM32-F407 evaluation board, which comes with CAN controller Micro-ROS is multi-platform capable. Besides Olimex STM32-E407 evaluation, it also supports many other manufacturers' Microcontroller evaluation boards. However, Olimex STM32-E407 has the advantage of its support on development issues from the eProsima. Olimex STM32-F407 features:

STM32F407ZGT6 Cortex-M4 210DMIPS, 1MB Flash, 196KB RAM, 3 x 12-bit 2.4 MSPS A/D, 2 x 12-bit D/A converters, USB OTG FS and USB OTG HS, Ethernet, 14 timers, 2xUSART, 4xUART, 3 x SPI, 3 x I2C, Ethernet, 2 CANs, 3 x 12-bit ADCs, 2 x 12-bit DACs, 114 GPIOs, Camera interface, JTAG connector with ARM 2x10 pin layout for programming/debugging, Ethernet 100Mbit.

Besides the available support on development issues, Olimex STM32-F407 Ethernet 100Mbit is very handy for verifying the UDP/TCP transport of eProsima DDS-XRCE implementation. Furthermore, the multiple UARTs also very convenient for implementing the serial console for verification and data monitoring purposes.

However, for establishing a CAN-Bus, it needs a CAN transceiver to convert CAN-TX and CAN-RX to CAN-H and CAN-L signals that can travel on longer wires. The CAN transceiver from Philips, namely the TJA1050 module, is employed in MsBOT development. Figure 5 shows that the CAN transceiver resides between the STM32-E407 Microcontroller and the USB-CAN adapter.

The JTag dongle from Olimex has dual functionalities. Besides the flash programming and GDB debugging through JTag, a USB-serial adapter allows serial UART communication with a PC through a USB interface. The Olimex JTag will enumerate as a GDB debugger/programmer and a serial terminal, i.e., /dev/ttyUSB0, in Linux systems. Through the device file/dev/ttyUSB0, data on the Microcontroller can be transferred and displayed on

a computer for various purposes. One of the most important applications is to display data through the serial console of the computer for debugging or logging important events during program execution on a Microcontroller. Figure 7 above shows a USB cable connecting the PC and the Olimex JTag dongle, while the other end of the JTag dongle is split into two connections, namely JTag and serial UART interfaces.

3.10 MsBOT 2-Step Piggyback Plug-and-Play System Description

The focal points of MsBOT's research activities are implementing a Plug-and-Play (PnP) robotic system that features auto-detection and auto-configuration when installing new robotic components or removing old robotic components. It aims to ease teachers and students in configuring robotic systems for respective teaching and learning purposes and, as a result, reduce or better eliminate human intervention to save time and effort. Besides, a successful Plug-and-Play model can induce further development, becoming a de-facto standard and enabling horizontal integration amongst robotic manufacturers. Micro-ROS offers many advantages, facilitating the development of ROS 2-compliant data distributed systems and Microcontroller-optimized modules and supporting robust real-time performance through user-selected RTOSes. For details on Micro-ROS modules, please refer to Figure 6 above.

In addition, Micro-ROS features an open-source permissive license. eProsima, as the host of the Micro-ROS forum and system repository, together with many leading manufacturers, actively provides support and maintenance. Micro-ROS is compliant with XRCE-DDS specification, which features a client-server architecture. The client-server architecture enables low-resource devices acting as XRCE Clients to communicate to a server called XRCE Agent, which acts on behalf of its clients in the DDS Global Data Space.

MsBOT Plug-and-Play software system is implemented in accordance with Micro-ROS's architecture which splits the standard ROS2 load amongst the Micro-ROS Agent and the Micro-ROS clients. By the convenience that MsBOT Master also performs as a Micro-ROS Agent. Any other MsBOT component which runs on a more powerful system like Raspberry Pi 4 can operate as a Micro-ROS agent. In other words, a MsBOT can have a MsBOT master and a MsBOT component that either runs on a PC or a Raspberry Pi 4. However, the PCs which MsBOT Master runs on are convenient and easy to set up compared to a

Microcontroller system. For example, the administration of MsBOT master specifications and future development. It needs to be configured prior to the components can be assembled through the Plug-and-Play operation. Hence, a PC with relevant development applications can be useful and convenient for performing tasks.

The Plug-and-Play feature of MsBOT is achieved through the FastRTPS [88] protocols of eProsima XRCE-DDS implementation, i.e., Micro-ROS. The MsBOT Plug-and-Play leverages on the Plug-and-Play feature of the Micro XRCE-DDS Agent and the APIs to implement the Micro XRCE-DDS Clients [28]. Hence, it is named as MsBOT 2-Step Piggyback Plug-and-Play operation, where another Plug-and-Play feature is built on top of the other Plug-and-Play.

The Plug-and-Play software system is constructed with two hardware entities: the MsBOT Master and the MsBOT components. From FastRTPS [88] point of view, the MsBOT Master operates as a Publisher and a Replier, while the MsBOT components operate as Subscribers and Requesters, respectively. Despite being a distributed robotic system, it can be explained using analogous of an interest group of certain characteristics and its members. The group has a database of its members' profiles. Members can access the group facilities, including other members, through the group's administration. Members will be verified for their identities and registered for entering and deregistered for leaving the facilities. The design of MsBOT takes into consideration of:

- Open technology – Avoid re-inventing the wheels, and reuse and support currently available robotics technology, especially open-source and open technology. Instead, the MsBOT Plug-and-Play leverages on the Plug-and-Play feature of the Micro XRCE-DDS Agent and the Micro XRCE-DDS Clients APIs [28] to build the second Plug-and-Play feature solely for MsBOT applications, namely 2-Step Piggyback Plug-and-Play operation.
- Security – Only the minimal information of the master robotic is being published, and no components' information is being published. The MsBOT components only send out minimal specifications of the components through the Req messages for authentication purposes. The MsBOT Master and MsBOT inventory do not send positive confirmation information in the Rep messages. In addition, the relationship of the Master unit with its components is not published to the public network. No public knows the entire robot's functionality or composition to minimise eavesdropping.
- Practicality – The design is duplicable. A new educational robot of different functionality and/or learning/teaching objectives can be created easily by simply duplicating the structure of MsBOT Master publishing messages and its components database but making

changes to the robot publishing information and components' specifications. In addition, the design employs systematic bottom-up approaches where the development of robots is divided into the component and the composition of the components.

- Extendibility – Ability to add new functionalities at the component level or the system level by adding a new entry in the Health Inventory database. A new robot system can also be created at ease when in need.
- Interoperability – A component of one robotic system operates on another robotic system with minimal changes in the Master profile and Health Inventory database. The robotic operation, both the system and component levels, is rather a functional dependent, not hardware dependent
- Sustainability – eProsima Micro-ROS is an open-source solution. Besides, the overwhelming support from the open-source community and global technological manufacturers can provide long-term sustainable resolution.

The interest group is equivalent to the MsBOT Master entity, and the members are equivalent to MsBOT Components. Coincidently, the MsBOT Master entity shares the same hardware system with the Micro-ROS Agent for convenience and efficiency. In contrast, the MsBOT components are the names given to the Micro-ROS Clients with respect to the MsBOT project.

The administrator of the interest group periodically publishes newsletters. Anyone can subscribe to the newsletters, however, when only members have authenticated using the database of its members. Only members that match the profile will be granted access to the facilities. Members' profiles are protected by this method as they are not being published publicly. The members that enter the facilities will be registered and similarly get deregistered when leaving the facilities. New members need to be added by the administration prior to accessing the facilities. Otherwise, it will fail in the authentication process.

3.10.1 MsBOT Master

The MsBOT Master is configured as a Publisher in accordance with the OMG's DDS-RTPS and XRCE-DDS protocols. MsBOT Master is configured as a Publisher that publishes the master profile to all the subscribers. By default, the MsBOT components subscribe to the MsBOT Master.

Without the subscription, MsBOT components would not be able to respond with a request message to the publishing message. Through the contents of the request messages, a MsBOT component is authenticated and then given the authorisation to perform the required robotic operation. Figure 8 below shows that MsBOT is connected to three components, also called participants in the Micro-ROS convention. Through the publishing messages, the subscribed participants respond with request messages. The request messages carry the MsBOT components' profiles which get verified and authenticated and finally granted access to the MsBOT facilities.

However, unlike normal database systems with a master database and a duplicated copy of the master database for active processes, in MsBOT, the active statuses are handled in the extra schema of the MsBOT Master Inventory database. However, the MsBOT Master inventory is also the MsBOT Health Inventory with inactive status in the Status column. For simplicity, the master inventory is another name for the health inventory.

The master inventory lists robotic components that make up the MsBOT. However, the entry of the inventory lists can be added, removed, or replaced by the dedicated administrator. Being a master components inventory, it is responsible for verifying and authenticating the connected components prior to registering to the Health Inventory database.

3.10.2 MsBOT Components

Micro-ROS empowers low resources devices to become first-class participants of the ROS2 ecosystem, manifesting smaller robots, smart embedded devices and IoT. Figure 8 shows that the MsBOT comprises a MsBOT Master, acting as a Micro-ROS agent, and connected to three MsBOT Components. The MsBOT Components are also called Micro-ROS clients with respect to the Micro-ROS convention. However, from the XRCE-DDS RTPS protocol, a perspective is an identifier of an RTPS entity. A Micro-ROS Client can only communicate with others be in the DDS Global space or the other Micro-Clients in the same network after a participant has been created. A Participant can have multiple roles, i.e., a Publisher or Publishers, a Subscriber or Subscribers, or Publishers and Subscribers.

The Participants are subscribers subscribing to the Publisher MsBOT Master inventory. Upon powering up, the first task of the subscribers is to respond to the MsBOT Master inventory publishing messages with its MsBOT Component characteristics. This information

is authenticated by the MsBOT Master Inventory and then granted access authorisation by creating an entry in the MsBOT Health Inventory.

In addition, the participants can communicate with one another through RTPS protocols. Participants can operate as Publishers publishing the intended content, allowing the interested participants who operate as Subscribers to subscribe to the published content. Through this manner, participants communicate with other participants. Participants publish messages by broadcasting them into the network. Every participant in the network can receive it through a subscription. However, to be selective, participants could set verification on the participants' IDs, dropping the messages if the verification failed or passing it on otherwise.

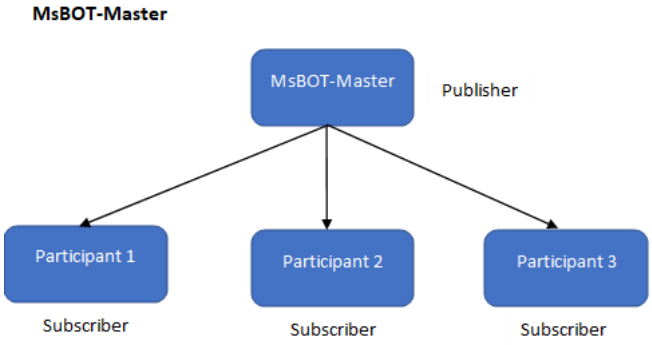

Figure 8 The MsBOT master during communication with three components

3.10.3 MsBOT Health-Inventory

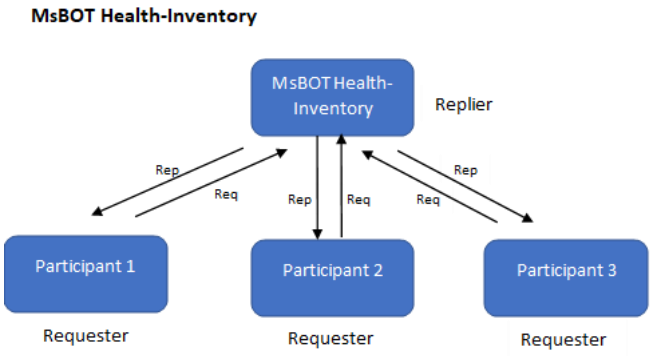

Figure 9 The MsBOT Health Inventory in communication with three components

The MsBOT Health Inventory is responsible for the health status of the verified registered MsBOT components. Upon verification and authentication, components' statuses are created dynamically in the Health Inventory database. In this case, the MsBOT Health Inventory

designated as a Replier will set three at the health status counter in the health inventory database. At the time when the timer in the health inventory of the Replier lapses, the corresponding status of the requester will be decremented. At the time when the status reaches zero, the corresponding participant will be delisted from the health inventory database.

The delisted participant is no longer an active MsBOT component; all the messages to and fro the delisted component will come to a halt. However, the component can still get listed again by going through the first verification process, i.e., upon receiving publishing messages from the Publisher MsBOT Master Inventory, respond to it by sending its component characteristic.

Figure 9 above shows a MsBOT Health-Inventory, a Replier by the Micro-ROS convention, replies to a Reply message to the corresponding participants, also called a requester in this setup. Once granted access through the first Rep message, the participants send out requests periodically informing the MsBOT Health Inventory that they are alive. Upon receiving the Request message, MsBOT Health Inventory sets a value of three on the health status counter and then replies to the requests accordingly.

A timer is set up to decrement the health status counter of all participants at a fixed interval of 2000 msec. In the event that the health status of a participant is negative, which means it has missed three heartbeats in a roll. The scenario implies that the participant is no longer active. Hence, it is high time to remove it from the health inventory database. Figure 9 above shows the MsBOT Health Inventory acts as a Replier replies Rep messages to the Req sending Participants, also called Requesters.

3.11 MsBOT 2-Step Piggyback Plug-and-Play System Design

The Plug-and-Play system is the focal point of MsBOT system development. Its goal is to hide the configurational details from teachers and students, especially at the tertiary level, allowing them to focus on teaching and learning activities instead. Unlike many other standalone robotic systems, MsBOT is leveraging eProsima's Micro XRCE-DDS implementation deriving from ROS 2, making MsBOT a distributed robotic system. Besides being open-source middleware, the eProsima Micro XRCE-DDS facilitates the Plug-and-Play feature connecting and discovering the Micro XRCE-DDS Agent and the Micro XRCE-DDS Clients.

As a distributed robotic system, its robotic components are not unnecessarily located in the same hardware unit; instead, in MsBOT, they are distributed remotely and are connected through CAN-Bus protocols.

In the MsBOT system design, the Plug-and-Play is achieved through two entities, namely MsBOT Master and MsBOT Component. Figure 8 shows the block diagram of a complete MsBOT Plug-and-Play System Design. The MsBOT Master has three main roles:

- Publish the MsBOT profile.
- Perform the MsBOT health inventory operation.
- Operate as Micro-ROS Agent.

All three roles can be in the same hardware unit, i.e., Raspberry or Linux-based PC. It is not a requirement for them to share the same hardware unit. The reason for setting up Micro-ROS Agent using a PC or Raspberry Pi 4 applies to MsBOT Master and the MsBOT Health-Inventory database. Nevertheless, having MsBOT Master and MsBOT Health Inventory on the same hardware unit does help improve the processing of the MsBOT Master applications and the MsBOT Health Inventory applications. Figure 9 shows the communications of the MsBOT-Master and the MsBOT Inventory participants residing in the same hardware unit. The timer set the 2000 msec interval, also called the heartbeat, to check the participants' health status. Besides being a subscriber to the MsBOT Master Inventory, Participants also operate as Requesters that send a heartbeat to the MsBOT Health Inventory. In the event the health status of the participant is negative, it will remove the participant's entry from the Health Inventory database. The removed participant is no longer part of the currently active system. When a participant's entry gets removed, the Replier sends no Rep message, i.e., MsBOT Health Inventory.

Robotic components like automation devices need to communicate with one another, complementing each other to form a complete, practical and useful robot system. In MsBOT, participants operate as publishers as well as subscribers. Depending on the topics of interest, subscribers subscribe to the publishing messages. Figure 10 below shows that Participant 1 publishes a topic, and Participant 2 subscribes to that, while publishes another topic, and Participant 3 subscribes to that. Similarly, Participant 3 publishes a topic, and Participant 1 subscribe to that.

Like other DDS-RTPS implementations, Micro-ROS's FastRTPS [59] publishes messages by broadcasting them to the network. Any participants in the network can access them. However, processing every published message is not wise. It is wasting much of resources.

Hence, to be efficient and sustainable, participants can define the topic of interest using the topic IDs. Only topic IDs that match the definitions will pass on to higher-level executives for further processing.

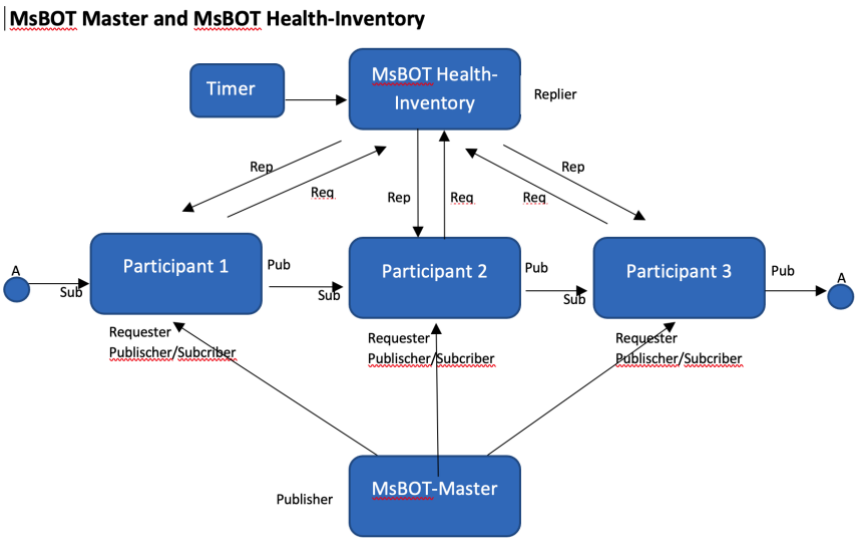

Figure 10 The communication between the MsBOT Master and the MsBOT Health Inventory and the three participants

3.12 MsBOT 2-Step Piggyback Plug-and-Play System Descriptions

The Plug-and-Play is an integral feature of MsBOT development. Unlike other resolutions, the MsBOT's Plug-and-Play resolution takes advantage of distributed systems' characteristics. The robotic components are not only hardware-independent and operating system agnostic but independent of the operating system device drivers and hardware interface protocols. MsBOT adopts the eProsima Micro-ROS system. MsBOT comprises the MsBOT Master and at least one MsBOT Component. Instead, the MsBOT Plug-and-Play leverages on the Plug-and-Play feature of the Micro-ROS (XRCE-DDS) Agent and the Micro-ROS (XRCE-DDS) Clients APIs [28] to build the second Plug-and-Play feature solely for MsBOT applications.

3.12.1 MsBOT Master System

The Plug-and-Play feature of MsBOT is achieved through the FastRTPS protocols of eProsima's XRCE-DDS implementation. MsBOT Master is responsible for publishing

MsBOT profiles enabling the dedicated MsBOT components to identify the robotic system to connect to. The MsBOT profiles describe what MsBOT is. It is an identifier identifying the robotic system from other robotic systems in the DDS world. The MsBOT profiles contain count, robot name, version, serial numbers, date released, and manufacturer name for identification purposes. Before further processing, this information is verified by subscribers, i.e., robotic components.

From the FastRTPS perspective, the publishing message is called topic. The topic which MsBOT Master publishes is achieved using the following code. MsBOT Master publishes the topic till the count limit has been reached or the session connection is dropped. The code below shows the conditions for publishing the topic and how the msbot_master topic is constructed.

```
 bool connected = true;
uint32_t count = 0;
while (connected && count < max_topics)
{
    msbot_master topic = {
      ++count, "MSBOT_MASTER", "MSBOT_V1","X100Y100Z100", "8-4-2021","MASSEY_ENGINEERING_DEP"
    };
    ucdrBuffer ub;
    uint32_t topic_size = msbot_master_size_of_topic(&topic, 0);
    uxr_prepare_output_stream(&session, reliable_out,DataWriter_id, &ub, topic_size);
    msbot_master_serialize_topic(&ub, &topic);

    printf("Send topic: %s, %s, %s, %s, %s, id: %i\n", topic.message, topic.name, topic.id, topic.dof, topic.make, topic.index);
    connected = uxr_run_session_time(&session, 5000);
}
```

Notes: The topic is sent out to the DDS world through the ucdrBuffer using msbot_master_serialize_topic(&ub, &topic). However, before sending it out, it needs to determine the topic's size and must establish the output stream. Figure 11 below shows the system flow of the MsBOT Master.

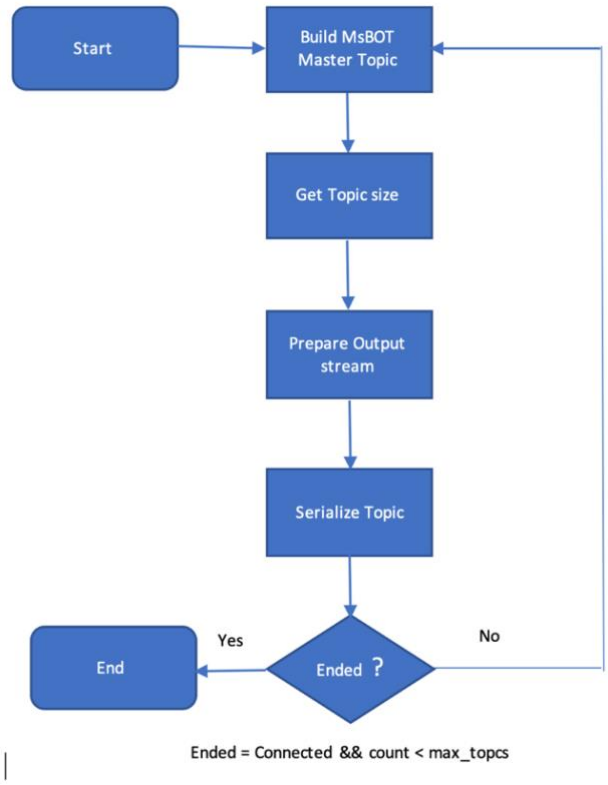

Figure 11 MsBOT Master System Flow chart

3.12.1.1 MsBOT Health Inventory System

MsBOT Health Inventory System is responsible for inventorying the health status of the active MsBOT components or Participants by Micro-ROS convention. Any participants can send the Req messages to the MsBOT Health Inventory System. However, only participants with valid identities matching the registered partnum(s) are kept alive in the health inventory and will be responded to by the health inventory manager.

The participants send out the Req messages at 1000 msec fixed intervals. Upon receiving the Req messages, the MsBOT Health Inventory system verifies the partnum(s) with the registered partnum(s) in the health inventory. If it matches but both its health_inv_part_reg and health_inv_life_status is empty, the sending entity is considered a new participant, and its partnum is registered in the registered partnum record. In the event, the partnum matches the partnum in the health_inv_part_reg column, which shows the participant is an active participant. As such, its

health_inv_life_status is reset to value three. Figure 12 below shows the participants' main activities upon receiving Req messages.

