卒業論文

軽量 ROS ランタイム mROS 2 の評価に向けたロボット制御アプリケーションの実装

公立はこだて未来大学
システム情報科学部　複雑系知能学科
知能システムコース　1020259

中村 碧

指導教員　　松原 克弥

提出日　　2024 年 1 月 25 日

BA Thesis

Implementing Robot Applications Toward Evaluating 'mROS 2' a Lightweight ROS Runtime

by

Aoi Nakamura

Intelligent Systems Course, Department of Complex and Intelligent Systems
School of Systems Information Science, Future University Hakodate

Supervisor:　　Katsuya Matsubara

Submitted on January 25th, 2024

Abstract–

In the realm of robot software development, the utilization of ROS has been on the rise. While the construction of robot software harnessing ROS often emphasizes cloud-based distributed robot systems, there are challenges in the flexibility of node placement. Specifically, the dynamic repositioning of nodes becomes complex due to differences in CPU architectures between the cloud and robots. Previous studies have pointed out the increase in overhead with dynamic placement mechanisms. Although approaches using mROS 2-POSIX have been proposed, their performance evaluations are mostly confined to microbenchmarks of network throughput, and the efficacy in real applications remains under-assessed. This study aims to compare the performance of mROS 2-POSIX with ROS 2. Based on experimental results, I demonstrate that mROS 2-POSIX can deliver the anticipated performance even in sophisticated applications.

Keywords: cloud robotics, Robot Operating System, WebAssembly

概 要： ロボットソフトウェア開発において，ROS の利用が増加している．ROS を活用したロボットソフトウェアの構築ではクラウドを用いた分散ロボットシステムが注目されているが，ノード配置の柔軟性が課題となっている．特に，クラウドとロボットの CPU アーキテクチャの違いから，動的なノード再配置が難しい．先行研究での動的配置機構はオーバーヘッドの増加が問題とされ，mROS 2-POSIX を用いたアプローチも提案されているが，性能評価はネットワークスループットのマイクロベンチマークに留まり，実アプリケーションにおける有用性が十分に評価されていない．本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価する．実験結果によって得た通信性能やメモリサイズをもとに，mROS 2-POSIX が複雑なアプリケーション上でも期待される性能を発揮できることを示す．

キーワード： クラウドロボティクス,Robot Operating System,WebAssembly

目次

第 1 章

はじめに

さまざまな産業向けやエンターテイメント関連のロボットシステム，自動車の自動運転技術や IoT システムのソフトウェア開発をサポートするフレームワークとして Robot Operating System（以下，ROS）の普及が進んでいる [1]．ROS のプログラミングモデルは，システムの各機能を独立したプログラムモジュール（ノード）として設計することにより，汎用性と再利用性を向上させて，各機能モジュール間のデータ交換を規定することで，効率的かつ柔軟なシステム構築を可能にしている．たとえば，カメラを操作して周囲の環境を撮影するノード，画像からオブジェクトを識別するノード，オブジェクトのデータをもとに動作制御を実行するノードを連携させることで，自動運転車の基本機能の一部を容易に実装できる．

ROS のプログラミングモデルは，ロボット/IoT とクラウドが協力する分散型システムにおいても有効である．ロボットシステムのソフトウェア処理は，外界の情報を取得する「センサー」，取得した情報を処理する「知能・制御系」，実際に動作するモーターなどの「動力系」の 3 要素に分類できる [2]．クラウドロボティクス [3] において，主に知能・制御系のノードを高い計算能力を持つクラウドに優先して配置することで，高度な知能・制御処理の実現を促進できる．さらに，ロボットが取得した情報や状態などをクラウドに集約・保存することで，複数のロボット間での情報共有と利用を容易にする．一方で，現行の ROS 実装では，各ノードの配置をシステム起動時に静的に設定する必要があり，クラウドとロボット間の最適なノード配置を事前に設計する必要がある．しかし，実際の環境で動作するロボットは，ネットワークの状況やバッテリー残量の変動など，システム運用前に予測することが困難な状況変化に対応する必要があり，設定したクラウドとロボット間のノード配置が最適でなくなる可能性がある．このような状況変化への対応として，ノードを動的に再配置するライブマイグレーション技術があるが，多くの場合でクラウドとロボット間の CPU アーキテクチャが異なり，命令セットがそれぞれ違うため，実行中のノードをシステム運用中にマイグレーションすることは技術的に困難である．

菅ら [4] は，WebAssembly（以下，Wasm）を用いることで，クラウドとロボット間での

実行状態を含む稼働中ノードの動的なマイグレーションする手法を実現した．Wasm とは Web 上で高速にプログラムを実行するために設計された仮想命令セットアーキテクチャのことで，1 つのバイナリが複数のアーキテクチャで動作するため，異種デバイス間でのマイグレーションに適しているといえる．課題として，ROS 2 を Wasm 化したことでライブマイグレーション後のファイルサイズのオーバーヘッドが増大し，ノードの実行時間が大幅に増えてしまう問題が残った．柿本ら [5] は，組込みデバイス向けの軽量な ROS 2 ランタイム実装である mROS 2-POSIX を採用し，ROS ランタイムの Wasm 化にともなうオーバヘッド増加に対処した．しかし，採用された mROS 2-POSIX 上で指定されたメッセージを往復させるシンプルなアプリケーション上でしか評価実験はされていない [6]．そのため，ライブマイグレーション後のオーバーヘッド増加を解決するロボットソフトウェア基盤として，アプリケーションが複雑化した場合の動作が明らかでない．

本研究では，クラウドとロボット間での実行状態を含む稼働中ノードの動的マイグレーションの実現に向けた mROS 2-POSIX の性能評価を行い，mROS 2 が ROS 2 と比べて動的配置機構ロボットソフトウェア基盤としてどの点で優位性があるのか明らかにすることを目指す．

第 2 章

使用した技術

2.1 ROS 2

ロボットシステムを開発するにあたって現在は ROS 2 が主流であるが，その前身である ROS は，スタンフォード人工知能研究所の研究プロジェクトとから移管された Willow Garage 社によって開発が始まったロボットソフトウェア開発基盤である．最初の正式なディストリビューション版は, 2010 年 3 月にリリースされた Box Turtle で, その後, Fuerte, Groovy, Hydro, Indigo, Jade, Kinetic, Lunar, Melodic, Noetic, Foxy, Galactic, Humble といったバージョンがリリースされている．ROS の利点は，分散型ロボットシステムの実現に向いている通信ミドルウェアの実装や RTPS（Real Time Publish Subscribe）通信プロトコル，プロジェクト管理やデバックおよびシミュレーションなどのための広範囲なツール群，豊富な OSS（Open Source Software）のパッケージやライブラリ，世界規模の活発なオープンソース開発コミュニティという 4 つの側面にある [8].