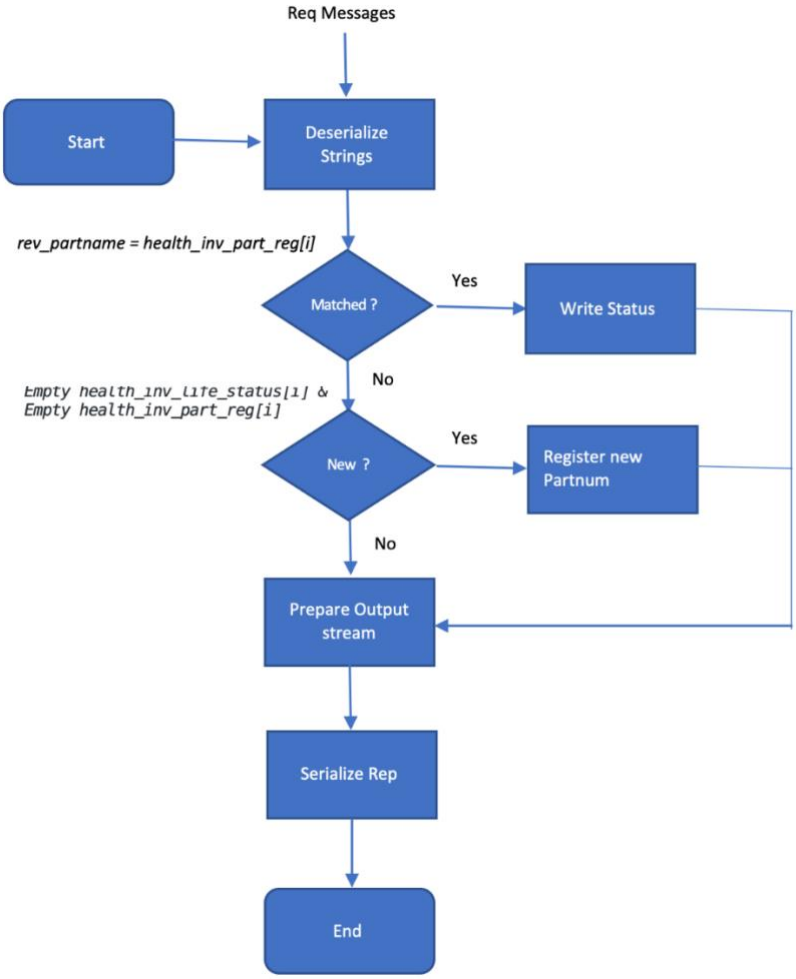

Figure 12 MsBOT health inventory system flow chart

3.12.1.2 MsBOT Heartbeat Manager

The Heartbeat Manager is responsible for the health inventory of the active participants. Upon receiving Req messages from the active participants, their health_inv_life_status is reset to three. Suppose the participants break down and stop sending out Req messages, the values of health_inv_life_status need to update to reflect the situation. A fixed interval timer at 2000ms is set up for constantly decrementing the values of health_inv_life_status. When the values reach zeroes, the corresponding participants are no longer active and need to be removed

from the health_inv_part_reg list. Similarly, the corresponding health_inv_life_status value needs to be cleared.

The following lines show that the timer, namely timer3, is set up using the start_timer API. It is set to a 2000 msec interval. While the time_handler3 below shows execution routines when the fixed interval is up.

```
    /* Set up the timer3 call-back */
timer3 = start_timer(2000, time_handler3, TIMER_PERIODIC, NULL);


/* The timer3 call-back routines */
void time_handler3(size_t timer_id, void * user_data)
{
   printf("2000 ms timer expired. (%ld)\n", timer_id);
   printf("b4-health_inv_life_status[%d][%d][%d]\n",
health_inv_life_status[0],health_inv_life_status[1],health_inv_life_status);
   printf("b4-health_inv_state[%d][%d][%d]\n", health_inv_state[0],health_inv_state[1],health_inv_state );
   decrement_life_status();
   remove_dead_parts();
   printf("a5-health_inv_life_status[%d][%d][%d]\n",
health_inv_life_status[0],health_inv_life_status[1],health_inv_life_status );
   printf("a5-health_inv_state[%d][%d][%d]\n", health_inv_state[0],health_inv_state[1],health_inv_state );
}
```

3.12.1.3 Mutex Strategies

However, health_inv_life_status values are shared by two threads, i.e., the onRequest thread and the timer thread. A mutex or lock mechanism is employed to enforce limits on access to the resource to prevent the shared health_inv_life_status values from being “stolen” by other threads, which causes misjudgement. To be accurate, it is a spin_lock mechanism that blocks the thread’s execution requesting the lock until the active thread finishes its execution, passing access to the requesting resource. With a spinlock, the lock requesting thread waits (“spins”) until the lock becomes available. The two functions below show the implementation of the spin_lock mechanisms, namely set_glock() and clear_glock().

```
/* Set Lock routine */
void set_glock(void){
  while (g_lock);          < ------ spin till the lock is released by other threads
  g_lock = true;           < ----- Set the lock for the current active thread
}

/* Reset Lock routine */
void clear_glock(void) {
  g_lock = false;          < ----- Reset the lock. Other threads can obtain it
}
```

The timer thread requests the lock to prevent the contents from being "stolen" by other threads while the timer thread decrements the health_inv_life_status. After acquiring the lock, the timer thread verifies and decrements the health status. At the end of the process, it releases the lock. See details in the following code.

```
 set_glock();                          < ----- Requesting for Lock
if (health_inv_life_status[i]) {       < ----- Lock obtained
   health_inv_life_status[i]--;
}
clear_glock();                    < ---- Releasing the lock
```

3.12.2 MsBOT Components - The Participants

The MsBOT Components are responsible for performing the robotic tasks. The MsBOT components are called Participants by the Micro-ROS convention. MsBOT components cooperate to complete tasks. It is done by communicating with one another to share information, helping make sound decisions timely. In addition, MsBOT components allow access to the DDS world across remote networks.

As participants, MsBOT components entrust with many roles. Besides operating as subscribers for the MsBOT Master, MsBOT components also operate as requesters,

subscribers, and publishers. Figure 5 shows that Participant A publishes A messages and subscribes to C messages, and Participant B subscribes to A messages from Participant A while publishing B messages. The C participant subscribes to B messages while publishing C messages. MsBOT components use the Requester to register and update the MsBOT health status, the Publisher to publish the information and the Subscriber to obtain other components' information.

The operations and functionalities of MsBOT are pre-meditated and documented in the MsBOT blueprint. Fresh MsBOT components can refer to the MsBOT blueprint for resuming the robotic tasks. Add a new MsBOT component by adding a new entry to the MsBOT blueprint. Similarly, remove the existing MsBOT components by deleting the corresponding entries from the blueprint. In addition, the other components that subscribe to the removed entries will need to edit accordingly.

MsBOT components share a standard role, i.e., operate as a Requester to send a heartbeat to the MsBOT Master. The following code shows that the Req message is formed using the partnum and the localpartname. The Req message is being sent out through the ucdrBuffer.

```
/* Req message construction and transmission */
 ucdrBuffer ub;
uint32_t topic_size = HelloWorld_size_of_topic(&topic, 0);
uxr_prepare_output_stream(&session, reliable_out,DataWriter_id, &ub, topic_size);
HelloWorld_serialize_topic(&ub, &topic);
ucdr_init_buffer(&ub, request, sizeof(request));
ucdr_serialize_string(&ub, partname);
ucdr_serialize_string(&ub, localpartname);
uint16_t request_id = uxr_buffer_request(&session, reliable_out, requester_id, request, sizeof(request));
```

Another standard role MsBOT components perform is subscribing to the Rep messages from the MsBOT Master. Upon receiving the Req messages, MsBOT Master replies to the Req messages with the Rep messages. The following shows the callback routine when receiving Rep messages. It shows a basic setup of a callback routine. Any other activities can be added when in need.

```
/* The callback routine of Rep messages of Participants */

void on_reply( uxrSession* session, uxrObjectId object_id, uint16_t request_id, uint16_t reply_id,
ucdrBuffer* ub, uint16_t length,  void* args)

{

   (void) object_id;

   (void) request_id;

   (void) length;

   //uint64_t result;

   char result[3*32];

   ucdr_deserialize_string(ub, result, sizeof(result));


#ifdef WIN32

   printf("Reply received: %I64u [id: %d]\n", result, reply_id);

#else

   //printf("Reply received: %" PRIu64 " [id: %d]\n", result, reply_id);

   printf("Reply received: %s [id: %d]\n", result, reply_id);

   #endif /* ifdef WIN32 */

}
```

3.12.3 MsBOT Components – The Participants' General-Purpose Input/Output (GPIO)

Unlike microcontrollers, PCs do not come with the general-purpose input and output pins, which users use for controlling or monitoring external peripherals. Despite the MsBOT being developed on the Linux PC, it would need to demonstrate its capability to control and monitor external peripherals like what a normal robot does. Besides, the USBGPIO module has significant educational value for students to learn how to learn using PCs to control external peripherals like a microcontroller. USBGPIO complements PCs to fulfil the learning and teaching of the Internet of Things (IoT) at one's convenience.

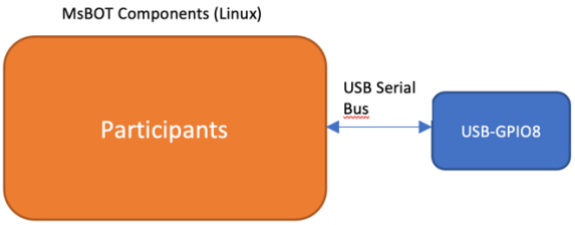

Figure 13 USB GPIO connection to the participant through the USB serial bus

The USBGPIO module is attached to the MsBOT Components through the USB interface. In fact, the USB interface enumerates as serial UART protocols. Figure 13 above shows that the Numeto USBGPIO module connects to the Participant through the USB serial bus. The corresponding software module for the USBGPIO module has been successfully implemented and is ready for deployment in the MsBOT robotic system.

In the Linux system, devices are represented by files. The Numeto USBGPIO module is represented as "*/dev/ttyACM0*" when connected. The following code snippets show the four significant functions of the USBGPIO module.

- Open the file device
  To open a device using the open() API with the corresponding device file as arguments. However, a device file needs to be represented as a string. See the code snippets below for details.

```
char *portname = "/dev/ttyACM0";

fad = open(portname, O_RDWR | O_NOCTTY | O_SYNC);  /* e.g., ttyUSB0 is the FT232 based USB2SERIAL Converter
*/
                                                    /* O_RDWR   - Read/Write access to serial port       */
                                                    /* O_NOCTTY - No terminal will control the process   */
                                                    /* Open in blocking mode,read will wait             */
```

- Close the file device
  To close a device, use the close() API with the corresponding device file identifier.

```
close(fd);
```

- The Write Operation

To write to the device, it uses the write() API with the corresponding device file identifier, and the Numeto write command, "*gpio set %d",* where *%d* is the variable of the intended write values. See the code snippets below for details.

```
sprintf(numeto_command, "gpio set %d\r", gpio_num);
    printf("numeto command %s \n", numeto_command);
    if (write(fd, numeto_command, strlen(numeto_command)) != strlen(numeto_command))
    {
       printf("Write error - %s \n", strerror(errno));
       exit (1);
    }
```

- The Read Operation
  To read from the device, it is achieved using two consecutive read operations. The first reading specifies the target, and the second reads the values from the buffer. The first read operation uses the read() API with the corresponding device file identifier, and the Numeto read command, "*gpio read %d",* where *%d* is the variable of the intended read target. It follows the second read operation still uses the read() API with the corresponding device file identifier as the first argument, but the second is the buffer identifier , and the third argument is the masking format. See the code snippets below for details.

```
sprintf(numeto_command, "gpio read %d\r", gpio_num);
printf(" numeto command %s \n", numeto_command);
    if (write(fd, numeto_command, strlen(numeto_command)) != strlen(numeto_command))
    {
       printf("Write error - %s \n", strerror(errno));
       exit (1);
    }
    printf("read_gpio write() succeeded \n");
    if ((len = read(fd,buf,255)) > 0) {
       buf[len] = '\0';
       temp = buf[13] - '0';
       *result = temp;
       printf("result len = %d GPIO-%d = %d \n", len, gpio_num, temp);
       status = true;
    }
```

3.13 MsBOT Debugging Tools

3.13.1 Retarget Serial UART printf()

Despite the increasing advancement in Microcontroller technology, adding more peripheral controllers, larger Flash and RAM size, faster system clock, etc., still debugging at runtime is challenging. GDB can offer a solution, but GBD is a burden for most Microcontrollers in many cases. Besides, it is not available or practical for Microcontrollers due to its inherently limited resource conditions.

Printing or displaying debugging messages at runtime is the most practical solution because it does not take up many Microcontroller resources. Besides, there is no learning curve for implementing it. To achieve that, send the debugging messages through the designated serial UART port to a dumb terminal or pseudo-terminal on a PC. However, the low-level direct access to UART port messaging is inherently fixed; variable values are not and cannot be incorporated into the debugging messages like the usage of printf(). Therefore, using printf() is a preferred solution. However, unlike the printf() function on PC, the printf() on Microcontrollers is not meant for the UART serial port. To make the printf() on Microcontrollers send messages through a designated serial port, it will need to modify the low-level transport routines to divert the messages to the designated serial port instead.

To achieve that, it involves two files, namely retargert.c and main.c. The goals are to redirect the debugging messages to the designated serial UART port. The *void RetargetInit(UART_HandleTypeDef *huart)* in file retarget.c is responsible for setting up the serial UART port. The calling routine passes the designated serial UART port in the function argument. It then assigns to the gHuart variable for use by other functions. See the code snippet below.

```
UART_HandleTypeDef *gHuart;

void RetargetInit(UART_HandleTypeDef *huart) {
  gHuart = huart;
  /* Disable I/O buffering for STDOUT stream, so that
   * chars are sent out as soon as they are printed. */
  setvbuf(stdout, NULL, _IONBF, 0);
}
```

The printf() routine uses the low-level _write() routine to write data onto the designed port. Hence, the transport will need to modify to the *HAL_UART_Transmit* instead. The relevant code snippet is shown below.

```
int _write(int fd, char* ptr, int len) {
 HAL_StatusTypeDef hstatus;

 if (fd == STDOUT_FILENO || fd == STDERR_FILENO) {
  hstatus = HAL_UART_Transmit(gHuart, (uint8_t *) ptr, len, HAL_MAX_DELAY);
  if (hstatus == HAL_OK)
   return len;
  else
   return EIO;
 }
 errno = EBADF;
 return -1;
}
```

To activate the retarget operation, it is realised by calling the *RetargetInit () routine* in the main.c. See below

```
Main.c
Int main(void){
    ;
    ;
    RetargetInit(&huart3);  // Assign uart3 for printf() to external ttyUSB* terminal
    :
    ;
}
```

3.13.2 Redirect to Serial Wired Viewer (SWV)

Serial Wire Viewer (SWV) [89] is a debugger probe designed by Instrumentation Trace MarcoCell(ITM) [89] for sending software-generated debug messages using a specific signal I/O called Serial Wire Output (SWO). SWV is a high-speed protocol that allows exporting of tracing signals of the CPU's real-time activities. The MsBOT eval board STM32E407F

supports the ITM protocols. Besides displaying traces in real-time through SWO, SWV is used as a software serial port. Debugging messages can be redirected to SWV for display. Not only is it fast but also it eliminates the use of any serial UART ports. To redirect the serial output to the SWV, override the __attribute__((weak)) int _write(int file, char *ptr, int len) with uint32_t _SWD_write(uint32_t file, char *ptr, uint32_t len) and replace the call to __io_putchar with ITM_SendChar. For details, please see the code snippet below.

```
/*
 * SWV_redirect.c
 *
 *  Created on: Oct 20, 2020
 *      Author: wiki-ros
 */
/*******************************************************************************
 * INCLUDES
 *******************************************************************************/
#include "common.h"
#include <unistd.h>

#if !defined (USART3_PRINTF) && !defined(USART6_PRINTF)
/*******************************************************************************
 * @fn      _write
 * @brief   Retarget printf to SWV ITM data console
 * @param       file
 *                              ptr
 *                              len
 * @return      None
 *******************************************************************************/
uint32_t _SWD_write(uint32_t file, char *ptr, uint32_t len)
{
    uint32_t i = 0;
    for(i = 0; i < len; i++)
    {
        ITM_SendChar((*ptr++));
    }
    return len;
}
#endif
```

4 Testing and Results

MsBOT approaches the development activities according to the Plan-Do-Check-Act (PDCA) [92] development cycle from its inception till the end. PDCA is not a single linear process but a circular process which facilitates continuous improvement. The primary purpose of adopting PDCA is because it helps MsBOT development for holistic reflection of the entire development activities by consistent direction alignment, pragmatic development activities, precise test specifications and algorithms.

No development is without issues. Changes in development strategies are not uncommon. Practising PDCA can prevent irreparable misalignment from the original goals. PDCA emphasises precise test specifications and algorithms, which are keys to evaluating the level of success of the development activities. Conversely, convoluted and ambiguous specifications can lead to inaccurate test algorithms, which is consequential to the success of the MsBOT accomplishment. At the end of each development, be it module-level or the entirety of the MsBOT project, performing a reflection on the adopted activities, decisions, and plans can be useful to identify the strengths and weaknesses of the entire aspects of the MsBOT project.

Besides practising the Plan and Do, the Check is also adopted in each development module. The outcome of each MsBOT development module is properly verified against its specifications. As a matter of fact, it is in the planning phase that the system specifications are identified and analysed prior to moving on to the Do phase. It follows by deriving the corresponding test algorithms in accordance with the specifications.

In order to effectively describe the test and verification processes, each module will detail the activities in accordance with the layout, which manifests systematic PDCA elements, namely,

- **Aims** – describes the goals of the development module.
- **Specifications** – specifies the numerical properties of the expectation.
- **Test Algorithms** – details the procedure for performing the test and verification process.
- **Results** – describes the results of the test and verification.
- **Reflection** – reflect on the results, weaknesses or strengths, and suggest potential improvement regimes.

4.1 Bottom-Up Stepwise Development Approach

The MsBOT project involves two broad units, namely MsBOT Component and MsBOT Master. The MsBOT Component adopts eProsima's Micro XRCE-DDS client middleware, while the MsBOT Master adopts the Micro XRCE-DDS agent. Both parts communicate using the CAN-Bus, which is a new approach and is being introduced in the MsBOT project. CAN-Bus development involves not only the MsBOT Component but also the MsBOT Master.

The MsBOT Component implements CAN-bus through the onboard CAN Controller, while the MsBOT Master realises a CAN interface through a USB-CAN adapter. The MsBOT Component employs a Microcontroller circular buffer CAN processing routines; meanwhile, the MsBOT Component employs the Linux kernel's Socket CAN protocols. Figure 14 below shows that MsBOT development modules can be grouped into three categories: CAN-Bus on MsBOT Components, CAN-Bus on MsBOT Master and MsBOT Applications.

MsBOT is bottom-up development that progresses from the physical layer CAN-Bus to the CAN-Bus transport layer and ends with the MsBOT applications. It is a pragmatic approach because the CAN-Bus resides at the physical and the driver layers. If the bottom layers are not working as they should, the development on the upper layer will not reflect the health and the correct behaviour of the MsBOT development.

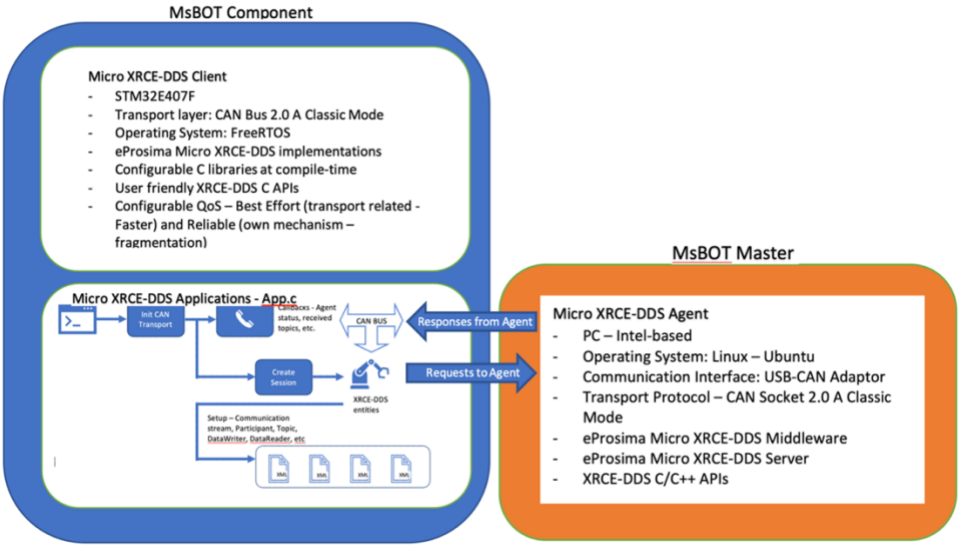

Figure 14 Illustration of the properties of the MsBOT component and the MsBOT master and the processes involved in establishing a DDS entity using the user application, App.c[28, 90].

4.2 The CAN-Bus of MsBOT Component

MsBOT employs Olimex STM32E407F evaluation board which comes with two CAN controllers, namely CAN1 and CAN2. At the initial conditions, the CAN controllers' states are unknown. The development at that time solely relies on the general conditions of the STM32E407F Eval board. In addition, the STM32E407F is new as far as the applications and the programming knowledge are concerned; testing and verification of the results at each stage are inevitable; perhaps it is a must to ensure a high-quality development outcome.

It is assumed that a functional STM32E407F eval board also has a functional CAN controller in a normal situation. However, because the CAN controller is the focal point of the development and was new to us where it needs time to acquire the relevant knowledge, the results of every stage of the CAN-Bus-related development would need to test and validated for its integrity.

In the MsBOT project, the CAN-Bus development involved broadly three sections. Each section is tested and verified for its integrity before progressing to the next. The test and verification methodology is managed using the PDCA approach [86]. The three sections of the CAN-Bus development are:

1. To establish a functional STM32E407F CAN controller.
2. To establish a fully functional communication with the CAN-Bus channel between two MsBOT Components.
3. To establish a fully functional circular buffer for removing the dependency of the upper layer on the lower layer Systems.