パッケージやライブラリに関しては公式のものだけでも多種多様であり, ロボット本体の制御, モーターの制御, センサーの制御, 画像処理, 音声認識, 自律走行, SLAM（Simultaneous Localization and Mapping）など，ロボットシステムに必要な機能を網羅している．短期間で機能の大枠を組み立てることができるため，研究開発，教育，産業用途においても，幅広い分野で利用されている．

ROS すでに 10 年以上の歴史を持っており，ロボット工学の発展に大きく貢献してきた．日本の企業ではソニーの aibo の事例が ROS を使った製品として代表的であり，他の企業でも商用商品に採用されることも多い．ロボット開発を取り巻く環境や ROS が研究用から商用にも活用され始めるという変遷を受け，2014 年より第 2 世代バージョンである ROS 2 の開発が始まった．これは ROS 1 は研究用途においては十分な性能を持っていたが，商用製品においてはリアルタイム性やセキュリティの問題があった．ROS 1 の設計は以下のようにされている．

- 単一のロボットで動作することを想定

- 多くのリソースを抱えているマシン上で動作されることを想定
- リアルタイム性は考慮しない
- 優れたネットワーク上で動作されることを想定（有線接続など）
- 研究，主に学術的なアプリケーションを想定
- 規制や禁止事項がなく，最大限の柔軟性がある（プログラムが main() から始まることを決定していないなど）

そのため，産業用に利用されることが増えるとこの設計が問題になることが多かった．

ROS 1 が抱える課題として以下のようなものがある．

1. メッセージ通信の仲介役として ROS Master が必要である．ROS Master が何らかの原因でダウンすると，全てのプロセスがダウンしてしまうという問題がある．
2. リアルタイム性を考慮していない設計になっている．
3. メッセージを暗号化せずに送受信を行っているため，セキュリティに問題がある．
4. 実行するマシンにリソースが少ないと動作が不安定になるという問題がある．
5. Ubuntu（Linux）でないと使いにくい

ROS 2 では，ROS 1 の上記の問題を解決するため，リアルタイムおよび組込みシステム向けの DDS（Data Distribution Service）[7] と呼ばれる通信ミドルウェアを採用し，通信の信頼性を確保するための QoS（Quality of Service）制御の機能を導入した．

この DDS の採用により，(1) の問題点がなくなり，QoS と DDS の組み合わせにより，(2) に対してリアルタイム性の実現を目指した．(3) のセキュリティの問題は ROS 2 でメッセージの暗号化が施されたことにより解決した．また，(4) に対応するためにリソースの少ないマシンで動作できるように開発されている．さらに Ubuntu 以外の OS にも対応した．

ROS 2 の構成を図 1 に示す．ROS 2 では図のようにユーザーが実装するアプリケーション層があり，その下にクライアントライブラリである rcl の層がある．rcl クライアントライブラリ層には通信概念を公開する API の定義がされており，DDS の上に構築された．rcl は，C++，Python といった各プログラミング言語用のクライアントライブラリに共通する機能を提供している API である．その下部に DDS と rmw（ROS Middleware Interface）がある．DDS はリアルタイム通信を行うための API とプロトコルを提供している国際ミドルウェア標準 []．rmw は様々な種類の DDS 実装，各 backend の差を吸収，最適化を施している層であり，rcl クライアントライブラリに対して共通のインタフェースを提供している．RTPS は DDS から呼び出される tranceport 層に位置する通信プロトコルである．backend として，RTOS や POSIX，NoC などの ROS 2 は様々な環境で動作するためを表す層がある．

このように，従来の ROS 1 では難しかったロボットシステムでも ROS 2 を使うことで，個々のロボットが独立して動作するだけでなく，複数のロボットが協調して動作する分散型

ロボットシステムの開発が可能になった.

2015 年 8 月には最初のディストリビューションであるアルファ版がリリースされ，2017 年の 12 月に ROS 2 が本リリースされた．2023 年の Metrics Report[] によると，ROS は 550,365,601 回ダウンロードされている．そのうち 30% が ROS のディストリビューションである Noetic で，32% が ROS 2 のディストリビューションである humble である．2023 年で ROS は Noetic，ROS 2 は humble が主流のディストリビューションであった．本研究は ubuntu 22.04 でリリースされた ROS 2 の humble ディストリビューションを使用している.

2.1.1 Publisher と Subscriber

ROS では基本的なノード間のデータ通信として Publish/Subscribe（Pub/Sub）通信型の非同期な通信プロトコルを採用している．データの送信側を Publisher（出版者）とよび，受信側を Subscriber（購読者）と呼ぶ．通信経路として，トピック（Topic）を介した通信が行われる．トピックで送受信されるデータはメッセージ（Message）と呼ばれ，車輪の角速度や回転量，現在位置の 3 次元座標など，基本型を組合せた任意の型を定義することができる．同じ名前のトピックに対して，様々な個数や種類のノードが任意のタイミングで登録，変更，削除ができる．さらに，メッセージの型が一致していれば，通信が行われる．Publisher 側は自由なタイミングで Topic に向けて通信を行い，非同期に動作する Subscriber 側は，Publisher からメッセージを受け取った際に対応するコールバック関数が実行される．ノードは Publisher や Subscriber，またその両方をプログラム次第で変化できるため，ユーザが考えたオリジナルのノードを作成することができる．この仕様のため，ノード同士の依存が少なくなり，ロボットシステム全体の機能の追加や削除が容易である特性は，柔軟なロボットシステムの構築の一助となる．ROS 2 で Publisher を作成する際, C++ で rclcpp::Node クラスの継承から create_publisher(), Python で import rclpy 後に.create_publisher() を行うことで Publisher を作成することができる．同様に Subscriber は C++ で rclcpp::Subscription クラスを使用して create_subscriber(),Python で import rclpy 後に create_subscriber を記述することで Subscriber を作成できる.