4.2.1 To Establish a Fully Functional STM32E407F CAN controller

It is logical to evaluate the Eval board, especially the onboard CAN controller's functionalities while exploring the programming of the CAN controllers. STM32E407 supports many test modes for its CAN controllers. In order to efficiently progress the CAN-Bus development, the Loop-back test mode is a perfect solution. It can provide a functional STM32E407 development platform. Still, it also provides a functional CAN controller, which results from the effort of evaluating and learning program the STM32E407 in general and the CAN controller in specific.

Figure 15 below shows the CAN controller's Loop-back test mode internal construction. The Loop-back test mode is a self-test function independent of any external peripherals. It is a great condition to begin the development by making sure the CAN controller in specific or the Olimex STM32E407F Eval board works properly without any influence of external factors.

In addition, the mode omits the acknowledged errors, which further eases the setup activities. In accordance with the wirings in Figure 15 below, the Loopback test mode performs internal feedback from its Tx output to its Rx input. While in the mode, the incoming CAN RX pin is disconnected, resulting in its incoming CAN Rx data not reaching the Rx FIFO buffer. In other words, the incoming Rx Data cannot influence the conditions of the transmitting Tx values. However, since the CAN Tx pin is still connected to the Tx, the developer can probe its transmitted signals on the CAN Tx pin.

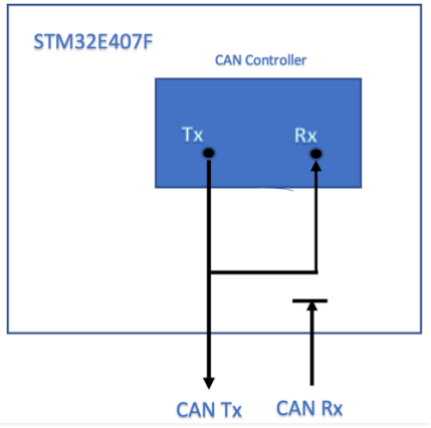

Figure 15 shows the internal construction of the Loopback test mode

Aims - A functional STM32E407F CAN controller

Specifications - The feedback Rx CAN frame matches the outgoing CAN Tx frame.

The CAN controller is considered functional when both the aimed conditions are fulfilled, i.e., the CAN controller has been programmed properly for sending CAN frames, and the outgoing CAN frames are redirected back to the input buffer for reading. Matching the incoming frames with respect to the outgoing frames manifests the operational status of the CAN controller.

Algorithms – Set up the STM32E407F's CAN controller for the loopback test mode.

Program the CAN-Bus controller to perform the loopback test mode. Send out a fixed CAN frame continuously. Read and match each incoming frame with the outgoing frame.

Results – The incoming frames match the outgoing frames. It is a passed.

The results show that the STM32E407F and its CAN controller are working properly despite the operating mode where the CAN controller is running is not the final setting. However, it resolves two major issues. Firstly, a functional STM32E407 Eval board provides a stable development platform for MsBOT development activities. Secondly, the CAN controller programming is correct for getting the CAN controller to operate in the intended functionalities.

Reflection - The loopback test mode kills two birds with one stone by resolving the fundamental needs, i.e., stable development progress and efficient learning curve. Despite other test modes available, the loopback test mode was adequate for two reasons, simplicity and time reduction. The loopback test mode is simple and easy and does not involve any external peripherals. The test mode can be realised with the naked STM32E407F Eval board itself. Besides, as a bonus of the test mode. If the real CAN signals probe is required, the signal can be probed at the CAN Tx pin. It can be used to validate the operation of the CAN controller and learn the characteristics of the CAN Tx signals prior to the next development activities. Another advantage of the setting is allowing the Philips CAN transceiver module, namely TJA1050, can conveniently be plugged in for validating its operation status, which otherwise will need another round of hardware and software setting and validation.

4.2.2 To Establish a Fully Functional Communication CAN Bus Channel Between Two MsBOT Components

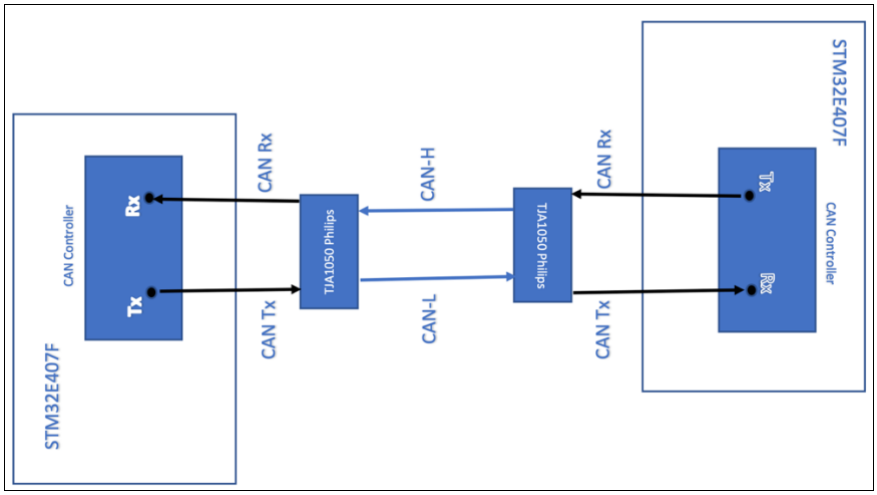

Figure 16 A Full Functional CAN-Bus Test Construction using two STM32E407F Eval boards

A fully functional CAN Bus communication channel is absolutely a must for MsBOT. It serves as the physical layer of the MsBOT system, which the CAN transport relies on. Despite the STM32E407F and the CAN controller being proven to work in the previous section, some known issues still need to be cleared. Firstly, the programming model and know-how for sending out and receiving CAN frames for bi-directional communication at the intended bitrate. Secondly, the newly added components, namely Philips TJA1050, are CAN transceivers. The CAN transceiver converts CAN-Tx and CAN-Rx digital signals to CAN-H and CAN-L differential voltage signals for highly reliable transferring data through a longer distance and electromagnetically noisy environment. Figure 16 above shows two MsBOT Components connected through CAN-Bus, which is realised using the CAN controllers at each end. However, it does not show any terminal resistors, despite being installed throughout the MsBOT project. CAN bus communication is bidirectional, in which a terminal resistor at each end absorbs the CAN signal energy. Otherwise, the signals get reflected from the cable ends. The standard termination device for a CAN bus is a 120 Ohm resistor.

There are two modes STM32E407F CAN controller can operate on, i.e., the polling method and the interrupt service routine (ISR) method. The polling is much easier to realise than the ISR method. The polling mechanism can inherit the work of the previous programming setting, i.e., the Loopback Test mode and modify it by simply reconfiguring the relevant CAN controller control bits. In contrast, the ISR method needs, besides reconfiguring the CAN controller control bits but also the programming of the ISR function. The programming of the ISR method can be challenging for users who are new to or not familiar with STM32E407F and its CAN controller. In order to increase the success rate and reduce the time on unproductive debugging activities, the MsBOT took two progressive

steps. Firstly, implementing the polling method. Once succeeded, only then proceed to implement the ISR method.

MsBOT employs CAN standard Classic 2.0A mode for the entire project despite the STM32E407F Eval boards supporting two modes, the standard CAN Classic and the extended mode. The CAN standard classic frame has 11 bits identifier (CAN 2.0A), while the extended is identical except it has a longer identifier, i.e., 29-bit. The extended CAN frame is classified as CAN 2.0B. The corresponding CAN standard classic frame architecture is shown in Figure 17 below.

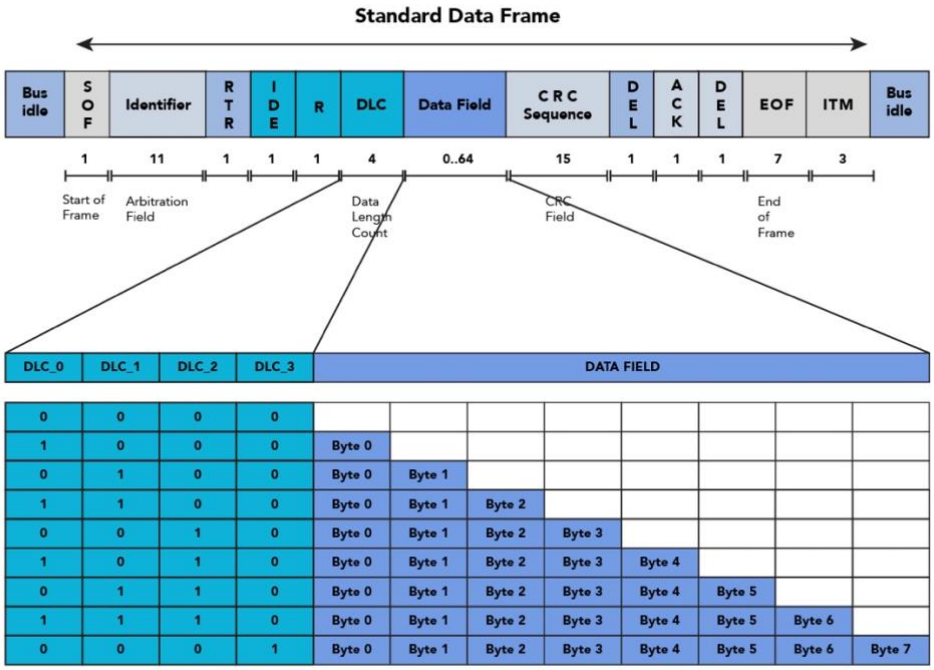

Figure 17 The Standard Classic Mode CAN Frame Architecture

The standard Classic mode CAN frames start with a “Start of Frame” bit (SOF), followed by the message identifier and the “Remote Transmission Request” bit (RTR). The RTR bit differentiates between a Data Frame and a Remote Frame, RTR = 0 represents Dominant in a data frame, and RTR = 1 represents Recessive in remote transmission request. Depending on the frame data length, which is represented by the 4-bit DLC. The table in Figure 17 shows that 4-bit values of 0000 are zero data length, 0001 is 1 byte, 0010 is 2 bytes, and 0100 is 8

bytes data frame accordingly. RTR = 0; DOMINANT in data frame RTR = 1; RECESSIVE in remote frame

4.2.2.1 The Polling Method

The polling method can be achieved using the previous Loopback test mode without difficulties in programming, except for the configuration of CAN control bits for achieving the Classic CAN 2.0 mode, which runs at 500kbps. The polling method eases debugging because the transmission and reception of CAN frames are under the programmer's control. In addition, the programming settings are easy by comparison to the ISR method. The advantages help to keep MsBOT development on schedule. From a time, management perspective, saving time on setting and debugging activities implies more time and effort can be spent on the CAN transceiver TJA1050 module construction. Having a stable construction and consistent performance system is paramount for the next development activities and the entire MsBOT development in general. Otherwise, reconstructing and redesigning can easily break and miss the schedule.

Aims – A stable construction and consistent performance end-to-end CAN-bus communication channel – The physical layer

Specifications – The contents of the incoming CAN frame at A STM32E407F Eval board match the outgoing CAN frame sent out by B STM32E407F Eval board. Both the systems were set to run at 500kbps bitrate. Figure 16 shows two STM32E407F Eval boards, namely A and B. The boards use their respective CAN controller for realising the CAN interface. In addition, both the CAN controllers are connected through two TJA1050 CAN transceivers—one at each end of the CAN controller. The test aims to validate that the construction of the CAN-Bus communication channel is at stable working conditions. In addition, it tests the correct use of the control bits for configuring the Classic mode running at bitrate 500kbps. To differentiate the source of the signals, node A transmits 0x5A data, while node B transmits 0xA5 data. Once the communication has been established, run the established communication continuously for some time to validate the consistent performance of the CAN-Bus.

Algorithms – Construct the system in accordance with the wiring in Figure 16. Program the CAN controllers with the relevant control bits to set up the standard Classic mode and run at 500kb/s bitrate. Sends out a fixed frame continuously while continuously polling readings from the FIFO0 buffer and matching the CAN frame of the fixed outgoing CAN frame from the other CAN controller. For the details of the control bits settings, please refer to the previous chapter, i.e., Methodology.

Results – The matching processes show that the incoming CAN frames match the fixed outgoing CAN frames. Probe readings at CAN-H and CAN-L of Node show a full operation of both the CAN controllers, i.e., A and B. Figure 18 and Figure 19 show the signals captured during the trouble-shooting phases. Upon the two being connected, both sides received frames from the corresponding transmitting end. They probed for the CAN-TX and CAN-RX signals before the transceiver and the CAN-H and CAN-L signals after the transceiver. Figure 18 shows the CAN_H and the CAN_L signals of node A, sending out ID 0x321 and0xA5 data frames. Both the CAN-H and CAN-L at the A node are at a bitrate of 500kbps. Figure 19 shows the CAN_H and the CAN_L signals of node B, sending out frames ID 0x321 and 0x5A data. The CAN-H and CAN-L at the B node are also at a bitrate 500kbits/sec.

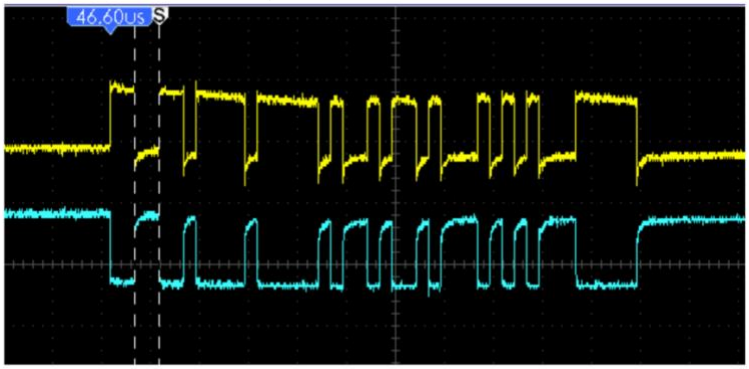

Figure 18 The CAN-H and CAN-L signals of node A

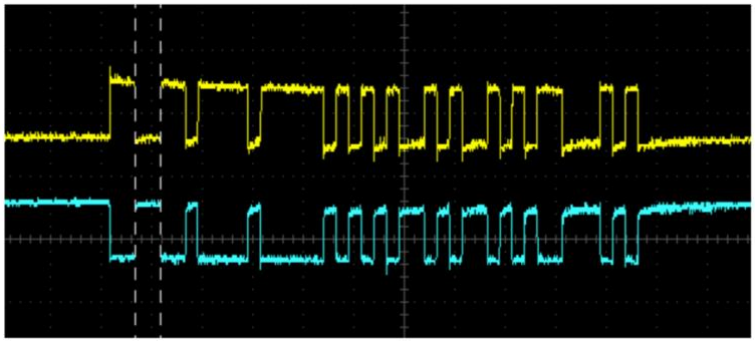

Figure 19 The CAN-H and CAN-L of node B

Reflection – The MsBOT's end-to-end communication was established successfully, a manifestation of STM32E407F evals boards, the CAN controllers and its intended programs are in correct working conditions. The transmitting frames were successfully received consistently, another indicator of positive progress and consistent performance. Despite the CAN bus running at 500kb/s, it is only a provisional and arbitrary bitrate to begin with the development. However, it is set to run at a different bitrate to meet the new system requirements. When in need, CAN Classic mode operates on its maximum bitrate of 1Mb/s.

The interrupt method is the most practical operation mode for MsBOT. It removes the wastage of processing power, but it also improves overall system performance. The bitrate reading and the messages in the Callback routine manifest that the setup, both at the hardware and the software aspects, is stable. It is ready for the next level of development. Despite the CAN running at 500kb/s, it is only provisional. It can be set to a different bitrate at ease when in need. However, the maximum bitrate CAN Classic mode operates on is 1Mb/s. Due to the simplistic nature of the current system, the transmitting action is not done through the interrupt method but through direct execution, sending out the CAN frame sequentially. Nonetheless, it can also adopt the interrupt method when in need. When in need, CAN Classic mode operates on its maximum bitrate of 1Mb/s.

Polling mode is not a realistic operation mode because it is a rigid algorithm and consumes a lot of processing power. Much time and effort are spent on polling operations. However, it is a necessary and worthwhile step for learning about the characteristic of the CAN controller and the programming know-how. Besides, the polling operation simplifies the debugging algorithms as fewer variables are considered. It needs not to consider if other conditions may cause the transmission

and reception to fail. In contrast, focus on one variable, i.e., the reception for the receiver or the transmission for the transmitter.

Further improvisation is needed, i.e., operate in interrupt mode. In the interrupt mode, the polling routine is omitted and replaced with an interrupt routine, resulting in the CPU having more resources for other processing.

4.2.2.2 The Interrupt Method

The Interrupt method adopts the interrupt service routine (ISR) operating mode in MsBOT. Upon receiving a complete CAN frame, the CAN controller triggers an interrupt request that is subsequently picked up and passed the control to the ISR routine. In fact, the interrupt routine plays an important role in handling incoming CAN frames, i.e., disassociating the reading FIFO0 task from the CAN frame processing task. The ISR must mainly read the complete CAN frame from the designated buffer, i.e., FIFO0 and store the frame in the circular buffer. The applications at the upper layer can read the frames at a different pace or at different time when processing resources are available. With this feature, applications can perform a non-blocking I/O reading of the CAN frames stored in the circular buffer. The feature frees the central processing unit (CPU) from being tight to a particular task. In contrast, it allows CPUs to perform additional tasks such as displaying quick debugging messages, switching ON/OFF other control bits, etc.

MsBOT project can realise the interrupt mode relatively easily by taking advantage of the previously completed polling settings. It involves little changes at the control bit registers and adding a new ISR routine. The polling software routines are replaced with an ISR routine. Upon receiving a CAN frame, the system initiates an interrupt request, which gets picked up and then transfers the control to the ISR routine. Clearing the interrupt request bits allows interrupting to reoccur, even if it is in the middle of ISR. It is common to switch off an interrupt request at the beginning of an ISR and switch it back on before leaving the ISR.

STM32E407F has a different way of programming ISR. Instead of ISR, STM32E407F comes with a predefined ISR for the CAN controller specifically meant to read the CAN frame from the FIFO0 HAL_CAN_RxFifo0MsgPendingCallback. However, STM32E407's instructions advise users not to modify the code. Instead, users implement it in the user's file.

At times, this new way of ISR can be dreadful for new users to comprehend. Instead of ISR, STM32E407F uses Callback through notifications. It is a higher-level implementation without needing users to know ISR details. To implement a needed ISR, users need to

identify the needed characteristic and choose the predefined Callback routine. Users can opt for implementing a user-defined callback routine by simply creating a new callback routine of the same name in the user's file. In user's code, user can activate or deactivate the notification using HAL_CAN_ActivateNotification() and HAL_CAN_DeactivateNotification() function respectively. The callback of interest is HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan) which is being implemented in MsBOT. Please refer to the below code for details.

```
/* NOTE: This function Should not be modified, when the callback is needed,
        the HAL_CAN_RxFifo0MsgPendingCallback could be implemented in the
        user file
  */

void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
        CAN_RxHeaderTypeDef rxHeader;
        uint8_t rcvd_msg[8];
        uint8_t arr_val [4];
        int i = 0;

#ifdef DEBUG_WITH_PRINTF_V3
        printf("\n\rHAL_CAN_RxFifo0MsgPendingCallback() entered \n\r");
#endif
        if (HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &rxHeader,rcvd_msg) != HAL_OK)
        {
                printf("\n\rHAL_CAN_RxFifo Callback - HAL_CAN_GetRxMessage() Failed\n\r");
                ISR_Error_Handler();
        }

#ifdef DEBUG_WITH_PRINTF_V3
        if ((rxHeader.IDE == CAN_ID_STD)) // && (rxHeader.StdId == 0x321))
        {
```

```
                    //printf("\n\rID = 0x%x RdDataLength %d , RxData 0x%x\n\r", (int)(rxHeader.DLC),
(unsigned int)rcvd_msg[0]);

                    printf("\n\r ID = 0x%x RdDataLength %d \n\r", (int)rxHeader.StdId, (int)(rxHeader.DLC));

                    /* print received data */

                    for (i=0; i< rxHeader.DLC; i++)

                    {

                              printf("[%d]=0x%x ", i, (int)rcvd_msg[i]);

                    }


                    printf("\n\r");


          }

#endif

          /*

           * Store the CAN frame in this structure only

           * i.e., | rxHeader.StdId (4 bytes) | rxHeader.DLC (4 bytes) | data (4 bytes) |

           *

           * Note: the term "element" depicts data type of uint32_t

           */

          if ((rxHeader.IDE == CAN_ID_STD) && (rxHeader.StdId == 0x120))

          {

                    EnqueueFrame(rxHeader.StdId, rxHeader.DLC, rcvd_msg);

                    printf("\n\rcir buf after count %d \n\r", circularByteBuffer_Count(&cb_han));

          }

          else {

                    printf("\n\r NON-0x120 ID = 0x%x RdDataLength %d \n\r", (int)rxHeader.StdId,
(int)(rxHeader.DLC));

          }

          HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);

}
```

Aims – A stable construction and consistent performance end-to-end CAN-bus communication channel using the ISR Callback routine.

Notes: The construction of CAN-Bus forms the physical layer while the ISR callback routine form the Data-Link layer of the MsBOT system.

Specifications – The contents of the incoming CAN frame at A STM32E407F Eval board match the outgoing CAN frame sent out by B STM32E407F Eval board. Both systems run at 500kb/s bitrate.

Algorithms - Implement the above callback, i.e., *HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)* routine in the user's file. Be reminded to enable the notification us*ing the HAL_CAN_ActivateNotification(&hcan1,CAN_IT_RX_FIFO0_MSG_PENDING)* function in the main routine() as shown below. For reading the CAN frame, use the API *HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &rxHeader,rcvd_msg).* In the end, to run it on both systems.

```
Int main() {
        ;
        ;
         HAL_CAN_ActivateNotification(&hcan1,CAN_IT_RX_FIFO0_MSG_PENDING);
        ;
        ;
}
```

Results – The incoming CAN frames at B-node match the outgoing CAN frames from the A-node. The two were connected, and both sides received frames from the corresponding transmitting end. The probe readings at CAN-H and CAN-L of A-Node manifest a full operation of both the CAN controllers, i.e., A and B run at 500kb/s bitrate. Figure 18 and Figure 19 show the signals captured during the full-running operation. The CAN-TX and CAN-RX signals were probed before the transceiver; the CAN-H and CAN-L signals were probed after the transceiver. Figure 18 shows the CAN_H and the CAN_L signals of node A, sending out frames of ID 0x321 and 0xA5 data.

Below are the operating status messages displayed in the respective Callback routines through the retarget printf() function.

Notes: Node A: messages in the Callback routine show that it has just entered the callback routine. The received CAN frame's ID is 0x123, and the data length is 1 unit of 0xA5.