ROS 2 の通信ミドルウェアである DDS と通信プロトコルである RTPS は，通信相手の探索および通信経路の確立を自律的に行う．この機能の実現は，RTPS の SPDP（Simple Participant Discover Protocol）と SEDP（Simple Endpoint Discover Protocol）[9] というプロトコルによって行われる．ここで，RTPS では，ROS 2 のノードに相当するものを Participant（参加者）と呼ぶ．通信相手を探索するためには自身の情報を送信するモジュールを Writer，ほかの Participant から情報を受け取るモジュールを Reader と呼ぶ．SPDP は，Participant の情報を送受信するためのプロトコルであり，SEDP は，Topic の情報を送受信するためのプロトコルである．この通信のエンドポイントは，Participant 同士のパブ

リッシュ，サブスクライブである．

RTPS を OSI 参照モデルに例えると transport 層に位置し，UDP/IP の上に実装されている．UDP 通信にはパケットの到着に関する保証がないが，ROS 2 の DDS ではこれを補助する QoS（Quality of Service）制御の機能がある．QoS 制御は，サブスクライバ，パブリッシャごとに設定でき，厳格な条件であれば RELIABLE（信頼性の高い通信）を，緩やかな条件であれば BEST EFFORT（ベストエフォート通信）を選択することができる．ROS 2 のデフォルト DDS である FastRTPS[10] では，QoS 制御の機能が実装されており，各ノードが通信するときの QoS 設定は RELIABLE である．

2.1.2 ノードの作り方とパッケージ

ROS 2 においてノードを作成する場所は決まっており，一般的には ros2_ワーキングスペース（ws）というディレクトリを作った後，その中に src というディレクトリを作成し，その中にノードを作成する．src ディレクトリを作成した後，ros2_ws ディレクトリで colcon build というコマンドを実行することで，ros2_ws の中でノードを作成することがきる．ROS 2 の場合，ノードの集まりのことをパッケージと呼ぶ．パッケージは ros2 pkg create というコマンドを実行することで作成することができ，src ディレクトリの直下で実行することで，パッケージを作成することができる．パッケージを作成する前にそのパッケージの中で使用する言語を決める必要がある．ROS 2 は C++ と Python の 2 つの言語をサポートしており，ユーザーの好みに合わせて変更できる．

2.1.3 オーバーレイとアンダーレイ

また，ROS 2 には重要な概念にアンダーレイとオーバーレイがある．アンダーレイは，完成されたパッケージをインストールするワークスペースであり，安定した環境をオーバーレイに提供するためにある．オーバーレイは，ユーザー自身で作成したパッケージを扱うワークスペースであり，先ほどの ros2_ws はオーバーレイにあたる．ROS 2 ではユーザーが作るノードやパッケージをオーバーレイに作成し，必要に応じてアンダーレイのパッケージを参照して使用するのが一般的である．このオーバーレイ上でユーザは colcon build を実行し，パッケージをビルドする．ユーザーはパッケージをビルドした後，すぐに実行することができない．ROS 2 では source コマンドを用いてオーバーレイ環境を読み込むことでアンダーレイがオーバーレイより優先されることなく，開発することができる．ROS 2 においてオーバーレイとアンダーレイは，複雑化するロボットシステムに柔軟性と拡張性をもたらす概念であることがわかる．ROS 2 のデバック方法として様々なコマンドが用意されており，ros2 topic list というコマンドを実行することで，現在実行されているトピックの一覧を表示することができる．ros2 topic echo は，指定したトピックのメッセージを表示することができる．また，現在実行されているノードを視覚的に確認できるように rqt_graph と呼ばれるも

のが用意されている．開発者は rqt_graph を用いて，目に見えない ROS 2 のノード間の通信を確認することができる．こうしたアンダーレイ機能の充実によって，ROS 2 は ROS 1 よりも柔軟性と拡張性を持つことができた．

2.1.4 ROS 2 が対応している DDS

ROS 2 にはデフォルトで FastRTPS という DDS が実装されている．DDS とは，OMG（Object Management Group）が定めたデータ交換のための仕様である．これによって分散型ネットワークでも効率的に通信が可能になっている．FastRTPS のほかにも，RTI Connext DDS[11] や eProsima Micro XRCE-DDS[12] という DDS が ROS 2 に対応している．Ubuntu22.04 の ROS 2 humble では，FastRTPS はもちろんのこと，デフォルトで Ecripse Cyclone DDS[13]，Gurum DDS[14]，RTI Connext DDS がインストールできる．

2.1.5 Service と Action

ROS 2 では，Publisher とサブスクライバー以外にも Service 通信と Action 通信がある．Service 通信は，サーバーとクライアントの 2 つのノード間でリクエストとレスポンスをやり取りする通信である．これは既存のクライアント・サーバーモデル [15] によく似ており，Subscribe するのではなく，ノードがメッセージの値を欲しいタイミングでリクエストする仕組みになっている．そのため，ロボットに対する命令やデータの取得，計算結果を受け渡しに適しており，Service 通信はリアルタイム性や連続的なデータには向かないが，確実に応答するというロボットシステムにおいて重要な役割を果たす．さらに，Service 通信は同期的な通信であるため，Service 通信を行うノードは，Service 通信が完了するまで他の処理を行うことができない．そのため，クライアントの処理を間接的にブロックすることができる．一方，Action 通信は，Service 通信と同様にサーバーとクライアントの 2 つのノード間でリクエストとレスポンスをやり取りする通信である．しかし，Action 通信は Service 通信と異なり，リクエストに対するレスポンスを即座に返すのではなく，リクエストに対するレスポンスを返すまでの間に，進捗状況を返すことができる．この通信方式によって，長時間のタスクやフィードバックが可能なタスク，中断可能なタスクに適しているため，開発者は柔軟性を保ちながら，ロボットシステムを構築することができる．さらに，Action 通信は非同期的な通信であるため，Action 通信を行うノードは，Action 通信が完了するまで他の処理を行うことができる．これによって，複雑なタスクを行うノードでも Action 通信を用いることで同時に処理することが可能であり，多くのリソースがあるマシンで高度なノードを動作させることができる．

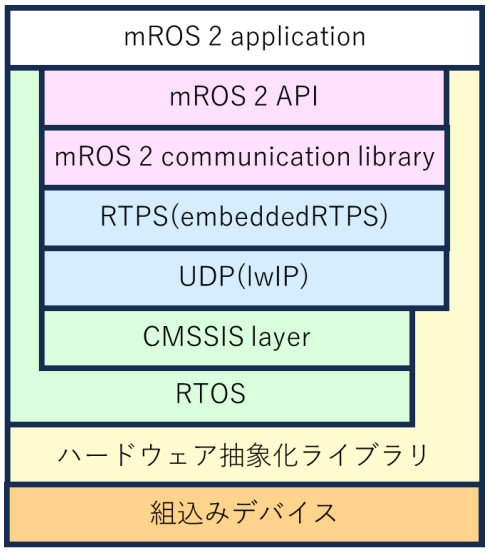

図 2.1: mROS 2 の内部構成

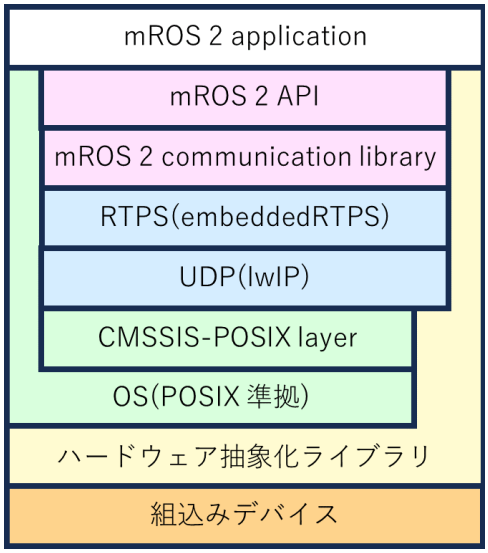

図 2.2: mROS 2-POSIX の内部構成

2.2 mROS 2

mROS 2 は，ROS 2 ノードの軽量実行環境である．ROS 2 を採用するロボットシステムにおいて通信方式とメモリ軽量な実行環境を確立することができる組み込み技術を導入することにより，分散型のロボットシステムにおける応答性やリアルタイム性の向上，消費電力の削減が可能になる．mROS 2 は汎用 OS 上で実行される ROS 2 ノードと通信できることを目的として実装された．そのため，Pub/Sub 通信の相手と経路を自律的に探索できるように設計され，ROS 2 および RTPS の利点が，組み込み技術導入時に損なわれないようになっている．ROS 2 に対応している DDS の種類として，FastRTPS，RTI Connext DDS，eProsima Micro XRCE-DDS を挙げたが，既存の組込みデバイス向けの ROS 2 ノード実行環境である micro-ROS[16] がある．micro-ROS は RTPS の軽量規格である DDS-XRCE（DDS For Extremely Resource Constrained Enviroments）の実装である Micro XRCE-DDS を採用している．Micro XRCE-DDS は，ホストとして ROS 2 ノードを実行するデバイスと通信する際に，Agent ノードと呼ばれるノードの稼働が必要となる．通信の仲介の役割を Agent ノードは担っており，Agent ノードは，ホストデバイス上の ROS 2 ノードと RTPS に則った通信，組み込みデバイス上のノードと XRCE に則った通信をそれぞれ行うため，応答性とリアルタイム性の低下が懸念される．また，複数の組み込みデバイスを用いる場合，Agent ノードは分散型システム全体の通信を仲介するため，Agent ノードの数が増えると通信の遅延が増加する．そのため mROS 2 は，DDS として XRCE-DDS を採用せず，embeddeRTPS[17] を採用している．embeddeRTPS は，一つの Domain クラ

スから Participant，Writer，Reader の 3 つのインスタンスが生成される．つまり，初期化処理の段階で embeddedRTPS の提供する Pub/Sub 通信が利用できるようになる．これによって XRCE-DDS のように Agent ノードが必要なく，組み込みデバイス上での通信の遅延を抑えることができる．この embeddeRTPS が採用されたのは通信の遅延を抑えることができるだけではない．この RTPS は，SPDP と SEDP が実装されており，通信の宛先や受け手として自立性を確保できる RTPS であること点である．また，ROS 2 で代表的な FastRTPS と通信の確認ができているため親和性が高い点も上げられる．以上の設計思想により mROS 2 は，計算資源の限定的な組み込みデバイス上での稼働を想定した組込みデバイスのリアルタイム性の向上および消費電力の削減ができるソフトウェア基盤である．

2.2.1 mros2 の内部構成

図 2.1 に mros2 の内部構成を示す．ユーザーアプリケーションからの階層順で，mROS 2 通信ライブラリ，通信プロトコルスタック，RTOS，ハードウェア抽象化ライブラリによって構成される．mROS 2 通信ライブラリは，ユーザアプリケーションに対して，ROS 2 通信のトピックに関する基本的な API を提供している．主な API として void mros2::init()，void mros2::Node::create_node()，void mros2::Publisher::create_publisher()，void mros2::Subscriber::create_subscriber() がある．通信プロトコルスタックには，C++ で実装された embeddedRTPS を採用している．先ほど述べたようにこの RTPS には SPDP と SEDP が実装されており，計算資源の限定的な組込みデバイス上での稼働を想定した設計であるかつ，ROS 2 の代表的な RTPS である FastRTPS と通信の確認できているという理由がある．UDP については組込み向けの C による軽量実装である lwIP[18] が採用されている．RTOS には，TOPPERS/ASP3 カーネル [19] が採用されており，高分解能タイマやティックレスの低消費電力な処理遅延機能など，高いリアルタイム性と安全性が求められる軽量な組込みシステムに適した設計がなされている．lwip は CMSIS-RTOS API に依存しているため，それぞれの API 差分を吸収するラッパが用意されている．

2.2.2 mROS 2 の通信機能

mROS 2 の通信機能は，mROS 2 の通信ライブラリにある init task（初期化処理）と RTPS/UDP と通信ライブラリを介する write task（パブリッシュ処理）と reader task（サブスクライブ処理）の 3 つのタスクに分けられる．またアプリケーション層にある user task（ユーザーアプリケーション）という開発者が実装するタスクがある．user task は，ROS 2 のノードに相当している．Pub/Sub 通信を行うアプリケーションである．mROS 2 の API を介してパブリッシュやサブスクライブを行うことができる．init task は ROS 2 としてのノードの情報の初期化を行う．APImros 2::init() が呼ばれたときに，対象の組込みデバイスを RTPS の Participant として登録する．writer task と reader task に関しては，パブ

リッシュおよびサブスクライブに関する処理を担う．これらのタスクは user task からの依頼を mROS 2API を介して受け，RTPS の該当機能を立ち上げる．writer task はパブリッシュの依頼を受けて起動し，reader task はサブスクライブの依頼を受けて起動する．

2.2.3 mROS 2-POSIX の内部構成

mROS 2 が POSIX[20] に対応したのが mROS 2-POSIX である．

図 2.2 は，mROS 2-POSIX のソフトウェア構成を示す．mROS 2-POSIX アプリケーション層は，ユーザが実装する ROS 2 ノードに相当する．つまり，ROS 2 におけるオーバーレイに相当する層である．mROS 2-POSIX API 層および通信ライブラリ層は，メッセージを非同期にパブリッシュやサブスクライブするためのコミュニケーションチャネルである ROS 2 の Topic に相当する API および通信機能を提供する階層である．本階層は，ROS 2 のネイティブなクライアント通信ライブラリである rclcpp と互換性を保つように設計されている．mROS 2 通信ライブラリでは，rclcpp のうち pub/sub 通信の基本的な機能のみ実装されている．利用可能な機能は制限されているものの，組込み技術を導入する ROS 2 開発者は，汎用 OS 向けのプログラミングスタイルを踏襲しながら C++ によって mROS 2 のアプリケーションを実装できる．そのため mROS 2-POSIX は Service 通信や Action 通信には対応していない．