HAL_CAN_RxFifo0MsgPendingCallback() entered

ID = 0x123 RdDataLength 8

Data[0]=0xA5

Enqueue id = 0x123 [0]=0x23 [1]=0x1 `2`=0x0 [3]=0x0

Enqueue DCL = 0x1 [0]=0x01

Enqueue data[0]=0xA5

Notes: Node B: messages in the Callback routine shows that it has just entered the callback routine. The received CAN frame's ID is 0x321, and the data length is 1 unit of 0x5A.

HAL_CAN_RxFifo0MsgPendingCallback() entered

ID = 0x321 RdDataLength 8

Data[0]=0xA5

Enqueue id = 0x321 [0]=0x21 [1]=0x3 `2`=0x0 [3]=0x0

Enqueue DCL = 0x1 [0]=0x01

Enqueue data[0]=0x5A

Reflection – The interrupt method is the most practical operation mode for MsBOT. It removes wastage of processing power resulting in improving overall system performance. The bitrate reading and the messages in the ISR Callback routine manifest that the setup both at the hardware and the software aspects, is stable. Unlike other Microcontrollers' ISR routines, STM32E407F implements ISR using a callback routine. As such, don't get confused with the callback routines in high-level programming, e.g., Microsoft Windows programming.

It is now ready for the next level of development. Despite the CAN running at 500kb/s, it is only provisional. It can be set to a different bitrate at ease when in need. However, the maximum bitrate CAN Classic mode operates on is 1Mb/s. Due to the simplistic nature of the current system, the transmitting action is not done through the

interrupt method but through direct execution, which sends out the CAN frame sequentially. Nonetheless, it can also adopt the interrupt method when in need.

In facts, the interrupt method is harder to implement, especially true for newcomers. The developer must be well-versed with the board system information and the board programming implementation, and the development tools, including the programming languages, etc. Debugging in an active interrupt environment can be challenging because single-step and break within ISR may pose risk in breaking the program. More attention and care are needed.

4.2.3 To Establish a Functional Circular Buffer for Disassociating Dependence of Upper Layer to Lower Layer Systems

The circular buffer plays an integral role in handing unsynchronous incoming data streams, particularly the MsBOT incoming CAN frames. Like other communication systems, the MsBOT CAN frame comes unannounced. The polling method for handling the CAN frame is impractical, producing unnecessary processing wastage. Hence, it causes overwhelming stress to the overall system performance.

The interrupt method resolves some parts of the former issues. It needs a way to break the upper layer's dependency on the bottom layer—a circular buffer response to the issues. The circular buffer as intermediate storage allows the interrupt method to store data read from the FIFO0 and disassociate the lower layer routine from the frame processing routine at the upper layer. The intermediate storage is implemented in a manner behaving like a circle. Hence, given the name circular buffer. Unlike linear First-In-First-Out queue data structures, circular buffers do not have a head and a tail. Instead, it has the in-marker and the out-marker. The circular buffer is empty when the in-marker equals the out-marker plus one. When the in-marker equals to out-marker, the circular buffer is full.

MsBOT implements the size of a circular buffer about 200 times the maximum length of a Classic CAN frame to prevent overflow of incoming frames. In a nutshell, during the Callback routine, CAN frame read from the FIFO0 is being enqueued onto the circular buffer. While at the upper layer, the retrieving of CAN frames from the circular buffer is called dequeue. Both are storing and retrieving are performed in complete frames. Incomplete CAN frames are considered invalid, therefore, are being disposed of.

Aims - To establish a fully functional circular buffer for upper-layer processing.

Specifications – The retrieved frames at the upper layer match the stored frames in the bottom layer.

Algorithms – To make use of previous successful interrupt systems. A-Node repeatedly sends out a fixed frame, while B-Node in the Callback routine “enqueue” the frame - stores into the circular buffer - read from the FIFO0. At the upper application layer, it retrieves the CAN frames and compares them to the stored frame if they are the same.

Results – The retrieved frame matches the stored frame. The first segment of debugging messages below shows in the Callback routine. It shows it received the frame of CAN ID 0x123 and a data length of 4 bytes. It stored the frame through the "enqueue" function onto the circular buffer. The second segment of debugging messages below shows the reading function, cb_read_CAN_frame() was in action. There were 312 bytes in the circular buffer. These numbers show that there have been many calls of the Callback routines. Hence, numerous frames have been stored. These scenarios show that the disassociation does work successfully. It has caused a disassociation of the dependence of the upper layer and lower layers. The contents of the CAN frame match the contents of the stored frame. The following shows evidence of the successful ISR callback routine where has been entered and performed the CAN frame reading from the FIFO0. It has stored a frame of ID 0x123 and a length of 4 bytes.

/* Debugging messages while executing the Callback routine */

HAL_CAN_RxFifo0MsgPendingCallback() entered

ID = 0x123 RdDataLength 4

Data[0]=0x0 Data[1]=0x0 Data`2`=0xfc Data[3]=0xf

Enqueue id = 0x123 [0]=0x23 [1]=0x1 `2`=0x0 [3]=0x0

Enqueue DCL = 0x4 [0]=0x4 [1]=0x0`2`=0x0 [3]=0x0

Enqueue data[0]=0x0 [1]=0x0 `2`=0xfc [3]=0xf

The following shows that the upper layer reads the CAN frame stored by the ISR Callback routine above. It has read a frame of ID 0x123 and a length of 4 bytes

./* Debugging messages when reading CAN frames at the uppper layer */

cb_read_CAN_frame() entered

circularByteBuffer length 312

CAN ID, 0x123

CAN DLC, 0x4

CAN payload: data[0] = 0x0 data[1] = 0x0 data`2` = 0xfc data[3] = 0xf

Reflection – Like other communication, incoming data streams arrive asynchronously at the interval a receiver would not expect. A common resolution is implementing a dedicated ISR for the asynchronous incoming data streams. It frees the programmer from introducing a resolution to handle incoming data streams while processing user applications. In the worst scenario, the data arriving at a very rapid interval can significantly impact overall system performance.

Circular buffers or any other intermediate storage can remediate the issue above. The MsBOT circular buffer serves the situation well by disassociating the dependence of the upper layer processing on the lower upper callback routine. The circular buffer size needs to be further fine-tuned to optimise the storage usage versus allocation rates to reduce unnecessary wastage.

However, data buffers for communication systems are implemented on byte-level data streams. Regardless of data sizes, be it 16-bit or 32-bit, they are serialised to individual bytes when transmitting and are deserialised up reception. Since the STM32E407F Microcontroller is of 32-bit data where a native integer is 32-bit, the integer value needs to serialise to 4 individual bytes. The incoming data needs to be deserialised before passing it on for further processing. The MsBOT circular buffer routines handle the serialisation and deserialisation without causing unnecessary stress on the system or the upper layer.

Spinlock is used in the MsBOT to prevent the common data from being "stolen" by other threads. The common data, such as the buffer count and the data in the buffer, are accessible by the ISR and the upper layer. It may be suddenly accessed and updated by the upper layer reading in the middle of storing the buffer count by the ISR routine, resulting in an erroneous buffer count. The MsBOT spinlock causes the ISR to lock the buffer count and the buffer content from being accessed by the upper layer reading operation. The same applies to the upper layer reading operation. Having obtained the lock, it blocks the ISR routine from accessing the buffer counter

and the buffer itself. As soon as they finish the read or write operation, they release the spinlock allowing the other thread to own the spinlock.

Despite the benefit of the interrupt method implemented for receiving the CAN frames, it is not being implemented on the CAN transmitting frames because the overall processes of the transmitting activities are relatively simplistic. However, it is only provisional. When in need, the transmitting interrupt can be implemented at ease. In addition, the corresponding circular buffer of a similar manner will also be implemented accordingly.

4.2.4 To Establish CAN-Bus Communication channels between the STM32E407 MsBOT Component and the PC MsBOT Master

Besides MsBOT Component, MsBOT Master is another integral part of MsBOT. Without the MsBOT Master, MsBOT Components could not communicate with one another. They cannot operate in a collective system. The MsBOT Master hosts the Micro-ROS Agent, which acts as a server for the MsBOT Components, which also hosts Micro-ROS Clients to communicate among themselves or the DDS entities in the DDS world.

Like the communication channel amongst MsBOT Components, MsBOT Master communicates with MsBOT Components through CAN-Bus. However, unlike MsBOT Components which run on a STM32E407F Micro-controller, MsBOT Master runs on a Linux Ubuntu variant. The most convenient way of setting a CAN-Bus interface on PCs is using a USB-to-CAN adapter. Figure 20 below shows the construction of CAN-Bus of MsBOT Master and MsBOT Component. The MsBOT Master employs a USB-to-CAN adapter to realise a CAN-Bus interface.

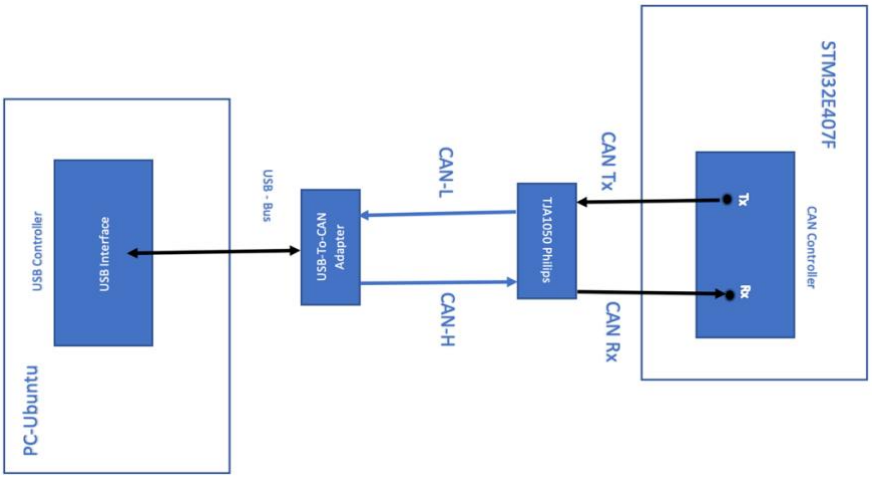

Figure 20 CAN-Bus construction of MsBOT Master and MsBOT Component

4.2.4.1 To Establish the CAN-Bus Communication channels through the USB-to-CAN adapter's commands

Aims - To establish a fully functional communication between the MsBOT and the MsBOT Master through the USB-to-CAN adapter's commands.

Specifications – The received frames match the sent frames.

Algorithms – It uses previous successful interrupt systems of the MsBOT Component, sending out frames it receives back to the MsBOT Master. The MsBOT Master repeatedly sends out a fixed frame using the USB-to-CAN commands in a Linux terminal. The received CAN frames are compared with the sending frames at a different terminal set as a frame receiver.

The USB-to-CAN adapter in the market has many different CAN-Bus implementations. Generally, they are broadly classified as non-SocketCAN or SocketCAN systems. For MsBOT, the USB-to-CAN adapter must be SocketCAN compliant in order to utilise the SocketCAN package in the Linux system. The advantages of SocketCAN are that the CAN protocol uses the Berkeley socket API, and the Linux network stack and its CAN device drivers are implemented as network

interfaces, resulting in it having a set of command-line commands resembling the network interfacing commands. For instance, MsBOT set up the USB-to-CAN adapter with the following commands.

1) Set can0 at bitrate 125000bps

#sudo iplink set can0 type can bitrate 125000

2) Activate can0

#sudo ifconfig can0 up

3) Set can0 as a receiver

#candump can0

4) Send out a CAN frame through can1 of CAN-ID 0x123 and Data length = 5 and data of 0x12 0x34 0x56 0x78 0x90

#cansend 0123#1234567890

At the terminal where can0 is an active receiver, it receives a CAN frame of:

CAN-ID 0x123

Data length = 5

Data = 0x12 0x34 0x56 0x78 0x90

Results – The received CAN frame match the corresponding transmitted frames.

Reflection – This is a pragmatic approach to establishing the CAN-Bus communication channels. It began with establishing a stable working MsBOT Component and applied a new variable, the USB-to-CAN adapter. The positive results endorsed that the CAN Socket would work, and the choice of USB-to-CAN adapter was correct because the underlying technology of the USB-to-CAN adapter is CAN Socket. It is a paramount milestone, not only it verifies the previous work on the MsBOT Component, but also it provides confidence that the CAN Socket is a right choce. Nontheless, it denotes a clearer pathway ahead.

4.2.4.2 To Establish the CAN-Bus Communication channels through CAN Socket

The CAN Socket APIs were inspired to mimic TCP/IP protocols to ease the learning curve. It enables programmers who know the TCP/IP protocols will find CAN Socket APIs very familiar and easy to learn.

Aims - To establish a functional communication between the MsBOT and the MsBOT Master through CAN Socket.

Specifications – The received frames match the sent frames.

Algorithms - The following shows the MsBOT adopts CAN Socket APIs and the applications of CAN Socket APIs in various parts of MsBOT Master.

i) Creating the CAN socket, any errors that occur result in the return of value -1.

```
/* Specify can0 device */
Strcpy(ifr.ifr_name, "can0");
Ret = ioctl(sck, SIOCGIFINDEX, &ifr);
If (ret < 0) {
        perror("ioctl interface index match up failed!");
        return 1;
}
```

ii) Binding the CAN socket to "can0" device

```
/* Bind the CAN socket to can0 */
addr.can_family = PF_CAN;
addr.can_ifindex = ifr.ifr_ifrindex;
ret = bind(sck, (struct sockaddr *) &addr, sizeof(addr));
```

```
If (ret < 0) {
        perror("binding failed!");
        return 1;
}
```

iii) The CAN filtering rules allow users to block unwanted CAN frames based on their frame pattern. Disable filtering rules set the applications as a sender only, not able to any incoming frames.

```
/* Disable filtering rules, i.e., set as a sender, not receiver */
setsockopt(sck, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
```

iv) Construct a CAN frame prior to transmitting

```
/* Construct a CAN frame */
frame.can_id = 0123;
frame.can_dlc = 8;
frame.data[0] = 1;
frame.data[1] = 2;
frame.data`2`= 3;
frame.data[3] = 4;
frame.data[4] = 5;
frame.data[5] = 6;
frame.data[6] = 7;
frame.data[7] = 8;
if (! (frame.can_di&(CAN_EFF_FLAG))
        printf("Standard Classic frame!\n");
else
        printf("Extended frame!\n");
```

v) Sending CAN frame to the CAN Bus. Use the return value of write() to check whether all data has been sent successfully

```
/* Send CAN frame out */

nbytes = write(sck, &frame, sizeof(frame));

If (nbytes != sizeof(frame)) {

        Printf("write() frame incomplete!\n");

    }
```

vi) Reading CAN frames from CAN Bus

```
Nbytes = read(sck, sizeof(frame));
```

vii) Close the can0 device and disable the CAN Socket

```
/* Close can0 through its handler */

Close(sck);
```

Results – The received CAN frame match the corresponding transmitted frames.

Reflection – By far, the CAN-Bus communication channel between the MsBOT Component and the MsBOT Master has been established using the command line approach and the CAN Socket approach. Its advantage is that the command-line approach needs no programming. Users can key in the above commands to set up the USB-CAN adapter. Hence, it is a good choice for a new setup that needs quick validation of the well-being of the setup.

Eventually, the system would need to integrate with the CAN Socket. Besides, the MsBOT transport is CAN-Bus which cannot work autonomously with the command line method. In the MsBOT Master, the CAN Socket was integrated into the transport layer of the Micro-ROS Agent application. A test and verification are needed to validate the relevant code.

In summary, the integration of CAN Socket into the MsBOT Master was carried out very effectively and efficiently. Firstly, it began with the command-line approach to quickly validate the overall performance of the hardware construction and the CAN Socket APIs – CAN Socket Command-line is implemented using the CAN Socket APIs. The results of the commands will reveal the well-being of the MsBOT Master construction. Secondly, the application of CAN Socket in the Micro-ROS Agent program, namely MicroXRCEAGent.cpp. However, the test and verification are performed only at the transport layer or the code where CAN Socket is applied at this stage. The test and verification scheme omitted to perform on the layers above the CAN Socket because, at this stage, it aimed to test the integration of the CAN Socket APIs. Lastly, to perform the test and verification to the entire CAN-enabled Micro-ROS Client program, namely app.c, at the application layer in conjunction with the Micro-ROS Agent, namely MicroXRCEAGent.cpp. As a reminder, MsBOT Components, which hosts the Micro-ROS Clients, needs the Micro-ROS Agent residing in MsBOT Master to work as a complete MsBOT system.

Integrating CAN Socket into Micro-ROS Agent was a painstaking task. It involved massive code research and understanding. Tenth of thousands, if not the millions, line of code needs to understand to some extent. Uncertainties and helplessness were the words to describe the experience in the progress of development. Nevertheless, through effective strategies – planning and discussion, and perseverance and optimism, the CAN transport was successfully integrated into MicroXRCEAgent.cpp program.

4.2.5 To develop the user applications that utilise the CAN-Bus Communication channels between the STM32E407 MsBOT Component and the PC MsBOT Master

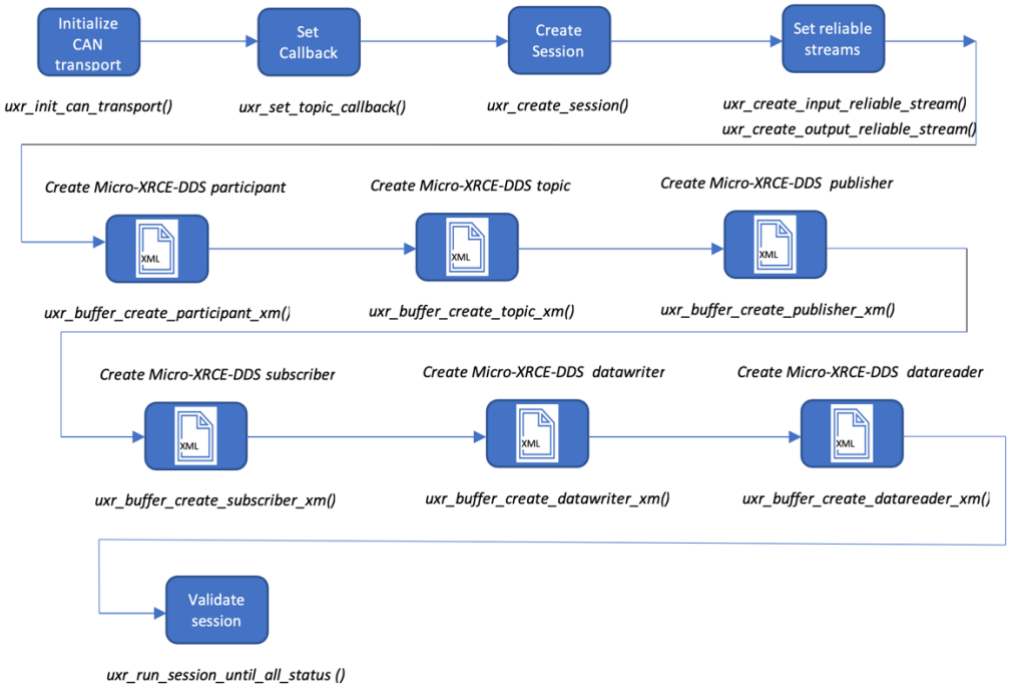

Figure 21 illustrates the conditions and the processes involved in establishing a MsBOT DDS entity

The user application app.c[90] was written to perform as a Micro-ROS Client. It is through the user application app.c a DDS entity is established. However, prior to establishing a DDS entity, there are a few conditions to fulfil. Figure 21 above illustrates the conditions and the processes involved in establishing the MsBOT DDS entity. Firstly, the underlying transport must be established, i.e., the CAN bus for the MsBOT project. It transports the data between the MsBOT Component and the Master. Secondly, it must succeed in creating a callback service routine in response to responses from the Micro-ROS Agent. A session must be created which analogous to a communication channel between the MsBOT Component and the MsBOT Master. It follows by setting up reliable streams, participant, topic, publisher, subscriber, DataReader and DataWriter. The final step is to validate the quality of the communication streams till all the statuses are positive.

Aims – To establish a functional CAN-bus communication channel between a MsBOT Component and the MsBOT Master at the transport layer and the user application layer.

Specifications –

Stage 1 – Using the command line method, the MsBOT Master can send and receive CAN frames from the MsBOT Component. Respectively, the MsBOT Component can receive and send CAN frames to and from the MsBOT Master. The frames received match the sent frames from the other end.

Stage 2 – With the CAN Socket method, the MsBOT Master can receive CAN frames from the MsBOT Component. The frames that arrive in MsBOT Master match the frames sent from the MsBOT Component. As the matter of fact, this is the test of the CAN transport layer established using the CAN Socket.

Stage 3 – Once the CAN transport has succeeded, the next test is the newly implemented transport layer. If the new transport works properly, the user's application should run properly till the end.

Algorithms –

Stage 1 – Construct the system in accordance with Figure 5. On the MsBOT Component, run the previous interrupt method program, sending out CAN frame of ID = 0x123 and of DLC = 0x4, of data 0x12, 0x34, 0x56, 0x78. On the MsBOT Master, open a terminal, run the following commands:

1. Set up the can0 device at a bitrate of 500kbps.

   #*sudo iplink set can0 type can bitrate 500000*

   #*sudo ifconfig can0 up*

2. Set up the can0 as a receiver

   #*candump can0*

   Notes: Ctrl + C to exit the receiver mode

3. Set up the can0 as a transmitter and send out CAN frame of CAN-ID 0x123 and Data length = 4 and data of 0x12 0x34 0x56 0x78

#cansend 0123#12345678

Stage 1 - Stage 2 is performed only after stage 1 has succeeded.

Construct the MsBOT Master's transport layer using the CAN Socket APIs. Please refer to CAN-Bus transport implementation in the Methodology chapter for details. Without hardware any changes on MsBOT Component, reuse the program to send out a fixed CAN frame of ID = 0x123 and of DLC = 0x4, of data 0x12, 0x34, 0x56, 0x78. On the MsBOT Master, through the new CAN Socket CAN transport layer, receive a fixed CAN frame of ID = 0x123 and of DLC = 0x4, of data 0x12, 0x34, 0x56, 0x78.

Stage 3 - Stage 3 is performed only after stage 2 has succeeded.

Run the newly CAN-Bus integrated app.c user applications in MsBOT Component. The MsBOT Component communicated with the MsBOT Master using Micro XRCE-DDS through the CAN transport. Through the Micro-ROS Agent residing in the MsBOT Master, the Micro-ROS Client of the MsBOT Component uses the Micro-DDS-XRCE protocol to establish itself as a DDS client communicating with the DDS global data space.

Results –

Stage 1

The MsBOT Master was set to the receiving mode, which was activated by the command line #candump can0. It received frames the CAN frame sent from the MsBOT Component. The following shows the CAN frame and its attributes.

CAN-ID 0x123

Data length = 5

Data = 0x12 0x34 0x56 0x78 0x90

The MsBOT Master was set to the transmitting mode achieved by the command-line #cansend 0123#12345678, and in the MsBOT Component, it received the CAN frame sent by the MsBOT Master using the command line before. The following shows the CAN frame and its attributes.

HAL_CAN_RxFifo0MsgPendingCallback() entered

ID = 0x123

RdDataLength 4

Data [0]=0x12 [1]=0x34 `2`=0x56 [3]=0x78

Enqueue id = 0x123 [0]=0x12 [1]=0x34 `2`=0x56 [3]=0x78

Enqueue DCL = 0x4

Enqueue data[0]=0x12 [1]=0x34 =0x56 [3]=0x78

In the MsBOT Master's command-line test operation, the contents of the received messages showed the establishment of the CAN-Bus communication. It is evidence of the solid and stable physical CAN-Bus channel between the USB-to-CAN adapter of the MsBOT Master and the MsBOT Component. It is a quick way to test the functionality of the physical layer, i.e., the USB-to-CAN adapter, as well as the data-link layer, i.e., the USB-to-CAN device driver.

Stage 2 – It introduced the new item, i.e., the program to establish the CAN transport using the CAN Socket APIs in the MsBOT Master. The test and verification operation performed at the MsBOT Component and the MsBOT Master displayed following messages in its respective serial terminal.

Notes: The debugging messages of the MsBOT Master showing in the Linux terminal

CAN2Agent::recv_message() entered

SocketCAN:: handler 3

SocketCAN::read getsockopt() succeeded len = 4

SocketCAN:: Read() - A Frame memory space is ready and about to read from CAN interface can0

Read() returns values = 16 and data length of 4

SocketCAN: Read() Classis frame.flags = 291

SocketCAN: Read() Non-Error frame.flags = 291

SocketCAN: Read() frame.can_dlc= 4, frame.can_id = 291

data[0x12] data[0x34] data[0x56] data[0x78]

SocketCAN: Read() ended

inFrame id=0x123, fr_num=1 length=4

msg_len 0

CAN2Agent::recv_message() exited

Notes: The MsBOT Component debugging messages showing in the Linux terminal

HAL_CAN_RxFifo0MsgPendingCallback() entered

ID = 0x123 RdDataLength 4

Data [0]=0x12 [1]=0x34 `2`=0x56 [3]=0x78

Enqueue id = 0x123 [0]=0x12 [1]=0x34 `2`=0x56 [3]=0x78

Enqueue DCL = 0x4

Enqueue data[0]=0x12 [1]=0x34 =0x56 [3]=0x78

Stage 3 – The MsBOT Component run the app.c and the MsBOT Master run the MicroXRCEAgent.cpp. The corresponding test and verification operations showed the respective debugging messages in the MsBOT Component and the MsBOT Master through the Terminal.

The debugging messages of the MsBOT Component showing in the Linux terminal.

Notes: Sending out CREATE_SESSION XRCE-DDS message

CAN2 Tx buf before framing begin >

[0]=0x80 [1]=0x0 `2`=0x0 [3]=0x0 [4]=0x0 [5]=0x1 [6]=0x10 [7]=0x0

[8]=0x58 [9]=0x52 [10]=0x43 [11]=0x45 [12]=0x1 [13]=0x0 [14]=0x1 [15]=0xf

[16]=0xba [17]=0x5e [18]=0xba [19]=0x11 [20]=0x81 [21]=0x0 `2`=0xec [23]=0x5

CAN2 Tx buf before framing ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 3 begins >

[0]=0x3 [1]=0x80 =0x0 [3]=0x0 [4]=0x0 [5]=0x0 [6]=0x1 [7]=0x10

CAN2 Tx frame 3 ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 2 begins >

[0]=0x2 [1]=0x0 `2`=0x58 [3]=0x52 [4]=0x43 [5]=0x45 [6]=0x1 [7]=0x0

CAN2 Tx frame 2 ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 1 begins >

[0]=0x1 [1]=0x1 `2`=0xf [3]=0xba [4]=0x5e [5]=0xba [6]=0x11 [7]=0x81

CAN2 Tx frame 1 ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 0 begins >

[0]=0x0 [1]=0x0 `2`=0xec [3]=0x5

CAN2 Tx frame 0 ended >

send_can_msg - TX_CAN_message() bytes_sent = 24

Notes: Received response on the CREATE_SESSION XRCE-DDS message from Micro-ROS Agent

listen_message() must_be_read is 1

remaining ub length 15 and submessage length 0

uxr_read_submessage_header() ready_to_read 1

remaining ub length 0 and submessage length 11

uxr_read_submessage_header() ready_to_read 0

[0]=0x81 [1]=0x0 `2`=0x0 [3]=0x0 [4]=0x4 [5]=0x1 [6]=0xb [7]=0x0

[8]=0x0 [9]=0x0 [10]=0x58 [11]=0x52 [12]=0x43 [13]=0x45 [14]=0x1 [15]=0x0

[16]=0x1 [17]=0xf [18]=0x0

listen_message()-session->info.last_requested_status 0

wait_session_status TRUE-session->info.last_requested_status = 0

wait_session_status - session->info.last_requested_status = 0, poll_ms = 25

wait_session_status - attempts = 10 loop 2

wait_session_status- bl_status 0 loop 2

SESSION STATUS received = 1

SESSION created 1

Notes: Use XML descriptions to establish the participant, topic, reliable streams, DataReader, DataWriter

appMain: uxr_create_session() succeeded

uxr_buffer_submessage_header() creating

SESSION STATUS received = 1

SESSION created 1

appMain: uxr_create_session() ***succeeded***

uxr_buffer_submessage_header() creating

remaining ub length 88 and submessage length 88

uxr_buffer_submessage_header() creating

remaining ub length 96 and submessage length 96

appMain: uxr_buffer_create_topic_xml() succeeded

uxr_buffer_submessage_header() creating

remaining ub length 16 and submessage length 15

appMain: uxr_buffer_create_publisher_xml() succeeded

uxr_buffer_submessage_header() creating

remaining ub length 16 and submessage length 15

appMain: uxr_buffer_create_subscriber_xml() succeeded

uxr_buffer_submessage_header() creating

remaining ub length 176 and submessage length 175

appMain: uxr_buffer_create_DataReader_xml() succeeded

appMain: uxr_object_id() succeeded

uxr_buffer_submessage_header() creating

remaining ub length 176 and submessage length 174

appMain: uxr_buffer_create_DataWriterr_xml() succeeded

Notes: The Stream Quality validation - appMain: uxr_run_session_until_all_status() messages

appMain: uxr_run_session_until_all_status() Begins

CAN2 Tx buf before framing begin >

send_can_msg - TX_CAN_message() bytes_sent = 416

CAN2 Tx buf before framing begin >

[0]=0x81 [1]=0x80 `2`=0x1 [3]=0x0 <-session_id=0x81, stream_id=0x80, sequenceNr(16)=0x0001 ->

[4]=0x1 [5]=0x5 [6]=0xae [7]=0x0 <-submessage_id=1=CREATE, flags=0x5(Replace bit-2=1, ReUse bit-1=0, bit-0=1), submessageLength=0xae=174 ->

[8]=0x0 [9]=0xf [10]=0x0 [11]=0x15 [12]=0x5 [13]=0x2 [14]=0xa5 [15]=0xa5

[16]=0xa0 [17]=0x0 [18]=0x0 [19]=0x0 [20]=0x3c [21]=0x64 `2`=0x64 [23]=0x73

[24]=0x3e [25]=0x3c [26]=0x64 [27]=0x61 [28]=0x74 [29]=0x61 [30]=0x5f [31]=0x77

[32]=0x72 [33]=0x69 [34]=0x74 [35]=0x65 [36]=0x72 [37]=0x3e [38]=0x3c [39]=0x74

[40]=0x6f [41]=0x70 [42]=0x69 [43]=0x63 [44]=0x3e [45]=0x3c [46]=0x6b [47]=0x69

[48]=0x6e [49]=0x64 [50]=0x3e [51]=0x4e [52]=0x4f [53]=0x5f [54]=0x4b [55]=0x45

[56]=0x59 [57]=0x3c [58]=0x2f [59]=0x6b [60]=0x69 [61]=0x6e [62]=0x64 [63]=0x3e

[64]=0x3c [65]=0x6e [66]=0x61 [67]=0x6d [68]=0x65 [69]=0x3e [70]=0x72 [71]=0x74

[72]=0x2f [73]=0x6f [74]=0x6c [75]=0x69 [76]=0x6d [77]=0x65 [78]=0x78 [79]=0x2f

[80]=0x70 [81]=0x75 [82]=0x62 [83]=0x6c [84]=0x69 [85]=0x73 `2`=0x68 [87]=0x65

[88]=0x72 [89]=0x3c [90]=0x2f [91]=0x6e [92]=0x61 [93]=0x6d [94]=0x65 [95]=0x3e

[96]=0x3c [97]=0x64 [98]=0x61 [99]=0x74 [100]=0x61 [101]=0x54 [102]=0x79 [103]=

[104]=0x65 [105]=0x3e [106]=0x67 [107]=0x65 [108]=0x6f [109]=0x6d [110]=0x65 [1

[112]=0x72 [113]=0x79 [114]=0x5f [115]=0x6d [116]=0x73 [117]=0x67 [118]=0x73 [1

[120]=0x3a [121]=0x6d [122]=0x73 [123]=0x67 [124]=0x3a [125]=0x3a [126]=0x64 [1

[128]=0x73 [129]=0x5f [130]=0x3a [131]=0x3a [132]=0x50 [133]=0x6f [134]=0x69 [1

[136]=0x74 [137]=0x33 [138]=0x32 [139]=0x5f [140]=0x3c [141]=0x2f [142]=0x64 [1

[144]=0x74 [145]=0x61 [146]=0x54 [147]=0x79 [148]=0x70 [149]=0x65 [150]=0x3e [1

[152]=0x2f [153]=0x74 [154]=0x6f [155]=0x70 [156]=0x69 [157]=0x63 [158]=0x3e [1

[160]=0x2f [161]=0x64 [162]=0x61 [163]=0x74 [164]=0x61 [165]=0x5f [166]=0x77 [1

[168]=0x69 [169]=0x74 [170]=0x65 [171]=0x72 [172]=0x3e [173]=0x3c [174]=0x2f [1

[176]=0x64 [177]=0x73 [178]=0x3e [179]=0x0 [180]=0x0 [181]=0x13 [182]=0xa5 [183

CAN2 Tx buf before framing ended >

ID = 0x120 RdDataLength 8

cir buf before count 314

Enqueue data[0]=0x1 [1]=0x81 `2`=0x0 [3]=0x0 [4]=0x0 [5]=0xb [6]=0x1 [7]=0x5 } <- HeartBeat=0xb, steam_id=0x0,

cir buf after count 330 } <- SequenceNr=0x0000

ID = 0x120 RdDataLength 7 } <-

submessageId=0xb, flag=0x1

cir buf before count 330 } <-

submsg_len=0x5

Enqueue data[0]=0x0 [1]=0x0 `2`=0x0 [3]=0x0 [4]=0x4 [5]=0x0 [6]=0x80 }

;

ID = 0x120 RdDataLength 7

cir buf before count 5010

Enqueue data[0]=0x0 [1]=0x0 `2`=0x2 [3]=0x0 [4]=0x5 [5]=0x0 [6]=0x80

cir buf after count 5025

send_can_msg - TX_CAN_message() bytes_sent = 13

[0]=0x81 [1]=0x80 `2`=0x2 [3]=0x0 [4]=0x5 [5]=0x1 [6]=0x6 [7]=0x0 <- TX STATUS=0x5 to uAgent

[8]=0x0 [9]=0xc [10]=0x0 [11]=0x13 [12]=0x0 [13]=0x0

listen_message()-session->info.last_requested_status 0

uxr_run_session_until_all_status(): [0]=1

uxr_run_session_until_all_status(): [1]=1

uxr_run_session_until_all_status():`2`=1

uxr_run_session_until_all_status(): [3]=1

uxr_run_session_until_all_status(): timeout=0

4.2.5.1 The MsBOT Component's Message Interpretation

The messages above are only excerpts of the entire debugging session. It intends to show a few important statuses of the newly implemented CAN transport of both the MsBOT Component and the MsBOT Master. Four parts of the messages are worth discussing.

1) The CREATE_CLIENT message

From the CREATE_CLIENT message above, the MsBOT Component sent out the message initiating the connection between Micro-ROS Client and Micro-ROS Agent.

Upon receiving the message, the Micro-ROS Agent creates a ProxyClient on the Micro-ROS Agent, resulting in the Micro-ROS Agent calling the Root::create_client operation. The message was sent from the user application app.c, and through the CAN transport layer and the CAN interface. It is a CREATE_CLIENT request message of 24-bytes long initiating a connection to the Micro-ROS Agent (i.e., MsBOT Master). It creates a new session on the Micro-ROS Agent. This message causes the Micro-ROS Agent to register to a new session, enabling any other XRCE communication with the Agent. The message carries the connection properties information for the Micro-ROS Agent to align with. The message comprises a 4-byte header, a 4-byte sub-message header and the sub-message body. The sessionid = 0x80, submessageid = 0x0, and clientKey = 0xBA5EBA11 are important properties for the Micro-ROS Agent to set up a new session.

[0]=0x80 [1]=0x0 `2`=0x0 [3]=0x0 <- Header - sessionId=0x80(128), streamID=0x0(referred as STREAMID_NONE), sequenceNr=0x0

[4]=0x0 [5]=0x1 [6]=0x10 [7]=0x0 < submessage - [4]=0 ->submessageID=0(CREATE_CLIENT), [5]=0x1->Endian *[5]=0x1 ->Little Endian, [6]=0x10, [7]=0x0* -> submessageLength 0x0010 = 16 bytes

[8]=0x58 [9]=0x52 [10]=0x43 [11]=0x45 <---Xrce_cookie

[12]=0x01 [13]=0x00 <--- Xrce_Version

[14]=0x1 [15]=0xf <--- Xrce_Vendor id => eProsima

[16]=0xba [17]=0x5e [18]=0xba [19]=0x11 <--- clientKey = 0xBA5EBA11

[20]=0x81 <--- session_id info_id

[21]=0x0 <--- Optional properties = false

`2`=0xfc [23]=0xf <--- Mtu

Submessage ID

value 0 - **CREATE_CLIENT - Client to Agent.**

To initiates the connection between Client and Agent. Creates a ProxyClient on the Agent, resulting in the Agent calling the Root::create_client operation.

value 1 - **CREATE - Client to Agent.**

Creates an XRCE Object, resulting in the Agent calling the ProxyClient::create operation.

Flags

Bit 0, **the 'Endianness' bit**,

value 0 - **big-endian and otherwise little-endian**.

CREATE_CLIENT_Payload object:

```
@extensibility(FINAL)
struct CLIENT_Representation {
        XrceCookie          xrce_cookie;
        XrceVersion         xrce_version;
        // XRCE_COOKIE
        XrceVendorId        xrce_vendor_id;
        ClientKey  client_key;
        SessionId   session_id;
        @optional           PropertySeq properties;
};
```

@extensibility(FINAL)

```
struct CREATE_CLIENT_Payload {
        CLIENT_Representation       client_representation;
};
```

2) The STATUS_AGENT message – the response from the Micro-ROS Agent

Upon receiving the submessageID=CREATE_CLIENT request, the Micro-ROS Agent reacts to create a corresponding ProxyClient by calling the Root::create_client operation. Eventually, it responds to the CREATE_CLIENT request with submessageID=STATUS_AGENT to inform the Micro-ROS Clientthat the session has been successfully created.

<Status-AGENT message> <- Response from uAgent

[0]=0x81 [1]=0x0 =0x0 [3]=0x0 [4]=0x4 [5]=0x1 [6]=0xb [7]=0x0

[8]=0x0 [9]=0x0 [10]=0x58 [11]=0x52 [12]=0x43 [13]=0x45 [14]=0x1 [15]=0x0

[16]=0x1 [17]=0xf [18]=0x0

Msg[4] = 0x4, value 4 of the fifth byte - Agent to Client, which was sent in response to CREATE_CLIENT. It contains information about the Agent; it carries the return value of the Root::create_client operation.

Micro-ROS Agent responded to the Micro-ROS Client CREATE_CLIENT message request.

[0]=0x81 [1]=0x0 `2`=0x0 [3]=0x0 <- Header - sessionId=0x81(129), streamID=0x0(referred as STREAMID_NONE), sequenceNr=0x0

[4]=0x4 [5]=0x1 [6]=0xb [7]=0x0 <- Submessage -[4]=4->submessageID=4 **STATUS_AGENT,**

[5]=0x1 ->Little Endian, [6]=0xb, [7]=0x0 -> submessageLength 0xb = 11 bytes

[8]=0x0 [9]=0x0 [10]=0x58 [11]=0x52 [12]=0x43 [13]=0x45 [14]=0x1 [15]=0x0

[16]=0x1 [17]=0xf [18]=0x0

3) Use XML descriptor to establish the participant, topic, reliable streams, DataReader, DataWriter.

XRCE-DDS messages are generally large and not readable for humans. As a resolution, the eProsima XRCE-DDS protocol adopts XML descriptors to construct the messages for establishing the participant, topic, communication streams, DataReader, DataWriter, etc. The XML descriptors help to structure in a more readable and well-structured components called schema to improve readability and ease modification. The debugging messages show the relevant XRCE-DDS messages were successfully created and sent through the CAN interface. The items were created using the XML descriptor and sent through the CAN interface. The Items being created using the XML descriptor are:

The topic *< - appMain: uxr_buffer_create_topic_xml() succeeded*

The publisher *<- appMain: uxr_buffer_create_publisher_xml() succeeded*

The subscriber *<- appMain: uxr_buffer_create_subscriber_xml() succeeded*

The DataReader *<- appMain: uxr_buffer_create_DataReader_xml() succeeded*

The DataWriter *<- appMain: uxr_buffer_create_DataWriterr_xmlData() succeeded*

4) The Reliable Communication Stream Quality validation

Despite the Client session and the participant, the topic, etc., having succeeded, the user program app.c would need to validate the quality of the set reliable communication streams. It is achieved by using the API, bool uxr_run_session_until_all_status(). This function validates the internal functionality of a session through the following actions:

1. Flushing all output streams sends the data through the transport.
2. For each reliable stream, it validates the associated reliable behaviour.
3. Listen to messages from the Micro-ROS Agent and call the associated callback accordingly.

These are performed until all requested statuses have been received or by a timeout duration. Each request has a corresponding status presented in the request list. In the end, the function returns true if all statuses have been received and all of them have the value UXR_STATUS_OK or UXR_STATUS_OK_MATCHED, false otherwise.

The debugging messages show that out of 5 requests, it received 4 UXR_STATUS_OK, but it failed on the last request. Overall, the uxr_run_session_until_all_status() returned non-UXR_STATUS_OK, i.e., 0.

appMain: uxr_run_session_until_all_status() Begins

[0]=0x81 [1]=0x80 `2`=0x2 [3]=0x0 [4]=0x5 [5]=0x1 [6]=0x6 [7]=0x0 <- TX STATUS=0x5 to uAgent

[8]=0x0 [9]=0xc [10]=0x0 [11]=0x13 [12]=0x0 [13]=0x0

listen_message()-session->info.last_requested_status 0 <- STATUS=0x0

uxr_run_session_until_all_status(): [0]=1

uxr_run_session_until_all_status(): [1]=1

uxr_run_session_until_all_status():`2`=1

uxr_run_session_until_all_status(): [3]=1

uxr_run_session_until_all_status(): timeout=0

4.2.5.2 The MsBOT Master's message interpretation

The messages above are only excerpts of the entire MsBOT Master debugging messages. The key to showing the messages is the operation status of the Socket CAN-implemented CAN transport. In short, the CAN transport was bind() successfully and created a CAN socket with the CAN interface can0. Through the SocketCAN: read() function, the Micro-ROS Agent read the CAN frames sent from the MsBOT Component.

Notes: The debugging messages of the MsBOT Master – uAgent

can_frame is 16 bytesSocketCAN bind() succeeded

SocketCAN getsockopt() succeeded

SocketCAN error none

SocketCAN for interface can0

succeeded

CAN2Agent::recv_message() entered

SocketCAN:: handler 3

SocketCAN::read getsockopt() succeeded len = 4

SocketCAN:: Read() - A Frame memory space is ready and about to read from CAN interface can0

Read() returns values = 16 and data length of 8

SocketCAN: Read() Classis frame.flags = 291

SocketCAN: Read() Non-Error frame.flags = 291

SocketCAN: Read() frame.can_dlc= 8, frame.can_id = 291

data[3] data[128] data[0] data[0] data[0] data[0] data[1] data[16]

SocketCAN: Read() ended

inFrame id=0x123, fr_num=3 length=8

msg_len 0

CAN2Agent::recv_message() exited

CAN2Agent::recv_message() entered

Reflection

In addition to many functionalities, the MsBOT Master also acts as a Micro XRCE-DDS Agent that operates as a server for the MsBOT Component, also called eProsima Micro XRCE-DDS Clients. Micro XRCE-DDS provides both, a Plug-and-Play Micro XRCE-DDS Agent and an API layer which allows users to implement your Micro XRCE-DDS Clients [28]. It is the first step of the 2-Step Piggyback Plug-and-Play process. The Plug-and-Play is achieved using the discovery mechanism [91], which allows Micro-ROS Clients to discover Micro-ROS Agents in the network. At the initial state where Micro-ROS Clients do not know the IP address of any Micro-ROS Agent, it sends a discovery call by multicast and the reachable Agents respond by sending the IP and Port information. With the information, the Micro-ROS Client proceeds to communicate with the matched Micro-ROS Agent.

The Micro-ROS Agent receives XRCE-DDS messages containing operations from the Micro-ROS Clients. In compliance with the request received, Agents react to the operation by creating the Proxy-Client entities and sending out corresponding responses and heartbeats to keep track of the Micro-Client. The DDS entities use the Micro-ROS Agent to interact with the DDS Global Data Space. In the MsBOT, the Micro-ROS Client and the Micro-ROS Agent communication currently support the CAN bus. While it is running, the Agent attends to requests from the Micro-Clients and responds with the corresponding response.

Unknown variables are invariantly prevalent in any project development. The countermeasure for unknowns is employing a pragmatic strategy. It is not a silver bullet, but a pragmatic methodology helping developers analyse issues, develop resolutions, test and validate, and reflect on the results and practice. The bottom-up stepwise development adopted in the MsBOT project is pragmatic for newcomers to STM32E407F, CAN controller, and CAN protocols. Besides, the CAN bus for the MsBOT project involves not the physical layer but the data link (device driver) layer and the transport layer. Using Open System Interconnection (OSI) to build CAN Bus from the bottom physical layer upwards to the transport layer is a natural deed.