RTPS プロトコルスタックには UDP でパブリッシャとサブスクライバ C++ 実装の embeddedRTPS が採用されている．UDP については組込み向けの C 実装である lwIP が採用されている．通信層の embeddedRTPS および lwIP は CMSAIS-POSIX[21] に依存しており，図 1（b）に示す mROS 2 の CMSIS-RTOS を互換した層になっている．最下層にはハードウェアを抽象化したライブラリがある．

mROS 2-POSIX は図 2 に示す実行方式を採用している．リアルタイム OS では，組込みマイコンを実行資源の管理対象として，タスク単位でアプリケーションが実行される．POSIX においてはタスクに相当する概念はプロセスであり，そこから生成されるスレッドを実行単位として処理が進行している．しかし，mROS 2-POSIX は実行単位であるノードに POSIX のスレッドを対応づけ，組込みマイコンでの通信処理におけるイベント割込みについては，POSIX 準拠 OS におけるブロッキング API の発行に相当させて処理している．これらの方式によって，mROS 2-POSIX は POSIX 準拠 OS 上で仮想 ROS 2 ノードとして軽量環境下で実行することができる．

2.2.4 mROS 2-POSIX の動作フロー

mROS 2-POSIX の動作は以下のようになっている．

- netif_posix_add(NETIF_IPADDR, NETIF_NETMASK) によって，lwIP のネット

ワークインターフェースを初期化する．

- osKernelStart() によって，RTOS のカーネルを起動する．
- mros2::init(0, NULL) によって，mROS 2-POSIX の初期化を行う．
- mros2::Node::create_node(ノード名) によって，ノードを生成する．

mROS 2-POSIX は通常の mROS 2 と同様に lwip を利用して UDP stack を実装している．異なる点として，mROS 2 ‐ POSIX は POSIX レイヤが設けられていることが挙げられる．このように明確なレイヤを設けたことによって，netif_posix_add(NETIF_IPADDR, NETIF_NETMASK) のような実装が追加されている．引数に現在 mros2-posix をビルドしているマシンの IP アドレスとサブネットマスクを設定することによって，lwIP のネットワークインターフェースを初期化する．この NETIF_IPADDR と NETIF_NETMASK は，mROS 2-POSIX のヘッダファイルである mros2_posix_netif.h に定義されている．そのためビルド前に ip a などで自分の IP アドレスを確認し，その IP アドレスとサブネットマスクを設定する必要がある．この設定ををを行わないと，ROS 2 やほかの mROS 2 ノードと通信できなくなり，ros2 topic list などのデバックコマンドにも表示されない．

次に，osKernelStart() によって，RTOS のカーネルを起動する．その後，mros2::init(0, NULL) によって，mROS 2-POSIX のノードの初期化を行う．mros2::init() は mROS 2-POSIX のノードの初期化，つまり，ROS 2 としてノード情報を初期化するということである．これは対象の組込みデバイスを RTPS の Participant として登録する．このタスクを行った後，mros2::init() は休止状態に移行する．

そして，mros2::Node::create_node(ノード名) によって，ノードを生成する．この命令に紐づけられたインスタンス変数を介して，Publisher やサブスクライバーを生成する．

2.2.5 mROS 2-POSIX のパブリッシュ処理

mROS 2-POSIX においてパブリッシュ処理は mROS 2 とほとんど変更がない．まず，mros2::Node::create_publisher() によって mROS 2 ノードを Publisher として登録する．mros2::Node::create_publisher は関数テンプレートとして実装されており，第一引数にトピック名，第二引数にパブリッシュするメッセージのキューのサイズを設定する．例として，mros2::Node::create_publisher ＜型名＞ (トピック名,10) という形で記述する．¡型¿の中に入るのは，パブリッシュするメッセージの型である．ROS 2 同様に様々な種類の型を設定でき，ユーザーが自由に定義することができる．同様に”トピック名”にはトピック名が，10 はパブリッシュするメッセージのキューサイズ（履歴の長さ）が入る．定義された publisher がどのようにパブリッシュされるのか以下にフローを示す．

- mros2::Publisher.publish() を呼び出す．引数にはメッセージ情報が格納されたオブジェクトのポインタを渡す．

- mROS 2-POSIX の通信ライブラリ内でメッセージのシリアライズを行い，RTPS に則った形式に変換する．
- embeddedRTPS の提供する機能によって UDP パケットを作成し，タスク間通信で mROS 2-POSIX の Participant の writer にパブリッシュ処理を依頼する
- writer のタスクが実行され，RTPS の SPDP によって，Publisher 情報を送信する．

mROS 2 ‐ POSIX のパブリッシュ処理は，mROS 2 と同様に UDP パケットを作成し，RTPS の SPDP によって，Publisher 情報を送信する．mROS 2 通信ライブラリの通信処理効率化のために writer task は，CPU とネットワーク通信を平行に行うことができる．このため，embeddedRTPS や lwIP におけるメッセージ，UDP パケットの送信処理に関して，user task から分離して writer task で実行されるようになっている．

2.2.6 mROS 2-POSIX のサブスクライブ処理

mROS 2-POSIX においてサブスクライブ処理もパブリッシュ処理と同様にほとんど変わらない．mros2::Node::create_subscription() によって mROS 2 ノードをサブスクライバーとして登録し，メッセージをサブスクライブできるようにする．サブスクリプションも同様に関数テンプレートとして実装されており，テンプレートの仮引数にはメッセージ型，引数にはトピック名と RTPS における QoS のキャッシュに加えて，サブスクライブ時に呼び出されるコールバック関数を指定する．例として，mros2::Node::create_subscription ＜型名＞ (トピック名,10, コールバック関数) という形で記述する．これによって embeddedRTPS に紐づけられたため，サブスクライブ機能を利用できる．定義された subscriber がどのようにサブスクライブされるのか以下にフローを示す．

- ノードの Participant の reader が UDP パケットを受信する．
- 初期化タスク時に Participant 情報として登録されていた embeddedRTPS 内のコールバック関数が実行され，RTPS に則った形式に変換する
- mROS 2 通信ライブラリ内で RTPS パケットのデシリアライズを行い，メッセージに変換する．
- タスク間の通信によって登録されていたコールバック関数およびサブスクライブされたメッセージのオブジェクトポインタを渡す．
- メッセージのオブジェクトのポインタを引数としてコールバック関数を実行する．