The bottom-up stepwise development can work in conjunction with the Plan-Do-Check-Act (PDCA) approach. It enhances the pragmatism of the MsBOT project management in general and the test and verification specifically. In addition to the PDCA, reflection is another important project management feature. It reviews the

strengths and weaknesses of a resolution and shares the issues encountered, pitfalls to avoid, and potential resolutions. It is perfectly applicable for the stage 1, 2 and 3 approaches to the CAN bus development and the test and verification.

The stage 3 results showed that despite the session and the reliable communication streams, the XREC-DDS participant, topic, DataReader and DataWriter had been successfully created for a DDS entity to operate properly. It still needs to validate the reliable streams' quality. Including the validation to ensure proper communication between the Micro-ROS Agent and the Micro-Clients makes perfect sense. Without it, the quality of the communication streams would not be able to quantify for improvement purposes.

The results of being failed in the reliable communication reflect the existence of bandwidth issues in CAN transport. Further analysis shows that the bandwidth issues can be explained from:

1. The small payload of CAN standard classic frame. The MsBOT adopts CAN Standard Classic mode, which has a maximum 8 bytes payload. However, the DDS messages are generally large, ranging from tens to hundreds of bytes. For transmitting messages longer than the payload, the MsBOT introduces a secondary framing algorithm, i.e., breaking up the longer DDS messages to 7 bytes or shorter frames and sending them over the CAN Bus. Reserve a byte for denoting the frame numbers. The received frames are concatenated to form a complete DDS message on the receiving end. The following code shows a CLIENT_CREATE XRCE-DDS request message of 24 bytes long going through the secondary framing algorithms and ending with four frames for transmitting over the CAN classic mode. Transmitting four frames compared to the frame of the other transports, i.e., Serial of 30 bytes, UDP/TCP of 1500 bytes respectively, introduces at least four times longer transmission time.

    The MsBOT Component's CLIENT_CREATE XRCE-DDS request message:

    CAN2 Tx buf before framing begin >

    [0]=0x80 [1]=0x0 `2`=0x0 [3]=0x0 [4]=0x0 [5]=0x1 [6]=0x10 [7]=0x0
    [8]=0x58 [9]=0x52 [10]=0x43 [11]=0x45 [12]=0x1 [13]=0x0 [14]=0x1 [15]=0xf
    [16]=0xba [17]=0x5e [18]=0xba [19]=0x11 [20]=0x81 [21]=0x0 =0xec [23]=0x5

CAN2 Tx buf before framing ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 3 begins >

[0]=0x3 [1]=0x80 `2`=0x0 [3]=0x0 [4]=0x0 [5]=0x0 [6]=0x1 [7]=0x10
CAN2 Tx frame 3 ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 2 begins >

[0]=0x2 [1]=0x0 `2`=0x58 [3]=0x52 [4]=0x43 [5]=0x45 [6]=0x1 [7]=0x0
CAN2 Tx frame 2 ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 1 begins >

[0]=0x1 [1]=0x1 =0xf [3]=0xba [4]=0x5e [5]=0xba [6]=0x11 [7]=0x81
CAN2 Tx frame 1 ended >

HAL_CAN_AddTxMessage() Passed

CAN2 Tx sendbuf frames 0 begins >

[0]=0x0 [1]=0x0 `2`=0xec [3]=0x5
CAN2 Tx frame 0 ended >

Evidently, the Serial transport works. There are a few similarities and differences between the CAN classic mode transport and the Serial transport. Firstly, both employ secondary framing algorithms. However, the Serial framing size is 30 bytes long versus the CAN bus is only 8 bytes long. Secondly, the Serial is a one-to-one communication protocol, while the CAN bus is a many-to-many communication protocol. Thirdly, most XRCE-DDS messages are less than 30 bytes long, allowing one frame to carry the entire XRCE message.

Another good reference on the working transport frame size for XRCE-DDS protocol is RTI Connext Nano [58]. Like eProsima Micro XRCE-DDS implementation, RTi Connext Nano too implements OMG DDS-XRCE protocols for resource-constrained devices and adopts a client/server protocol. However, it specifies that its XRCE-DDS protocol works on any transport with at least 24 bytes. It is an optimum number because it covers the most frequently used XRCE-DDS messages.

For instance, XRCE-DDS heartbeat is 11 bytes, XRCE-DDS CREATE_SESSION is 24 bytes, and the STATUS_AGENT is 19 bytes long.

2. High overhead frame to payload ratio: A CAN standard classic 2.0A frame is of length 108 bits at the maximum frame size of which is made of 64 bits payload. Figure 22 below shows the CAN standard classic 2.0A frame structure. Calculating the overhead frame to payload ratio as 64 bits divided by 108 bits equals 0.59 rounded to 0.6. For a frame size of the 1-byte payload, the overhead frame ratio is 8 bits over 52, i.e., 108 – 56 (5 bytes) bits equals 0.15.

The calculations reveal that the overhead of CAN standard classic frames can be overwhelming for communications that require a high data rate. In the MsBOT situation, it suffers more from the CAN standard classic frame because of the long XRCE-DDS messages. Despite the secondary framing being a resolution for accommodating large XRCE-DDS messages, it causes the MsBOT to suffer further as it loses one more byte for representing the secondary frame index. In addition, not forget the secondary frame index byte has a limit of representing up to 256 secondary frames. Any XRCE-DDS messages longer than 256x7 = 1792 bytes will be dropped as errors.

Figure 22 Can Standard Classic 2.0A Frame Structure

3. The large XRCE-DDS messages – The DDS messages were originally designed for a TCPIP-based network. The TCPIP frame has a Maximum Transmission Unit (MTU) of about 1500 despite longer messages being possible. Similarly, the XRCE-DDS or specifically eProsima's Micro-XRCE-DDS messages are generally larger than the payload of the CAN classic mode frame. The message sizes range from tens to hundreds of bytes.

4. The secondary framing of the CAN classic 2.0A frame for handling large DDS messages further impact the strained conditions of the CAN-Bus. As such, the MsBOT had to implement the secondary framing algorithms to break up the large XRCE-DDS frames to 1+7 bytes long that fit perfectly into the 8-byte CAN frame.

The 1+7 de-framing structure represents one byte for representing frame indexes and left the 7 bytes for the de-framed XRCE-DDS messages. The resolution led despite the short message of 24 bytes will need four secondary CAN frames to transmit, not to mention the effects of the long XML-based messages which are a few hundred bytes long. From the debugging messages, it showed that the 418 bytes XML-based XRCE-DDS message took over a minute to complete the entire message. The effect is detrimental to the reliable QoS communication stream and the same to the best-effort QoS communication streams because the total transmission is too long to be practical.

5. The XML-based XRCE-DDS messages: Unlike hypertext markup language (HTML) organises texts in browsers, XML emphasises what contents to display in documents [86]. In order to help developers better in ease of use, readability, and maintainability, the Micro-ROS adopts XML to organise the XRCE-DDS messages. With the XML descriptors, eProsima's XRCE-DDS messages are increasingly readable, ease-of-use, and maintainable resulting in the developers now can construct XRCE-DDS messages as if code writing. For instance, the following code constructs a Micro-XRCE-DDS topic in the MsBOT Component. It first creates the participant id, then constructs the participant's XML descriptions and finally uses the API, namely uxr_buffer_create_topic_xml() to create an XML-based Micro-XRCE-DDS message.

```
// Create Micro-XRCE-DDS topic
    uxrObjectId topic_id = uxr_object_id(0x01, UXR_TOPIC_ID);
    const char* topic_xml =     "<dds>"
                                "<topic>"
                                   "<dataType>geometry_msgs::msg::dds_::Point32_</dataType>"
                                "</topic>"
                              "</dds>";
    uint16_t topic_req = uxr_buffer_create_topic_xml(&session, reliable_out,
                                                     topic_id, participant_id,
                                                     topic_xml, UXR_REPLACE);

    printf("\n\rappMain: uxr_buffer_create_topic_xml() succeeded\n\r");
```

Notes: The routine to create the XML-based Micro-XRCE-DDS topic messages.

Unfortunately, what works on TCP/IP transport may not work on CAN standard classic mode. The final data strings created from the above code were registered in the debugging messages below as 96 bytes XRCE-DDS sub-message, yet a complete message.

```
uxr_buffer_submessage_header() creating
 remaining ub length 88 and submessage length 88
 uxr_buffer_submessage_header() creating
 remaining ub length 96 and submessage length 96
appMain: uxr_buffer_create_topic_xml() succeeded
```

Notes: some the topic generation debugging messages captured when executing the app.c

The message size put much pressure on the CAN-Bus. Besides being large messages, they would need to go through the secondary de-framing before being able to get transmitted other than the CAN-Bus. The total effect of the large size XML-based XRCE-DDS messages is beyond the CAN-Bus capability and the Micro-ROS's expectation.

6. Inundated heartbeat frames in the Micro-ROS Client circular buffer – As soon as the ProxyClient is created the result of the CREATE_CLIENT XRCE-DDS, the Micro-ROS Agent is constantly sending out the heartbeat message. For the Micro-ROS Agent behaviour configuring details, please refer to CMakeCache.txt. The default heartbeat interval is set to 200ms. With the scenario of transmitting the message of 418 bytes, while the transmission was in progress, there are at least 300 heartbeat frames were being received in the Micro-ROS Client circular buffer. Despite the MsBOT Component running the FreeRTOS, it would take a long while before reaching the response message of the 418-byte long message in the circular buffer. the system would not attend to the responses in the circular buffer but momentarily service amongst the transmission and the reception. The time taken would be longer than solely transmission time. The scenario resulted in the Micro-ROS Client declaring time out for not receiving its response in time.

A resolution was implemented for remediating the issue, i.e., by hardcode some of the long XRCE-DDS messages and using a four-byte long symbol to represent each of the hard-coded messages. By only sending out the symbols, but not the real long messages, it would reduce

the transmission time and leave ample time for processing the incoming frames. The immediate effect of the resolution was the session creation took only two to three retries in comparison to eight to ten retries. However, the heartbeat frames still inundated the incoming buffer, so much so that it failed the reliable stream quality validation test, uxr_run_session_until_all_status().

Evidently, despite XRCE-DDS implementations and the effects of XML-based messages, the other limiting factors constitute the failed state of the MsBOT CAN transport; however, the size limitation of the CAN classic mode is the main reason for failing the reliable stream quality validation. A potential resolution is adopting the newer CAN technologies, namely Controller Area Network Flexible Data-Rate (CAN FD). The primary difference between the CAN Classic and CAN FD, besides many other features, is the message payload size has been increased to 64, compared to only 8 bytes in the classic CAN frame. However, the STM32E407F does not have the FD CAN controller or the USB-to-CAN adapter. The work could not be carried out without significant changes in the hardware of both the MsBOT Component and the MsBOT Master.

4.2.6 The Remedial Solution

In order to proceed with MsBOT's user application development, it was suggested to proceed with the development entirely on a Linux PC. Besides the STM32 controller, the eProsima Micro-XRCE-DDS also supports many Linux distributions. There are many advantages of using PC-based Linux for developing the Micro-XRCE-DDS user application, namely:

1. High performance – in contrast to the STM32 controller, PC-based Linux is much more powerful in terms of its processing power, multi-threading, and multi-processing. Multi-process ability eases the need to use multiple hardware to accomplish multiple processes. In addition, to be realistic, multiple processes consume a lot of processing power. Without a high-performance system, multiple sluggish processes can be challenging to accommodate.

2. On-system development – Unlike many Microcontroller-based developments, off-system development uses cross-compilers and uploads the object code onto the

Microcontroller for test runs. On-system development does all deeds on the system from the development till the final satisfactory version.

3. The integrated development environment and supporting tools – The native integrated development environment does provide some convenience avoiding unnecessary incompatibility issues. However, it is not uncommon to use unconventional tools. In a matter of fact, many development platforms are not complete, instead, relies on third-party tools. A PC-based is presumably able to support those unconventional needs. For instance, Eclipse programmer editor and MELD visual diff application.

4. The versatile USB interfaces and its various device drivers – The USB interface is one the most versatile interface ever developed on PCs. The versatility of the USB is its adaptability to other interfaces. For instance, the USB-to-CAN adapter has conveniently provided two CAN interface to the PC without much effort, simply by plugging in the adapter. The same applies to the USB-to-Serial UART adapter allowing serial debugging messages of a Microcontroller to display on a PC at ease.

Moving on to PC programs to realise the MsBOT Plug-and-Play features, the advantages, and disadvantages of adopting a PC for user application development were discussed despite the failure of the CAN bus. Evidently, it was rational to proceed with developing the MICRO-XRCE-DDS client application using and running on a PC.

4.3 The MsBOT MICRO-XRCE-DDS client User Applications

ROS 2, by its nature, is not suitable for resource-constraint devices. However, deriving from ROS2 implementation, eProsima Micro-XRCE-DDS implementation [80] enables extremely resource-constraints devices to operate as DDS entities communicating to the DDS world. The eProsima Micro-XRCE-DDS is given another name of Micro-ROS and has been officially proclaimed as the official extension of ROS2 middleware for Microcontrollers (MCUs).

The MsBOT user application is based on eProsima's DDS-RTPS protocols and Micro-DDS-XRCE implementation, which have shown numerous successes in distributed robotic systems. Unlike other DDS implementations, the eProsima's Micro-DDS-XRCE is not only

has a small footprint but also versatile making it suitable for extremely resource-constraint devices. Due to its promising versatility and robustness, it has been ported to many different Microcontrollers, including STM32, ESP32, etc. In addition, Micro-DDS-XRCE is open-source and has a large and active community resulting in Micro-DDS-XRCE gaining much popularity among extremely resource-constraints devices developers who are interested in developing XRCE-DDS-enabled distributed robotic systems.

The MsBOT user applications are leveraging the Micro-ROS technologies to develop the Plug-and-Play framework of the robotic educational system. Unlike conventional robotic systems, the robotic system, namely MsBOT, is made of remotely connected components through the CAN bus. In addition, the robotic components are self-contained intelligent units made of Microcontrollers and equipped with the latest networking technologies and various in-system and off-system connection interfaces. The Plug-and-Play framework is realised through the user applications using the Micro-XRCE-DDS APIs.

The following section lists the major software modules that construct the Plug-and-Play framework and elaborate on the test and verification processes applied to the Play-and-Play modules using the Plan-Do-Check-Act (PDCA) methodology.

4.3.1 The operation of the MsBOT publishers and subscribers - the first-step authentication process

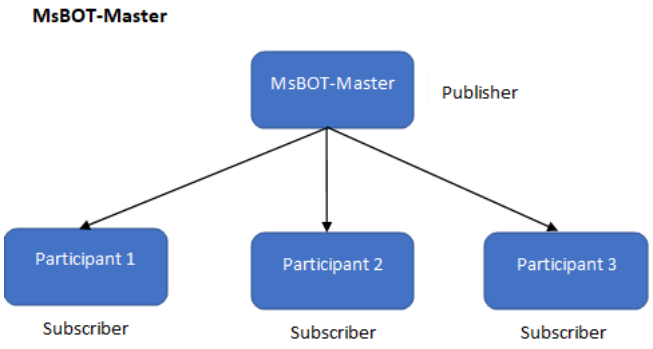

Figure 23 MsBOT Master as a publisher publishes MsBOT information as the first-step authentication process

Analogous to the distributing nature of newsletters, in eProsima Micro-ROS, only the subscribers get the publishing news. Similarly, to the nodes in ROS2, the participants are the XRCE-DDS entities which act as DDS entities from the DDS world perspective. Not all robotic components can access the MsBOT, but only those components that have been pre-

registered and passed the authentication get the authorisation to proceed to work as components of the MsBOT.

In Micro-ROS, to participate in the Micro-ROS operation, any piece of intelligence hardware will need to denounce as a participant. The participant will then proceed to set up its roles, whether as a publisher or subscriber or both and as a requester or replier. In addition, the participant can set up its topics besides the infrastructure settings such as the DataWriter, the DataReader, communication stream, etc.

Being a MsBOT Master, one of its roles is to publish its robotic information allowing the subscribers to verify the correct system to be associated with. Naturally, the subscribers are the products of hardware that have been denounced as Micro-ROS participants and have been set up as subscribers for the MsBOT Master's information topic. Figure 23 above shows the MsBOT Master publishes the MsBOT information for its MsBOT Components to subscribe for validating their validity.

Aims – The MsBOT Component subscribers who subscribe to the MsBOT Master can verify the correct MsBOT Master system which they associate with.

Specifications – Both the main_msbotmaster_publisch.c and the main_msbotmaster_subscribe.c run properly, and in the on_topic callback routine of main_msbotmaster_subscribe.c, the msbot_master topic's object_id matches the DataReader_id.

Algorithms – For the Linux-based Micro-ROS, both the Micro-ROS Client and the Micro-ROS Agent reside in the same PC system. Run the main_msbotmaster_publisch.c in the MsBOT Master and the main_msbotmaster_subscribe.c in a separate terminal.

Results – The received topic's object_id matched the DataReader's object_id.

Reflection – Despite it being a simplistic way to regulate unsolicited robotic components, it is, as a matter of fact, the first line of defence. Hence, it is regarded as the first-step authentication process. It is the first line of security for stopping unsolicited robotic components from using the MsBOT resources. In the real world, it is done through the default factory pre-set features. As such, any robotic components

that have not been programmed to subscribe to the topic will not gain access to the MsBOT. In addition, it can be an effective solution to mitigate the risks of being misused.

From another perspective, the published robotic information defines what the robotic system is. Hence, a new robotic system can be created by simply creating a new publisher in the MsBOT Master. To achieve that yet without compromising the security measures, an administrator who has the security key could unlock and reprogram the new information of the robotic system.

Despite it being simplistic, this feature has formed the basic security framework and can be a reference criterion of the framework for the robotic educational system. Further research is needed to standardise the robotic information and the authentication process.

4.3.2 The operation of the MsBOT requester and replier - the second-step authentication process

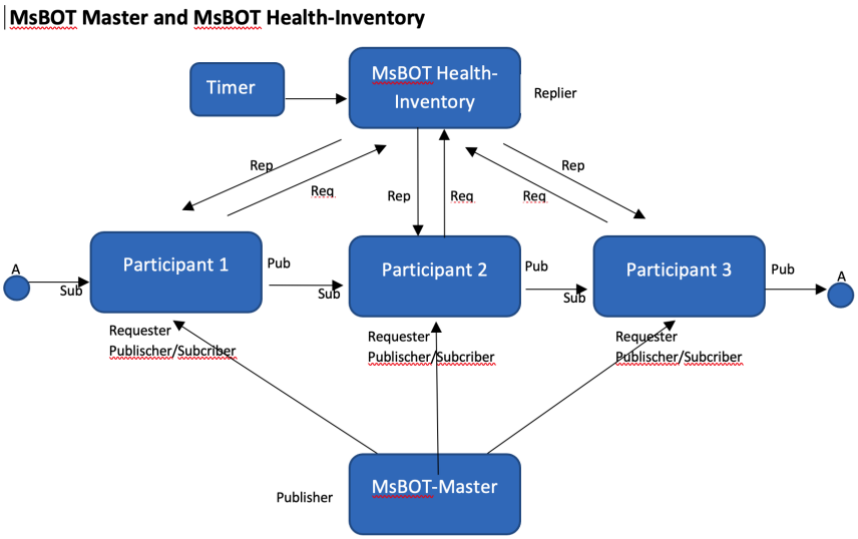

Figure 24 The MsBOT Health Inventory is a replier responsible for the second-step authentication process

Micro-ROS or specifically eProsima DDS-RTPS protocol, is a not client-server communication protocol. As the name RTPS implies, it means real-time Publisher Subscriber system. However, for achieving the client-server communication, the MsBOT adopts Micro-ROS Requester and Replier, mimicking client-requesting and server-responding operations.

After authenticating through the first-step authentication process, the MsBOT Component is granted access to the robotic system. It proceeds to the second step of authentication, i.e.,

the part information verification. Only the robotic components that match the registered part information are granted full access to the MsBOT.

The MsBOT component requests a DDS-RTPS message to the MsBOT Health-Inventory program residing in the MsBOT Master for parts verification. Once the parts are matched, the components are registered as active in the health inventory system. The MsBOT Master sends a reply message back to the sender. After completing the second-step authentication, the component is granted full access to the MsBOT Master. Once it has been registered in the health inventory system, to show that itself is alive, the MsBOT Component sends out a heartbeat every 2000ms the health inventory system. In the event the health inventory system misses out on three consecutive heartbeats, the corresponding MsBOT component will be marked as inactive. A new component of similar part information or the deregistered component can initiate a connection to the MsBOT Master by going through the two-step authentication processes.

The nature of the Micro-ROS Requester and Replier meet the communication requirements of the MsBOT components authentication processes despite Micro-ROS being an RTPS system. Figure 24 above shows that the MsBOT comprises one MsBOT Master and three MsBOT Components and their respective Participants.

Aims – To validate the full operation of the second-step authentication process.

Specifications –
The robotic component's part information gets verified and then registered by the health inventory system.
The component sends out a heartbeat every 2000ms. If the health inventory system misses out on three consecutive heartbeats, the component gets deregistered.

Algorithms: For the Linux-based Micro-ROS, both the Micro-ROS Client and the Micro-ROS Agent reside in the same PC system. Therefore, run each program in a separate terminal.

1. Run the health-inventory program in the MsBOT Master.
2. Run the uClient_AA_main.c program in the MsBOT Component.

Results: The part information of the uClient_AA component was validated and registered accordingly in the health inventory system. The health inventory deregistered the component when the uClient_AA halted for longer than 6000ms.

Reflection

The robotic part information validation serves as the second defence for the MsBOT to mitigate risks from being abused by any unauthorised devices. The health-inventory system is also tasked to administrate the registration and deregistration when validated components get connected and disconnected respectively. This is the second set of robotic frameworks for the MsBOT educational robots.

At the initial state when the Health Inventory system starts up, the database holds the predefined robotic part information of all the members of the MsBOT system. The information of each part is used for validating against the connected component's part information. In other words, the predefined robotic part information defines what the robotic components are and what the robotic system is.

To create a new robotic system for the DDS world, which can be achieved by creating a database, and filling up the components' information. Similarly, to remove a robotic system from the DDS network, simply remove the database. An administrator can modify, add or remove robotic components from the database to meet the universities' needs.

Nevertheless, the database at the current stage is hardcoded in the Heath Inventory system program. It is highly suggested that a more robust and complete database is created using the open-source SQL server not only does it inherit the authentication mechanism but also improved security. The part matching is through the simplistic validation algorithms. However, by having a robust auto-matching able to pull the appropriate part information that characterises the configuration of the robotic.