メッセージの Pub/Sub 通信は非同期的に行われるため，reader task がサブスクライブを待ち受けるには CPU 使用権を専有する必要があり，メッセージサブスクライブに対するコールバック関数を実行する必要がある．その為，この処理時間が長くなる場合に，次のメッセージのサブスクライブを待てないという問題が発生する．UDP パケットの到着を高い優先度で定期的に監視する役目を reader task が行うことで到着時にメッセージのサブスクラ

イブ処理を行うようにしている．

2.2.7 mROS 2-POSIX のメッセージ生成

ROS 2 では様々なメッセージ型を定義して通信することができるが，mROS 2-POSIX もオリジナルのメッセージファイルを用意してそのメッセージ型で通信することができる．そもそも mROS 2 にはデフォルトで sensor_msg/msg/image.hpp,std_msgs/msg/bool.hpp, byte.hpp，char.hpp，float32.hpp，float64.hpp，header.hpp，int16.hpp，int32.hpp，int64.hpp，int8.hpp，string.hpp，uint16.hpp，uint32.hpp，uint64.hpp，uint8.hpp というメッセージ型が用意されている．これらのメッセージ型は，mROS 2-POSIX でも利用することができるが，ほかのメッセージ型も mros2/mros2_header_generator の.py ファイルを用いて生成することができる．生成したいメッセージファイルは custom_msgs/の中に入れないと生成できない．また階層構造はかならず○○_msgs/msg/○○.msg という形にしないと生成できない．またメッセージファイルのなかにコメント文があるとコメントも一緒に取り込んで split するので注意する必要がある．やり方は mros2-posix/workspace に移動して python3 ../mros2/mros2_header_generator.py custom_msgs/msgs/xxx.msg という形で記述する．いろいろ制限が多いものの，やり方さえ覚えてしまえば，既存の ROS 2 と同様にメッセージファイルを生成することができる．

第 3 章

実装

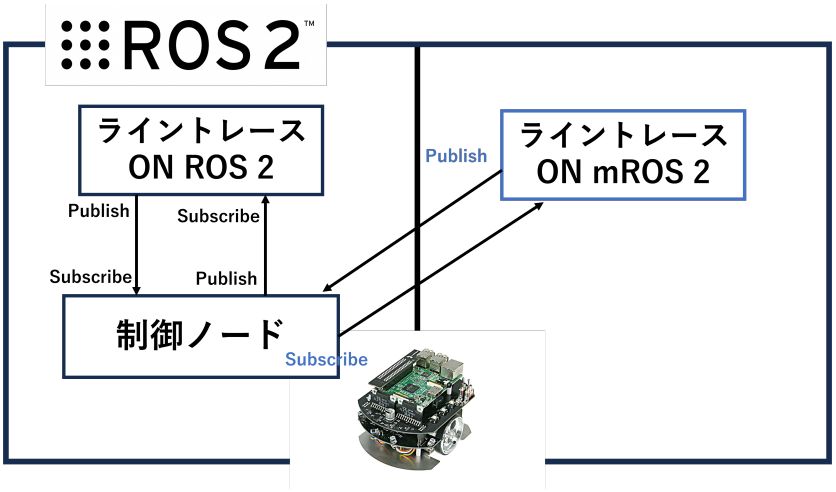

図 3.1: Raspberry Pi Mouse を制御するアプリケーションの構成図

本章では, mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって, mROS 2-POSIX と ROS 2 に実装するアプリケーションの概要について説明する.

3.1 実装アプリケーションの概要

本研究では ROS 2 で動作するライントレースノードを mROS 2-POSIX に移植し, mROS 2-POSIX と ROS 2 の性能を比較評価する. 比較評価のためのアプリケーションの構成図を図 4.1 に示す. ライントレースノードは, Raspberry Pi Mouse のセンサから取得した値をもとに, ラズパイマウスをライントレースさせるノードである. どのように動作するのかというと, まず, Switch2 を押すと, ライトセンサの値を床の値だと判断し保存する. Switch1 を押すと, ライトセンサの値を取得し, トレースするラインのセンサの値を保存する. Switch0 を押すと, 取得したライトセンサの値をもとに, ライトセンサの値がすべて床

の値になるまでモーターを回転させ前進させる．この動作を行うノードを mROS 2-POSIX と ROS 2 で実装し，比較評価を行う．

3.2 実装アプリケーションの詳細

本節では，実装アプリケーションの詳細について説明する．ROS 2，mROS 2-POSIX で実装するノードはサブスクライブするトピックが 2 つ，パブリッシュするトピックが 3 つある．サブスクライブするトピックは/light_sensors と/switchs である．このトピックはラズパイマウス制御ノードが現在のライトセンサの値とスイッチの値をパブリッシュするトピックである．light_sensors はライトセンサの値は int 型で配列になっており，switchs はスイッチの値は bool 型で配列になっている．それぞれオリジナルのメッセージ型で定義去れており，mROS 2-POSIX では ROS 2 のメッセージ型を変換するために，mros2_generator_msg を用いて生成した．これらの値はライントレース制御に使われており，制御ノードからのパブリッシュがあってライントレースは動作する．次に，パブリッシュするトピックは/cmd_vel，/buzzer および/leds である．/cmd_vel はラズパイマウスのモーターを制御するためのトピックであり，/buzzer はラズパイマウスのブザーを制御するためのトピックである．そして，/leds はラズパイマウスの LED を制御するためのトピックである．/cmd_vel は車輪などによく利用される geometory_msgs/Twist というメッセージ型で定義されている．角速度の angular と線速度の linear があり，それぞれ x,y,z の値がある．/buzzer は Int16 型のメッセージ型で定義されており，パブリッシュされた値の大きさでラズパイマウスのブザーの音の大きさが変わる．/leds は raspimouse_msgs/Leds というラズパイマウスオリジナルのメッセージ型でパブリッシュされる．bool によって 4 つある LED の点灯，消灯を制御する．これらのトピックはライントレース制御ノードがサブスクライブするトピックであり，このトピックを実装ノードによってパブリッシュされることで，ライントレース制御が行われる．ROS 2 の実装は，既存のライントレースノードを使用した．そのため，git clone コマンドで既存のライントレースノードをダウンロードし，ビルドを行った．mROS 2-POSIX の実装は，ROS 2 の実装を mROS 2-POSIX に移植した．移植にあたって，パブリッシュやサブスクライブなどのＡＰＩを mROS 2-POSIX のＡＰＩに変更した．

3.3 実装アプリケーションの課題と解決策

大きな課題となったのは通信である．ROS 2 の実装ではモーター電源の on/off にサービス通信を使っており，mROS 2-POSIX では第 2 章で述べた通り，Pub/Sub 通信でしか通信することができない．そのため，モーターを起動する際は，端末で ros2 Service call を使うことで mROS 2-POSIX のパブリッシュからモーターが動作するようにしなければならなった．また，第 3 章の Raspimouse の節で述べたように，ROS 2 と mROS 2-POSIX