A great example is the matchmaking of the device driver when a new PC part is added to a PC. Figure 25 below shows the new processing routines for the future research activities, the Hardware Information Module Libraries, the SQL Database Manager and the Auto-matching Manager. When a DDS-RTPS request message carries the newly robotic part information, the Auto-matching Manager matches the part information stored in the Hardware Information Module libraries. Once a matched part is found, it passes the control back to the auto-matching manager for

registering the part into the SQL database. However, unlike accessing the database itself, it passes to the SQL Database Manager.

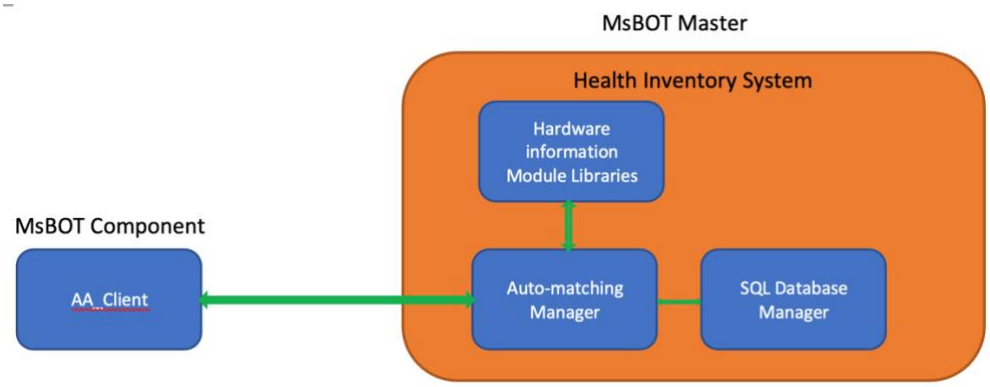

Figure 25 The three suggested modules for the future research activities

4.3.3 The multiple roles of the MsBOT component

An XRCE-DDS entity is not limited to one single role from the DDS-RTPS's perspective, but many roles were defined by the robotic system. The MsBOT system consists of four participants, uClient_AA, uClient_BB, uClient_CC and MsBOT Master. Each of the participants and the MsBOT Master is assigned multiple roles. Figure 24 above shows the participants are assigned multiple roles as publishers, subscribers, requesters, replier, etc. The details of the role of each participant are as below:

The MsBOT Master's roles

- A publisher to publish the MsBOT robot information.
- A replier to respond to the participants' parts authentication and to react to the heartbeats of the participants.

The uClient_AA's roles

- A subscriber for the MsBOT Master's robot information.
- A requester to send out the part information for part verification and registration and send out the heartbeats.
- A publisher to send out control instructions for uClient_BB.
- A subscriber to uClient_CC's control instructions.

The uClient_BB's roles

- A subscriber for the MsBOT Master's robot information.
- A requester to send out the part information for part verification and registration and send out the heartbeats.
- A publisher to send out control instructions for uClient_CC.
- A subscriber to uClient_AA's control instructions.

The uClient_CC's roles

- A subscriber for the MsBOT Master's robot information.
- A requester to send out the part information for part verification and registration and send out the heartbeats.
- A publisher to send out control instructions for uClient_AA.
- A subscriber to uClient_BB's control instructions.

Aims - To verify the roles of each participant as listed in the above paragraph.

Specifications -

- The MsBOT Master is tasked to perform:
    - MsBOT Master publishes the MsBOT robot information successfully.
    - MsBOT Master receives the participants' requests and successfully sends out the corresponding reply to the participants.
    - MsBOT Master receives the participants' heartbeat requests and acts on them accordingly
- The uClient_AA is tasked to perform:
    - uClient_AA subscribes successfully to the MsBOT Master.
    - uClient_AA sends the request successfully to the MsBOT Master.

- uClient_AA publishes the message to uClient_BB successfully.
- uClient_AA subscribes to the message to uClient_CC successfully.

- The uClient_BB is tasked to perform:
  - uClient_BB subscribes successfully to the MsBOT Master.
  - uClient_BB sends the request successfully to the MsBOT Master.
  - uClient_BB publish the message to uClient_CC successfully.
  - uClient_BB subscribes to the message to uClient_AA successfully.
  - The uClient_AA is tasked to perform:
  - uClient_CC subscribes successfully to the MsBOT Master.
  - uClient_CC sends the request successfully to the MsBOT Master.
  - uClient_CC publishes the message to uClient_AA successfully.
  - uClient_CC subscribes to the message to uClient_BB successfully.

Algorithms -

1. To perform the operation of the MsBOT publishers and subscribers - the first-step authentication process.
2. To perform the operation of the MsBOT requester and replier - the second-step authentication process.
3. To run programs, uClient_AA_main.c, uClient_BB_main.c, and uClient_CC_main.c in their respective MsBOT Component.

Results – For the results of the first-step authentication process, please refer to the test and verification, namely the MsBOT publishers and subscribers above. Similarly,

for the results of the second-step authentication process, please refer to the test and verification, namely the MsBOT requester and replier above.

The third test and verification results show that uClient_AA published its topic successfully, and the uClient_BB subscribed to uClient_AA's topic successfully. Similarly, uClient_BB published its topic successfully, and the uClient_CC subscribed to uClient_BB's topic successfully. The same applied to uClient_CC published its topic successfully, and the uClient_AA subscribed to uClient_AA's topic successfully.

Reflection - The MsBOT project, which leverages the Micro-ROS technology, would be constrained functionally if Micro-ROS had only limited roles a participant could perform. Fortunately, Micro-ROS allows participants to do multiple publishers, subscribers, requesters and repliers, and other roles. The DDS-RTPS protocols originally did not support the client-server operation. However, the nature of the client-server operation can be more efficient in some situations, such as the second-step authentication and the heartbeats are achieved using the DDS-RTPS requester and repliers.

Figure 26 below shows the communication among the MsBOT Components. The uClient_AA denoted as participant 1 publishes control instructions to the uClient_BB denoted as participant 2 while subscribing to the control message from uClient_CC denoted as participant 3. The uClient_BB publishes the control message to uClient_CC while subscribing to the control message from uClient_AA. Similarly, uClient_CC publishes the control messages to uClient_AA while subscribing to uClient_BB's control message.

Figure 26 The three Participants of the MsBOT Components communicate with one another.

The participants' multiple roles of the MsBOT intend to show how the participants of the Micro-ROS technology can communicate with one another despite being remotely apart. Collectively the participants can achieve and work as if it is a monolithic piece of a robotic system.

4.4 The reflection on the CAN-Bus Implementation

The CAN standard classic 2.0A frame has only a maximum payload of 8 bytes. The inherently long XRCE-DDS messages will need a secondary framing protocol to work with the CAN protocol, which causes the MsBOT to suffer one byte for representing the secondary frame index.

However, CAN-Bus has gained wide acceptance in the automobile industry and other harsh electric and electromagnetic environments automation and robotic systems in the commercial environment. This ability is ideal for safety-critical applications. These applications are low-level machine-to-machine communication which requires only a low data rate.

On the contrary, the high data rate TCP/UDP-based inter-network and Internet expect larger bandwidth transport protocols to transport Ethernet frames which can be 1500-byte MTU or longer. In the MsBOT, the XRCE-DDS protocols originated from TCP/UDP transport can hit the bottleneck when applying CAN-Bus, especially employing the CAN standard classic frames. To rub salt in wounds, the extremely high overhead of CAN standard classic frames is another killer despite its bitrate being 1Mbps at the maximum.

To move forward, the potential remedy for the limitations and yet taking advantage of the CAN bus characteristic is adopting CANFD. The improved protocol overcomes the CAN limit. CANFD transmits data at varying rates from 5Mbit/s to 8Mb/s, depending on the payload, and the payload can go as high as 64-byte long. Besides, the CAN FD protocol also supports the Classical CAN frame, which is compatible with the old CAN 2.0A standard. Both the CAN classic and the CANFD comply with internationally standardized ISO 11898-1:2015. Figure 27 below shows the compatible CAN classic frame structure of the CANFD.

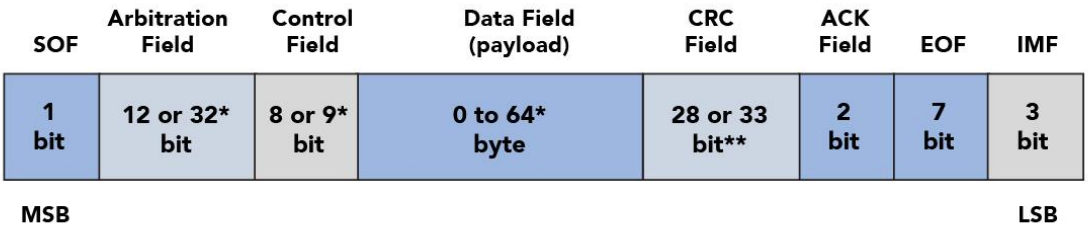

Figure 27 The structure of CAN FD data frames which are compatible with the Classical CAN data frames, but some extensions have been introduced

In order to introduce the CANFD in the MsBOT, the existing STM32E407F Eval board will need to adopt an external CANFD Controller because the board's CAN controller does not support it. The most convenient way is by connecting an external CANFD controller module through one of the existing SPI interfaces. Upon receiving a CANFD frame, the SPI interrupt will trigger the STM32 MCU to service the incoming CANFD frame. Similarly, it applies to the out-going CANFD frame. The CANFD -Bus will need a corresponding CANFD controller at the MsBOT Master, which can employ a USB-to-CANFD adapter. Figure 28 below shows the set-up of the new CANFD controller module connecting to the existing MsBOT Eval board and the new USB-to-CANFD adapter connecting to the existing MsBOT Master.

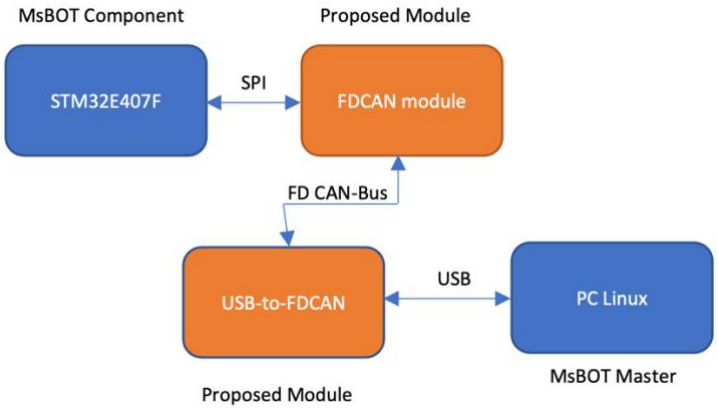

Figure 28 Introducing CANFD to MsBOT

5 Discussion and Conclusion

5.1 MsBOT Inspiration and Aspiration

Educational robots are inherently vertically integrated and stand-alone. User choices are locked into the manufacturers' resolutions, resulting in reduced competition among robotic manufacturers.

The fast-paced learning environment and increasingly complex robotic curriculum demand immediate resolutions that improve the quality of educational robotic systems, which are Plug-and-Play, interoperable across the board, easing component replacement and new component construction and testing, empowering users' choice, and leaving teachers and students more time to focus on teaching and studying.

MsBOT project was inspired to rectify the weaknesses of the inherent vertically integrated educational robotic systems and the meticulous maintenance and configuration tasks. Besides, it aspires to improve the robotics quality by introducing a Plug-and-Play robotic framework for the education industry, if not robotic industries. Instead of reinventing the wheels, the MsBOT framework uses the open-source yet well-established robotics technology by eProsima's Micro XRCE-DDS, the middleware of OMG's XRCE-DDS, etc. The MsBOT Plug-and-Play leverages the in-built Plug-and-Play feature of the Micro XRCE-DDS Agent and the APIs for Micro XRCE-DDS Clients [14] connecting and discovering the Micro XRCE-DDS Agent, which is called MsBOT 2-Step Piggyback Plug-and-Play implementation. But not least, to produce a real-life working Plug-and-Play education robot. With the Plug-and-Play feature in educational robotic systems, teachers and students are no longer bothered by the nitty-gritty of configurational details but rather focus on teaching and learning.

As robotic systems resemble automotive systems, hence it is logical to apply CAN-Bus in the MsBOT, which adopts the CAN standard classic 2.0A and is characterised as an 8-byte payload supporting bitrate up to 1 Mbps CAN-Bus protocol. The CAN-Bus transport protocol connects Micro-ROS clients and Micro-ROS agents. Being the transport of the MsBOT implicates it transports CAN messages amongst the MsBOT Components, also called Micro-ROS Clients and the MsBOT Master, also called Micro-ROS Agent. The CAN-Bus is achieved using the onboard CAN controllers that implement the ISR routines and the circular buffers. In contrast, the Micro-ROS Agent runs on Linux systems, leveraging the in-system CAN Socket kernel device driver to establish the CAN-Bus.

5.2 The Findings of the MsBOT Study

5.2.1 The MsBOT CAN-Bus Implementation

The reliable communication stream qualification in the stage 3 Test and Results showed that despite the success in creating the session and the DDS entity, a piece of evidence that the CAN-Bus has been established, yet the reliable communication streams of the CAN-Bus failed to meet the reliable state conditions. The analysis identifies that it is suffering from bandwidth issues, which are caused by:

1. The small payload of CAN standard classic 2.0A frame of 8 bytes is too short for the large XRCE-DDS messages. The first working reference, RTI Connext Nano [58], another XRCE-DDS implementation, specifies that its XRCE-DDS protocol works on any transport with at least 24 bytes. The second working reference, eProsima Micro XRCE-DDS implementation, employs 30 bytes for the serial UART transport.

2. The secondary framing of the CAN classic 2.0A frame for handling large DDS messages further impacts the CAN-Bus bandwidth's strained conditions. For instance, the heartbeat message of 13 bytes long will need to be deframed into two CAN frames: firstly, 1(index)+7(message) bytes long and secondly, 1(index)+6(message bytes long.

3. The XRCE-DDS messages are inherently large and originated from TCP/IP protocols. The smallest message is still larger than the payload of CAN classic mode.

4. The XML-based XRCE-DDS messages cause the messages to be even larger, putting further stress on the already strained CAN-Bus's bandwidth.

5. High overhead frame to payload ratio: A CAN standard classic 2.0A frame is of length 108 bits at the maximum frame size, but 64 bits payload. For instance, for the full-size frame of 64-bit, the overhead frame to payload ratio is 0.6, while for a frame size of the 1-byte payload, the overhead frame ratio is 0.15. The figures show that CAN standard classic 2.0A mode is unsuitable for large messaging communications.

6. Inundated heartbeat frames in the Micro-ROS Client circular buffer, causing the reliable communication stream validation routines to miss the timing. As soon as the ProxyClient is created, the micro-Agent constantly sends out the heartbeat messages. The longer it takes for the Micro-ROS Client to send out the message, the more heartbeat messages the Micro-ROS

Client receives. Hence, the reliable stream qualification routine interprets it misses out on the timing condition.

5.2.2 The MsBOT 2-Step Piggyback Plug-and-Play Framework

The results of the test on the MsBOT Health Inventory system and the user robotic applications of MsBOT Components modules show that:

The Plug-and-Play framework work, called MsBOT Piggyback Play-and-Play, uses the Play-and-Play feature of Micro-ROS middleware. It comprises the two-step authentication process for preventing unsolicited access to the MsBOT systems, the robot composition database defining what the system is, and the health inventory system keeping track of the MsBOT health status. The first authentication process defines what the robot is. Hence, only those robotic components made for the robot will be granted access to the MsBOT Master. Please see the Test and Results in section 4.3.1 for details. It then follows the second authentication process by validating the components' hardware information. Please see the test outcome of the Test and Results in section 4.3.2.

Only components that match the predefined hardware information stored in the health inventory database will be granted access to the MsBOT. After gaining access to the MsBOT, the components could perform the usual tasks. Please see the Test and Results in section 4.3.2 for details.

The health inventory system manages the registration and deregistration of the MsBOT Components accordingly when connecting and disconnecting. It only registers those matches the predefined components in the database. In the event that a component misses out on three consecutive heartbeats, the heath inventory deregisters the component from the list. Please see the Test and Results outcome in section 4.3.2 for details.

MsBOT is truly a distributed robotic system where the MsBOT Components can work collectively as a true robot despite being connected remotely. The multiple roles of the components are arbitrary tasks set to represent what the robotic components are capable of. They share the conditions with others, who depend on it for further actions. For example, the uClient_BB depends on uClient_AA, and the uClient_CC depends on uClient_BB, while the uClient_AA depends on uClient_CC. Please see the Test and Results in section 4.3.3 for details.

Despite the MsBOT Component being an extremely resource-constrained device that can still access the DDS world as if it is a full-fledged computer system. Please see the outcome of the Test and Results in section 4.3.3.

MsBOT is hardware agnostic. The MsBOT Master does not validate the hardware of MsBOT Components for making connections so long they support the same transport protocol. Instead, it recognises the XRCE-DDS messages. However, Micro-ROS supports many manufacturers' microcontrollers, and the list is growing. Please see the Micro-ROS [30, 32] for full descriptions.

5.3 The Limitation of the MsBOT

There is a misconception about the Plug-and-Play feature of robotic systems. Hence, enormous time was wasted, which could have been more productive if it had been understood before. Historically, Plug-and-Play is a prevalent feature in PCs where peripherals are plugged in; without further configuration, the peripheral works by itself. The work involved in accomplishing the Plug-and-Play feature is enormous, far beyond what any individual could.

In contrast, the strategy to develop the MsBOT Plug-and-Play feature is unconventional. Instead, it piggybacks on an existing Play-and-Play feature of an XRCE-DDS middleware, namely Micro-ROS. The approach opens a new horizontal of Plug-and-Play feature development because it no longer restricts the interface specifications or the operating systems. Instead, it can focus on the design and implementation of authentication and authorisation of solicited components and the mechanism of matching the configuration from the database and the effective database system.

Despite MsBOT being successful in adopting the piggyback Play-and-Play resolution, it still suffers from many limitations. The decision to use CAN-bus as the MsBOT transport is an oversight. At times when the decision was made, the knowledge of Micro-ROS, especially the bandwidth requirements of the Micro-ROS transport, was very minimal. It was beyond my knowledge to judge the implication of the CAN standard classic 2.0A to the Micro-ROS operation.

The decision to use the STM32E407F Eval board as the MsBOT development platform is erratic. Micro-ROS recommends the STM32E407F Eval board, and all the software modules

get support from the community. However, the in-system CAN controller of STM32E407F does not support CANFD.

The payload of CAN standard classic 2.0A mode is very limited. Originating from TCP/IP transport, XRCE-DDS messages are generally very large compared to the 8-byte payload of the CAN standard classic mode. Despite successfully implementing the CAN transport for both the MsBOT Master and the MsBOT Component, the MsBOT failed the reliable stream qualification test and ended with the system halt.

Despite the entire MsBOT user applications of the MsBOT Master and the MsBOT Components working properly, it may not work on the STM32E407F board. It is because the entire MsBOT user applications run on an Ubuntu PC which is very stable and powerful. In contrast, the STM32E407F is less powerful and runs on FreeRTOS. Hence, the results on the microcontroller board may be different.

The overall performance of the MsBOT 2-Step Piggyback Plug-and-Play design is satisfactory. However, the current design is meant for proofing the concepts and setting up a Plug-and-Play framework. For instance, the health inventory database is made using a multi-dimensional array. The matching algorithms are sequential search and match, which is unsuitable for the real-life system. In addition, the predefined robotic component and its hardware module information are hardcoded in the health inventory database because it is beyond the scope of the study framework.

The MsBOT Component user applications, namely uClient_AA, uClient_BB, and uClient_CC are meant for presenting the multiple roles of a Participant, which in real life, the Participant of a robotic component can use for achieving their designated system design. The same applies to the IO Serial boards, which are meant to show the capability of Micro-ROS Participants.

5.4. Recommendations for Implementation and Future Research

To improve the implications of these limitations, future studies could address two major areas, as shown in the sub-sections below.

5.4.1 Upgrade to CANFD

CAN standard classic 2.0A mode is the culprit for the failing state of the reliable communication stream qualification. The 8-byte payload is not viable for the Micro XRCE-DDS messaging systems. The findings parallel the Micro-ROS 30-byte payload serial transport and the RTi Connext Nano advice of requiring at least 24 bytes payload of any transports. To move forward, taking advantage of the knowledge learned from CAN bus transport implementation and the virtue of CAN-Bus characteristics, adopting CANFD is a sound decision. CANFD transmits data at varying rates from 5Mbit/s to 8Mb/s, depending on the payload, and the payload can go as high as 64-byte long. Besides, the CANFD is compatible with the old CAN 2.0A standard protocol. The CANFD and its predecessor CAN standard classic comply with internationally standardized ISO 11898-1:2015. Figure 27 in Chapter 4 shows the compatible CAN classic frame structure of the CANFD.

Since the existing STM32E407F Eval board does not CANFD controller that provides CANFD functionality, it will need to employ an external CANFD Controller. The most convenient way is by connecting an external CANFD controller module through one of the existing SPI interfaces. In other words, CANFD frames come through the SPI bus. The corresponding software design is instead of using the in-system CAN controller, it uses one of the in-system SPI. Upon receiving a CANFD frame by the CANFD controller, it triggers the SPI interrupt to the STM32F407 microcontroller to service the incoming CANFD frame. However, outgoing CANFD frames do not need an interrupt mechanism; instead, simply need to send them out through the SPI bus directly.

The MsBOT Master will need a corresponding CANFD controller for facilitating the CANFD-Bus. The easiest and most convenient option is employing a USB-to-CANFD adapter. The corresponding software changes are straightforward and only involve in reconfiguring the arguments of the SocketCAN APIs. Figure 28 in Chapter 4 shows the set-up of the new CANFD controller module connecting to the existing MsBOT Eval board and the new USB-to-CANFD adapter connecting to the existing MsBOT Master.

5.4.2 Improve the Health Inventory System

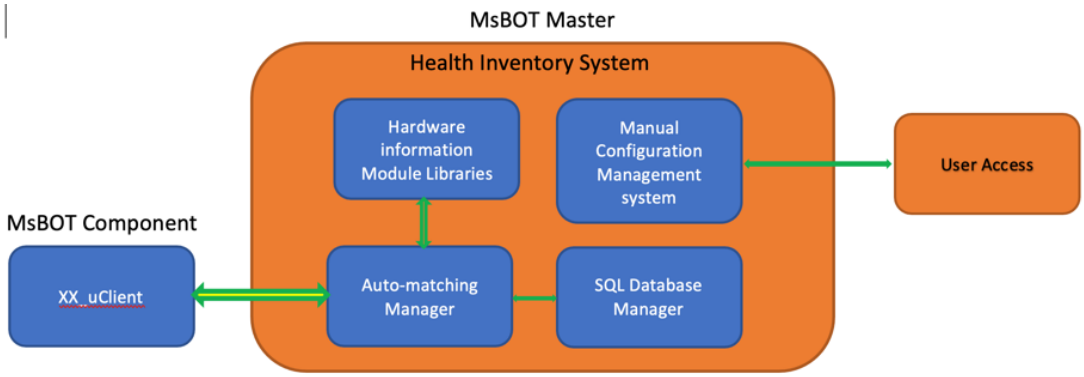

Figure 29 The new proposed modules for improving the MsBOT in the future

The health-inventory system is the second defence for the MsBOT to mitigate risks from being abused by any unauthorised devices. It is responsible for regulating the well-being of the on-board components. It is also treated as the second set of robotic frameworks for the MsBOT educational robots.

The Health Inventory system holds the predefined robotic part information of all the members of the MsBOT system. Nevertheless, the database at the current stage is hardcoded in the Heath Inventory system program for the sake of proofing the concepts. However, in the real world, it should be done with a proper database system, preferably open-source solutions. In other words, the Health Inventory should not have any software entity that stores the part information. It should employ a database management system to administrate the database including but not limited to data storing and retrieving management.

It needs a more robust and complete database management system that has an efficient search and match algorithm. The open-source SQL server can be a great candidate, not only does it inherit the authentication mechanism but also it improves the MsBOT part information security.

The part matching routine in the MsBOT Health Inventory is pure sequential strings matching algorithms. It is a simplistic validation algorithm; it is meant for the sake of proofing the concepts. However, it needs a robust auto-matching algorithm that able to identify what part is and for what robotic system, and finally pull the appropriate part information that characterises the configuration according to the robotic system.

Figure 29 above summarise the new processing routines for the above-mentioned future research activities, the Hardware Information Module Libraries, the SQL Database Manager and the Auto-matching Manager. Figure 29 shows that when a DDS-RTPS request message carries the robotic component's part information to the MsBOT Master, the Auto-matching Manager of the Heath Inventory matches the part information stored in the Hardware Information Module (HIM) database in SQL. Once a matched part is found, it passes the control back to the Auto-matching Manager for registering the part into the health inventory database in SQL. However, instead of accessing the database itself, it passes to the SQL Database Manager.

The predefined robotic part information libraries define what the robot is. The information of each part is used for validating against the connected component's part information. To create a new robotic system which can be achieved by creating a robotic component database and filling up the components' information. Similarly, to nullify a robotic system, simply remove the component database. However, only authorised administrators can directly access the database system. Hence, to avoid unwanted consequences, employing a robust and secured database management system is justified.

Finally, the current Health Inventory system does not allow users to override the predefined setting. However, in the real world, due to unforeseen circumstances, users' intervention is inevitable. Hence, a new module called Manual Configuration Management System (MCMS), similar to the Access Point configuration system, displays the setting values through HTML pages and allows users to configure the intended values.

5.5 The Application of MsBOT

The MsBOT is a perfect candidate for educational purposes. For the reasons:

Versatile – The MsBOT framework allows users to define their own robotic systems. In facts, the framework is tasked to facilitate the MsBOT 2-step Plug-and-Play, which comprises the MsBOT information validation and the Health Inventory system.

Plug-and-Play – Teachers and Students can navigate the system without needing to know the configurational details. However, some understanding of how the MsBOT Piggyback Plug-and-Play framework work is sufficient.

Easy – Adding and removing a robotic system is achieved simply by creating a robotic component database or removing the existing robotic component database respectively.

Convenient – The MsBOT Piggyback Plug-and-Play framework is a convenient platform to learn, study and practise DDS, XRCE-DDS robotic system. Simply use the MsBOT framework to create a new robotic system as you please using the microcontrollers or PCs. The Micro-ROS supports Microsoft Windows, Linux and macOS.

Safe and Secured– Robotic systems created by the MsBOT framework are safe and secured from unsolicited access due to the versatile Health- Inventory system. Cross access to other robotic systems is prevented.

A wide set of Microcontrollers – Micro-ROS supports many manufacturers' microcontrollers. The main targets of micro-ROS are mid-range 32-bits microcontroller families. Hence, it is high chances to the needs of teachers and students.

Hardware and Operating System Agnostic – Robotic components that are made by different hardware systems and run on different operating systems can still work together in the MsBOT framework.

5.6 Conclusion

MsBOT is the acronym of **M**as**s**ey ro**BOT**. The MsBOT project aspired to implement an open-Source Plug-and-Play distributed robotic framework for education with the intention to rectify the meticulous configuration and the weaknesses of inherent vertically integrated educational robotic systems using open-source solutions. The MsBOT adopts a 2-Step Piggyback Plug-and-Play strategy to realise the MsBOT Plug-and-Play framework for education. It leverages the open-source Micro-ROS robotics technology that provides the Plug-and-Play feature for auto-discovery connecting Micro-ROS Clients to Micro-ROS Agent. The CAN standard classic 2.0A mode of an 8-byte long payload is too little to achieve a smooth, reliable communication stream. When used on the inherently large XRCE-DDS messaging protocols, the strained bandwidth conditions worsen.

The MsBOT 2-Step Piggyback Plug-and-Play implementation has succeeded in implementing an open-source Plug-and-Play distributed robotic framework for education. The test and verification results have proven that the framework does Plug-and-Play

eliminating human intervention. With that framework, teachers and students can replicate to suit teaching and learning needs at ease.

References

[1] D. Kumar and L. Meeden, "A Robot Laboratory for Teaching Artificial Intelligence," vol. SIGCSE Bulletin, Volume 30, Number 1, 1998, pp.341-344., 1998.

[2] M. A. Garcia and H. Patterson-McNeill, "Learn how to develop software using the toy Lego Mindstorms.," 2002.

[3] P. Seymour, "Mindstorms: Children, computers, and powerful ideas.," 1980. Basic Books, Inc. NY, New York, 1993.

[4] E. B. B. Gyebi, M. Hanheide, and G. Cielniak, "Affordable Mobile Robotic Platforms for Teaching Computer Science at African Universities.," in *6th International Conference on Robotics in Education*, Yverdon-les-Bains, 2015. [Online]. Available: https://core.ac.uk/download/pdf/29202979.pdf. [Online]. Available: https://core.ac.uk/download/pdf/29202979.pdf

[5] S. C. Silk E., "Using Robotics to Teach Mathematics: Analysis of a Curriculum Designed and Implemented (AC2008-1261)," *American Society for Engineering Education, Pittsburgh* 2008.

[6] W. B. Gerecke U., Hohmann P. "A Modular Educational Robotic Toolbox to Support University Teaching Efforts in Engineering,." https://pdfs.semanticscholar.org/048c/95d5347854122c56a0ae59cebf628b796548.pdf (accessed.

[7] C. C. Chung C.J. Chung, Cole M. . "Assessing the Impact of an Autonomous Robotics Competition for STEM Education,." http://www.jstem.org/index.php/JSTEM/article/view/1704/1606 (accessed.

[8] M. Kochlan and M. Hodon, "Open hardware modular educational robotic platform - Yrobot," *23rd International Conference on Robotics in Alpe-Adria-Danube Region, IEEE RAAD 2014 - Conference Proceedings*, 01/06 2015, doi: 10.1109/RAAD.2014.7002246.

[9] Wikipedia, "Vendor Lock-in," May 2022. [Online]. Available: https://en.wikipedia.org/wiki/Vendor_lock-in.

[10] M.Jändel, "Plug-and-Play Robotics." [Online]. Available: https://www.foi.se/download/18.7fd35d7f166c56ebe0b1005e/1542623791562/Plug-and-play-robotics_FOI-S--3702--SE.pdf.

[11] A. Sorrentino, F. Cavallo, and L. Fiorini, "A Plug and Play Transparent Communication Layer for Cloud Robotics Architectures," *Robotics*, vol. 9, 03/22 2020, doi: 10.3390/robotics9010017.

[12] M. B. T. Haidegger, P. Gonçalves, M. K. Habib, S. K. V. Ragavan, H. Li, A. Vaccarella, R. Perrone, and E. Prestes, "Applied ontologies and standards for service robots," vol. 61, p. Robotics and Autonomous Systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188901300105X.

[13] I. Zamalloa, I. Muguruza, A. Hernandez, R. Kojcev, and V. Mayoral, "An information model for modular robots: the Hardware Robot Information Model (HRIM)," 02/05 2018.

[14] OMG, "The Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS) Specification," formal/2022-04-01 2022. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.5/PDF

[15] C. Pons, G. Perez, R. Giandini, and G. Baum, *Applying MDA and OMG Robotic Specification for Developing Robotic Systems*. 2016, pp. 51-67.

[16] T. Kotoku and M. Mizukawa, "Robot Middleware and its Standardization in OMG - Report on OMG Technical Meetings in St. Louis and Boston," in *2006 SICE-ICASE International Joint Conference*, 18-21 Oct. 2006 2006, pp. 2028-2031, doi: 10.1109/SICE.2006.315486.

[17] M. N. Mayangsari, Y. Kwon, S. C. Ahn, and J. Park, "Robotic Technology Component with UPnP communication (RTU Component)," in *2008 International Conference on Control, Automation and Systems*, 14-17 Oct. 2008 2008, pp. 1647-1652, doi: 10.1109/ICCAS.2008.4694496.

[18] N. Ando, "The Robotic Technology Component (RTC) and Its Relevant Specifications Standardization Activities in OMG," *Journal of the Robotics Society of Japan*, vol. 29, no. 4, pp. 333-336, 2011, doi: 10.7210/jrsj.29.333.

[19] W. Woodall. "ROS on DDS." Open Source Robotic Foundation Inc. https://design.ros2.org/articles/ros_on_dds.html (accessed 13 Oct 2021, 2021).

[20] "ROS 2 Design." https://design.ros2.org/ (accessed.

[21] G. Yoon, J. Choi, H. Park, and H. Choi, "Topic naming service for DDS," in *2016 International Conference on Information Networking (ICOIN)*, 13-15 Jan. 2016 2016, pp. 378-381, doi: 10.1109/ICOIN.2016.7427138.

[22] T. Harrison, D. Levine, and D. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, vol. 32, 09/02 2000, doi: 10.1145/263700.263734.

[23] OMG. "Common Object Request Broker Architecture." https://www.omg.org/spec/CORBA/3.4/About-CORBA/ (accessed.
[24] A. B. A.Makarenko, T. Kaupp "Orca: Components for Robotics." [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.7562&rep=rep1&type=pdf.
[25] Orca, "http://orca-robotics.sourceforge.net/index.html," 2022. [Online]. Available: http://orca-robotics.sourceforge.net/index.html.
[26] W. Hasselbring, *Component-based software engineering* (Handbook of Software Engineering and Knowledge Engineering). World Scientic Publishing Company, 2002.
[27] Wikipedia. "Data Distributed Services." https://en.wikipedia.org/wiki/Data_Distribution_Service (accessed 15 Oct 2021, 2021).
[28] "eProsima Micro XRCE-DDS." https://www.eprosima.com/index.php/component/ars/repository/eprosima-micro-xrce-dds (accessed.
[29] eProsima. "Introduction to XRCE." https://www.eprosima.com/index.php/resources-all/whitepapers/xrce (accessed.
[30] eProsima. "micro-ROS." https://micro.ros.org/ (accessed.
[31] "micro-ROS -- puts ROS 2 onto microcontrollers." https://micro-ros.github.io// (accessed.
[32] @FIWARE, "micro-ROS - Putting ROS 2 onto Microcontrollers - FIWARE," 2019-08-13 2019. [Online]. Available: https://www.fiware.org/2019/08/13/micro-ros-putting-ros-2-onto-microcontrollers/.
[33] T. k. d. community, "SocketCAN - Controller Area Network," 14 Mar 2022. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/can.html.
[34] Wikipedia, "SocketCAN," 2021. [Online]. Available: https://en.wikipedia.org/wiki/SocketCAN.
[35] L. Chow, "MsBOT Source Code," May 2022. [Online]. Available: https://github.com/vibnwis.
[36] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Technical report, EBSE Technical Report EBSE-2007-01, 2007 2007. Accessed: 2014/08/08/00:00:00. [Online]. Available: https://www.cs.auckland.ac.nz/~norsaremah/2007%20Guidelines%20for%20performing%20SLR%20in%20SE%20v2.3.pdf
[37] E. Macedo *et al.*, "On the security aspects of Internet of Things: A systematic literature review," *Journal of Communications and Networks*, vol. PP, pp. 1-14, 10/01 2019, doi: 10.1109/JCN.2019.000048.
[38] F. Benitti, "Exploring the educational potential of robotics in schools: A systematic review," *Computers & Education*, vol. 58, pp. 978-988, 04/01 2012, doi: 10.1016/j.compedu.2011.10.006.
[39] U. Gerecke, "A Modular Educational Robotic Toolbox to Support University Teaching Efforts in Engineering," 2003.
[40] F. López-Rodríguez and F. Cuesta, "Andruino-A1: Low-Cost Educational Mobile Robot Based on Android and Arduino," *Journal of Intelligent & Robotic Systems*, vol. 81, 05/03 2015, doi: 10.1007/s10846-015-0227-x.
[41] A. Takacs, G. Eigner, L. Kovacs, I. J. Rudas, and T. Haidegger, "Teacher's Kit: Development, Usability, and Communities of Modular Robotic Kits for Classroom Education," *IEEE Robotics & Automation Magazine*, vol. 23, no. 2, pp. 30-39, 2016, doi: 10.1109/MRA.2016.2548754.
[42] P. Engineer, "Home Automation Robots - Changing the World, One Robot at a Time," *Progressive Automations*, 2021. [Online]. Available: https://www.progressiveautomations.com/blogs/news/home-automation-robots-changing-the-world-one-robot-at-a-time.
[43] B. Jamie, "Four types of medical robot leading the way in healthcare… they are taking over, but they're here to help," *NS Healthcare*, 2020. [Online]. Available: https://www.ns-healthcare.com/analysis/medical-robots/.
[44] Y.-G. Ha, J.-C. Sohn, Y.-J. Cho, and H. Yoon, "Towards Ubiquitous Robotic Companion: Design and Implementation of Ubiquitous Robotic Service Framework," *Etri Journal - ETRI J*, vol. 27, pp. 666-676, 12/10 2005, doi: 10.4218/etrij.05.0105.0138.
[45] "About OPEN ROBOT HARDWARE." https://www.openrobothardware.org/sample-page/ (accessed.
[46] P. Christensson, "Plug and PLay," August 28 2021 2006. [Online]. Available: https://techterms.com/definition/plugandplay#:~:text=Plug%20and%20Play%2C%20sometimes%2C%20abbreviated,soon%20as%20they%20are%20connected.
[47] D. H. A.Hoek, A. L. Wolf, "Software Architecture, Configuration Management, and Configurable Distributed Systems: A M´enage a Trois," University of Colorado, 1998.
[48] C. R. Westfechtel B., "Software Architecture and Software Configuration Management," *In: Westfechtel B., van der Hoek A. (eds) Software Configuration Management*, 2003, doi: SCM 2001, SCM 2003.

[49] F. Cugini *et al.*, "Toward Plug-and-Play Software-Defined Elastic Optical Networks," *Journal of Lightwave Technology*, vol. 34, no. 6, pp. 1494-1500, 2016, doi: 10.1109/JLT.2015.2511802.
[50] AIST. "RT-Middleware : OpenRTM-aist version 1.0 has been released." https://www.aist.go.jp/aist_e/list/latest_research/2010/20100210/20100210.html (accessed.
[51] M. Quigley *et al.*, *ROS: an open-source Robot Operating System*. 2009.
[52] Wikipedia. "Robot Operating System." https://en.wikipedia.org/wiki/Robot_Operating_System (accessed.
[53] CMake. "Build with CMake. Build with Confidence." https://cmake.org/ (accessed.
[54] K. Kamei. "STANDARDIZATION ACTIVITIES FOR ROBOTIC SERVICES." IEEE.org. https://www.standardsuniversity.org/e-magazine/december-2017/standardization-activities-robotic-services/ (accessed 12 Oct 2021, 2021).
[55] D. Foundation. "What's in the DDS Standard?" Object Management Group. https://www.dds-foundation.org/omg-dds-standard/ (accessed 14 Oct 2021, 2021).
[56] OMG. "ABOUT THE RPC OVER DDS SPECIFICATION VERSION 1.0." OMG. https://www.omg.org/spec/DDS-RPC/1.0/About-DDS-RPC/#:~:text=This%20specification%20defines%20a%20Remote,to%20provide%20request%2Freply%20semantics. (accessed 13 Oct 2021, 2021).
[57] P. Krzyzanowski. "Remote Procedural Calls." https://people.cs.rutgers.edu/~pxk/416/notes/15-rpc.html (accessed.
[58] RTi. "About Real-Time Innovations." https://www.rti.com (accessed.
[59] eProsima. "FastDDS." https://www.eprosima.com/index.php/products-all/eprosima-fast-dds (accessed.
[60] A. Kampmann, A. Wustenberg, B. Alrifaee, and S. Kowalewski, *A Portable Implementation of the Real-Time Publish-Subscribe Protocol for Microcontrollers in Distributed Robotic Applications*. 2019, pp. 443-448.
[61] FreeRTOS. "Real-time operating system for microcontrollers." https://www.freertos.org/ (accessed.
[62] lwIP. "Lightweight IP stack." https://www.nongnu.org/lwip/2_1_x/index.html (accessed.
[63] eProsima. "eProsima Micro CDR." https://github.com/eProsima/Micro-CDR (accessed.
[64] foxglove. "cdr." https://github.com/foxglove/cdr#:~:text=Common%20Data%20Representation%20(CDR)%20defines,handling%20endianness%20and%20alignment%20requirements. (accessed.
[65] OMG. "OMG: RPC Over DDS (DDS-RPC)." https://www.omgwiki.org/dido/doku.php?id=dido:public:ra:xapend:xapend.b_stds:tech:omg:dds_rpc (accessed 14 Oct 2021, 2021).
[66] Wiki. "FreeRTPS." **https://github.com/ros2/freertps/wiki** (accessed.
[67] RTi. "RTi Connect Nano." https://www.rti.com/developers/rti-labs/connext-nano (accessed.
[68] T. Kaupp, A. Brooks, B. Upcroft, and A. Makarenko, *Building a Software Architecture for a Human-Robot Team Using the Orca Framework*. 2007, pp. 3736-3741.
[69] G. K. S.Haefliger, S.Spaeth, "Code Reuse in Open Source Software," *Management Science 54(1):180-193.*, Nov 2007, doi: https://doi.org/10.1287/mnsc.1070.0748.
[70] C. Szyperski, "Component Software," 1988. Addison-Wesley, Harlow, England,.
[71] M. S. M. Henning, "Distributed Programming with Ice," 2006. [Online]. Available: http://www.zeroc.com/ice.html.
[72] S. Enderle, H. Utz, S. Sablatnög, S. Simon, G. Kraetzschmar, and G. Palm, "Miro: Middleware for Autonomous Mobile Robots," *IFAC Proceedings Volumes*, vol. 34, no. 9, pp. 297-302, 2001/07/01/ 2001, doi: https://doi.org/10.1016/S1474-6670(17)41721-6.
[73] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro - middleware for mobile robot applications," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 493-497, 2002, doi: 10.1109/TRA.2002.802930.
[74] W. contributors. " OSI model." https://en.wikipedia.org/w/index.php?title=OSI_model&oldid=1059640397 (accessed.
[75] D. C. Schmidt, "ACE: an Object-Oriented Framework for Developing Distributed Applications," 1994.
[76] D. Krüger, I. Lil, N. Sünderhauf, R. Baumgartl, and P. Protzel, "Using and Extending the Miro Middleware for Autonomous Mobile Robots," 01/01 2006.
[77] S. C. Ahn, K. Lim, J.-W. Lee, H. Ko, Y.-M. Kwon, and H.-G. Kim, "UPnP Robot Middleware for Ubiquitous Robot Control," 2006.
[78] O. C. Foundation. "ABout UPnP." https://openconnectivity.org/search/upnp/ (accessed.
[79] DLNA. "Connect and Enjoy." https://www.dlna.org/ (accessed.
[80] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for Robotics: A Survey," in *2008 IEEE Conference on Robotics, Automation and Mechatronics*, 21-24 Sept. 2008 2008, pp. 736-742, doi: 10.1109/RAMECH.2008.4681485.

[81] F. Arvin, J. Espinosa, B. Bird, A. West, S. Watson, and B. Lennox, "Mona: an Affordable Open-Source Mobile Robot for Education and Research," *Journal of Intelligent & Robotic Systems*, vol. 94, no. 3, pp. 761-775, 2019/06/01 2019, doi: 10.1007/s10846-018-0866-9.
[82] Wikipedia. "Object Linking and Embedding." https://en.wikipedia.org/wiki/Object_Linking_and_Embedding (accessed.
[83] S. Kumar, A. Mall, R. Awasthi, K. Tripathi, and S. Lakshmi, "Component Object Model: An Overview & Practical Implementation," vol. 3, pp. 3-2017, 03/07 2017.
[84] Wikipedia, "Microsoft Robotics Developer Studio." [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Robotics_Developer_Studio.
[85] S. Corporation. "Sony Develops OPEN-R Architecture for Entertainment Robots." https://www.sony.com/en/SonyInfo/News/Press_Archive/199806/98-052/ (accessed.
[86] OMG, "DDS for eXtremely Resource Constrained Environments," 2019. [Online]. Available: chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.omg.org/spec/DDS-XRCE/1.0/PDF.
[87] SeeedStudio, "2 Channel CAN BUS FD Shield for Raspberry Pi," *Seedd Wiki*. [Online]. Available: https://wiki.seeedstudio.com/2-Channel-CAN-BUS-FD-Shield-for-Raspberry-Pi/.
[88] eProsima, "FastRTPS," 2022. [Online]. Available: https://fast-dds.docs.eprosima.com/en/v1.7.0/.
[89] C. I. Out, "Serial Wire Viewer (SWD + SWO)," *Code Inside Out*, 2021. [Online]. Available: https://www.codeinsideout.com/blog/stm32/swv/.
[90] eProsima. "eProsima Micro-XRCE-DDS-Apps." https://github.com/eProsima/Micro-XRCE-DDS-Apps/tree/microxrce_dev/FreeRTOS (accessed 2023).
[91] FraFin, "Discovery in Micro-ROS," May 2022. [Online]. Available: https://discourse.ros.org/t/discovery-in-micro-ros/16611.
[92] Wikipedia, "PDCA." May 2022. [Online]. Available: https://en.wikipedia.org/wiki/PDCA.