の Pub/Sub 通信がうまくいかないという問題がある．現在の実装ではそれぞれのパブリッシャー，サブスクライバーの QoS 設定を RELIABLE に設定し通信させている．ラズパイマウスからのサブスクリプションは常に成功しているが, mROS 2-POSIX からのパブリッシュは部分的に成功している．/leds トピックに対して mROS 2-POSIX からのパブリッシュは成功している．しかし，/cmd_vel と/buzzer に対して mROS 2-POSIX からのパブリッシュは失敗していると考えている．デバックツールである ros2 topic info を用いて/cmd_vel と/buzzer のトピックを確認したところ，mROS 2-POSIX からのパブリッシャーは存在しているのにも関わらず，トピックにデータを送信できていなかった．これは様々な原因が考えられるが，現在は mROS 2-POSIX の実装に問題があると考えている．理由は，ROS 2 制御ノードの端末に [RTPS_READER_HISTORY Error] Change payload size of 'X' bytes is larger than the history payload size of '11' bytes and cannot be resized. -¿ Function can_change_be_added_nts というエラーが出ていることから，mROS 2-POSIX の payload が大きすぎるために，パブリッシュが失敗していると考えている．この問題を解決するためにためしたことは以下の通りだ．

- ros2 を再構築する
- パブリッシャーとサブスクライバーの QoS 設定 RELIABLE にする
- 使用するメッセージ型を再生成する
- FastRTPS の HistoryMemoryPolicy を PREALLOCATED_WITH_REALLOC_MEMORY_MODE に変更する
- FastRTPS の payload_max_size500 に変更する
- FastRTPS から CycloneRTPS に変更する

ros2 を再構築したのは，同じエラーに対して，ros2 を再構築することで FIX したという報告があったから試したが，変化はなかった．mROS 2-POSIX のパブリッシューとサブスクライバーの QoS 設定を RELEABLE にすることで, mROS 2-POSIX のサブスクライバーがパブリッシャーを認識したが，mROS 2-POSIX からのパブリッシュに関して変化はなかった．mROS 2-POSIX の更新により，既存のメッセージ型を再構築しないと使えない場合があるので再構築を試したが，変化はなかった．payload の問題に対処するために FastRTPS の設定を変更するよう.xml ファイルを作り設定したが，設定ファイルを読み込めず，反映できなかった．この原因はわかっていない．CycloneRTPS は ROS 2 のデフォルトの RTPS である．しかし mROS 2-POSIX との互換性はないようで，CyclonRTPS にした際に，mROS 2-POSIX は ROS 2 と通信できなくなり，ros2 topic list 等のコマンドにも表示されなかった．これは mROS 2-POSIX の RTPS である embeddedRTPS が CycloneRTPS と互換性がないためだと考えられる．上記の解決策を試したが，いずれも解決には至らなかった．現在，解決方法として考えているのは以下の方法である．

- mROS 2-POSIX の payload の上限を決める
- ROS 2 制御ノード側のサブスクリプションの QoS 設定の Durability を TRANSIENT_LOCAL にする
- rqt console や rqt config を PC 側から立ち上げ，トピックとデータの流れを確認しながらデバックする

mROS 2-POSIX の payload の上限を決めることで，payload が大きすぎるというエラーを回避することができると考えている．payload の上限を決めることができる箇所を調査中である．また，ROS 2 制御ノード側のサブスクリプションの QoS 設定の Durability を TRANSIENT_LOCLA にすることでパブリッシュが成功しない現状を回避できる可能性があると考える．いままでのデバックはコンソールからのデバックであったため，PC の GUI を使ってデバックすることで，より詳細に原因を突き止めることができると考えている．上記の解決策を試しながら，今後とも問題の解決に向けて取り組んでいく．

第 4 章

評価

mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，mROS 2-POSIX に実装できるアプリケーションと同様のアプリケーションを ROS 2 に実装し，比較評価する．本章では，比較評価に用いるアプリケーションの選定基準と，比較評価に用いるアプリケーションの詳細について述べる．

4.1 アプリケーションの選定基準

本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，以下の基準を設けた．

- Pub-Sub 通信のみを使用したアプリケーションであること
- 組み込みデバイス上で動作できるアプリケーションであること
- 実行時間の計測が容易であること

4.1.1 Pub-Sub 通信のみを使用したアプリケーションであること

mROS 2-POSIX は，embeddedRTPS を利用して RTPS を実装している都合上，ROS 2 の Pub/Sub 通信のみの実装となっている．そのため，評価に用いるアプリケーションは Pub/Sub 通信のみを使用したアプリケーションである必要があり，この条件を満たすことで，mROS 2-POSIX と ROS 2 の通信性能を比較評価できる．

4.1.2 組み込みデバイス上で動作できるアプリケーションであること

先行研究での動的配置機構はオーバーヘッドの増加が問題とされ，mROS 2-POSIX を用いたアプローチも提案されているが，性能評価はネットワークスループットのマイクロベンチマークに留まり，実アプリケーションにおける有用性が十分に評価されていない．そのた

め，評価アプリケーションの実装条件として，動的配置機構が実現される組み込みデバイス上で動作するアプリケーションでなくてはならない．

4.1.3 実行時間の計測が容易であること

今まで検討した手法では，実装に際しての課題もあるが，実装後の実験の有用性について疑問があった．実行時間の計測が容易なアプリケーションであることで，mROS 2-POSIX と ROS 2 それぞれの優位な点を明らかにすることができると考える．

4.2 評価手法

以上のアプリケーションの選定基準を満たしているのが第 4 章の実装である．ROS 2 と mROS 2-POSIX それぞれの環境でラウンドトリップタイムを計測し，比較評価を行う．また，実行時間だけでなく，アプリケーション稼働時のメモリにも着目したい．組込みデバイス上ではメモリの容量が限られているため，メモリの使用量が少ない方が望ましい．以上を踏まえてアプリケーションの評価実験を行う．

第 5 章

関連研究

この章では本研究の関連研究について述べる．

5.1 ロボットソフトウェア軽量実行環境 mROS 2 の POSIX 対応に向けた実装および評価

高瀬らは，ROS 2 の軽量実行環境である mROS 2 の POSIX 対応に向けた実装および評価を行った．評価の方法として 2 つのデバイスを用意し，別のデバイス上のノードから std_msgs/String.msg 型のメッセージをパブリッシュする．このノードを pub とする．その後，他方のデバイス上のノードを使って pub から送られてきたメッセージをサブスクライブし，サブスクライブしたメッセージをそのまま pub ノードと同じデバイスで動作する別のノード（sub ノード）にパブリッシュしてサブスクライブする．その間のラウンドトリップタイム（RTT）を計測することで mROS 2-POSIX の評価を ROS 2 と比較して行っている．この評価実験では，mROS 2-POSIX と ROS 2 の RTT の比較を行っているが，mROS 2-POSIX と ROS 2 のメモリ使用量に関しては比較されていない．

5.2 クラウド連携を対象としたアーキテクチャ中立な ROS ランタイムの検討

柿本らは，クラウド連携を対象としたアーキテクチャ中立な ROS ランタイムを実装し，実装後の評価実験を行った．実装として，組込みデバイス向け ROS 2 ランタイム実装である mROS 2 を WebAssembly ランタイム上で動作させることで，アーキテクチャ中立な ROS ノード実行状態を実現している．評価実験として，実装した mROS 2 on Wasm を使って，メモリ使用量と RTT の計測を行い，実装を評価した．実験として計算処理を行うの 0 度をクラウドを想定したデバイス上で動作させ，その際にかかる時間を計測し，ロボットを想定したデバイス上で動作させた際にかかる時間を比較している．この評価実験で mROS 2 と

ROS ランタイムの間に処理時間の差が認められ，mROS 2 の軽量化が証明された．

5.3 mROS 2: 組込みデバイス向けの ROS 2 ノード軽量実行環境

高瀬らは組込みデバイス向けの高効率な ROS 2 通信方式およびメモリ軽量な実行環境を確立するために，提案として軽量ランタイムである mROS 2 を設計，実装，評価した研究である．mROS 2 を評価するにあたって，ROS と micro-ROS を用いている．micro-ROS は ROS 2 の組込みデバイス向けの軽量実行環境である．mROS 2 と micro-ROS の違いは，ROS 2 と通信する際の Agent ノードの有無である．Agent ノードを立ち上げなければならない，micro-ROS は通信に遅延が発生する恐れがある．通信性能の評価実験として std_msgs::msg::Int32 のメッセージをエコーバックするアプリケーションを用いている．アプリケーションでの RTT を計測することで通信性能を評価した．結果は，mROS 2 の RTT が一番小さくなり次いで汎用デバイス同士をつないだ ROS 2 環境が早かった．この実験結果から組込みデバイス上で動作した mROS 2 の RTT は汎用デバイス上で動作した ROS よりも高速であることが分かった．

第 6 章

おわりに

本研究は，ROS 2 の軽量実行環境である mROS 2-POSIX の開発を行い，mROS 2-POSIX 上に ROS 2 で実装されているノードの実装を行うことで，mROS 2-POSIX の有用性を検証することを目的としている．有用性の検証過程で，Sphero sprk+ と Wii Fit Board を用いたロボットの動作を制御するアプリケーションの実装を検討した．また，Wii Fit Board と Raspimouse を用いたロボットの動作を制御するアプリケーションの実装を検討した．上記の検討は有用性のある評価実験が行えないと判断し，一部実装のみで終了した．そのため，本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価するために，Raspimouse で動作するライントレースノードを mROS 2-POSIX に移植し，比較評価試みた．実装の結果，mROS 2-POSIX と ROS 2 の間で通信が成功せず，比較評価ができなかった．しかし，mROS 2-POSIX と ROS 2 の間で通信が成功した場合の評価実験の計画を立て，実装を行った．今後の課題として，現状の問題の解決策を一つ一つ試していき，通信の不具合を解消していきたい．

謝辞

この度の研究を通じて、多大なるご指導とご支援を賜りました松原　克弥先生に心からの感謝の意を表します．また、日々の研究生活において、絶えず励ましと支えを提供してくださった研究室の仲間たち、先輩方にも深く感謝申し上げます．

参考文献

[1] ROSWiki：ROS/Introduction, urlhttp://wiki.ros.org/ROS/Introduction.

[2] ロボット政策研究会: ロボット政策研究会 報告書 RT 革命が日本を飛躍させる ,https://warp.da.ndl.go.jp/info:ndljp/pid/286890/www.meti.go.jp/press/20060516002/robot-houkokusho-set.pdf(2006).

[3] Kehoe et al. explored cloud-based robot grasping utilizing the Google object recognition engine, presenting their findings in the 2013 IEEE International Conference on Robotics and Automation, pages 4263-4270.

[4] 菅文人, 松原克弥: クラウドロボティクスにおける異種デバイス間タスクマイグレーション機構の検討，研究報告組込みシステム (EMB)，Vol. 2022， No. 36， pp. 1-7(2022).

[5] 柿本翔大，松原克弥: クラウド連携を対象としたアーキテクチャ中立な ROS ランタイムの実現，情報処理学会研究報告， Vol. 2023-EMB-62，No. 51 ，pp. 1-7(2023).

[6] 高瀬英希，田中晴亮，細合晋太郎: ロボットソフトウェア軽量実行環境 mROS 2 の POSIX 対応に向けた実装および評価，日本ロボット学会誌，Vol. 2023-EMB-41，No. 8，pp. 724-727(2023).

[7] Object Management Group: About the DDS Interoper-ability Wire Protocol Version 2.5 (online), https://www.omg.org/spec/DDSI-RTPS/2.5 (2024.01.25).

[8] 高瀬英希:ROS (Robot Operating System) の紹介と IoT/IOT 分野への展開,RICC-PIoT workshop 2022.

[9] Fastdds Simple Discovery Settings5.3.2, https://fast-dds.docs.eprosima.com/en/latest/fastdds/discovery/simple.html

[10] Fastrtps ros index https://index.ros.org/r/fastrtps/

[11] RTI Connext DDS, https://www.rti.com/en/

[12] Micro XRCE-DDS, https://micro-xrce-dds.docs.eprosima.com/en/latest/index.html

[13] Eclipse Cyclone DDS, https://cyclonedds.io/

[14] Gurum DDS, https://gurum.cc/index_eng

[15] クライアント・サーバモデル, https://www.ibm.com/docs/ja/txseries/8.2?topic=computing-clientserver-model

[16] “micro-ROS — ROS 2 for microcontrollers,” https://micro.ros.org/

[17]

[18] ”embeddedRTPS”, https://github.com/embedded-software-laboratory/embeddedRTPS

[19] ”lwIP”, https://savannah.nongnu.org/projects/lwip/

[20] ”TOPPERS/ASP3 kernel”, https://www.toppers.jp/asp3-kernel.html

[21] ”POSIX”, https://ibm.com/docs/ja/zos/2.5.0?topic=ulero-posix

[22] ”CMSIS-POSIX”, https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html