並列型行動アーキテクチャを有するロボットのラピッドプロトタイピングを実現するミドルウェア開発

松永 大介，山本 智哉，藤井 春香，小島 岳史

人と同じ空間で柔軟に動き、非定型な作業を自動化するロボットの実現には、認識、計画、制御といった異なる特性を持つ機能を並列処理し、即応性を高める必要がある。

この要件を満たすアーキテクチャとして、並列型行動アーキテクチャが存在する。このアーキテクチャでは認識・計画などの高次情報処理を行う上位処理とロボットの動作機能を実現する下位処理が並列処理される。さらに、上位処理が下位処理に対して必要に応じて作用することにより、ロボットアプリケーションの即応性が担保される。

筆者らはこのようなアーキテクチャを持つロボットのラピッドプロトタイピングを実現するため、ロボット技術開発用ミドルウェアを開発した。本ミドルウェアは、リアルタイム処理や並列処理などの実装に専門性が必要な機能を提供し、開発を効率化する。また、実際に本ミドルウェアを用いて並列型行動アーキテクチャを有する複数のロボットを短期間で開発することができた。

本稿では、本ミドルウェアの仕様と有用性について既存のものと比較を交えて述べる。また、本ミドルウェアとその開発環境の適用事例を用いて、性能検証結果と開発効率に関する考察を述べる。

Development of a Middleware for Rapid Prototyping of Robots Based on Parallel Action Architecture

MATSUNAGA Daisuke, YAMAMOTO Tomoya, FUJII Haruka and KOJIMA Takeshi

To realize robots that can move flexibly in human environments and automate non-standard tasks, it is essential to parallel process functions like recognition, planning, and control to enhance responsiveness. The parallel behavior architecture addresses this need by executing high-level information processing in upper layers and motion capabilities in lower layers simultaneously. This structure ensures that upper processing can influence lower processing as needed, maintaining the responsiveness of the robot application.

The authors developed middleware for robotic technology to enable rapid prototyping of robots with such architectures. This middleware provides functionalities that require expertise in implementing real-time processing and parallel processing, thereby streamlining the development process. Additionally, using this middleware, multiple robots with a parallel behavior architecture were developed in a short period of time.

This paper presents the specifications and advantages of the middleware, comparing it with existing solutions. It also presents performance verification results and considerations regarding development efficiency using case studies of the middleware and its development environment.

1. まえがき

近年、少子化や人手不足などの問題を背景に、ロボットによる生産性向上のニーズが高まっている。これらを受けオムロンでは、主に FA 分野でライン作業などの定型な作業の自動化に取り組んできた。今後はより広範な分野・作業に対応するため、人と同じ空間で柔軟に動き、非定型な作業を自動化する様々なロボットの実現を目指している[1)]。このようなロボットの開発では、開発初期の要件定

Contact : *MATSUNAGA Daisuke* daisuke.matsunaga@omron.com

義では想定が難しい条件や作業の変更が多く発生する。そのため、ラピッドプロトタイピングによりユーザからのフィードバックを受けて改良を繰り返す開発アプローチが有用である。

ロボットが人と同じ空間で作業するためには、未知の環境や環境の動的な変化に柔軟に対応しながら、確実に作業を遂行することが必要である。そのためには、認識、計画、制御といったロボット内部処理を高サイクルで連携させる必要がある。

ロボットの内部処理は認識・計画といった高次情報を取り扱う上位処理と、モータ制御のようなロボットの動作機能を実現する下位処理に分類できる。ここで、下位処理は上位処理の結果を入力として使用するため上位処理に対して依存関係をもつ。さらに、計画処理や認識処理といった上位処理間にも依存関係が存在する。これらの関係を直接モデル化した従来型のシーケンシャルな実行方式では、上位処理が下位処理を律速して周期性を実現できず、ロボットアプリケーションの即応性が低下する問題があった。Brooks の包摂アーキテクチャ[2] や Yamamoto と Sugihara の積層変調アーキテクチャ[3] は、この問題に対する一つの方向性を示した（図 1）。これらの並列型行動アーキテクチャは、上位と下位の処理を独立かつ並列に実行し、上位処理が必要に応じて下位処理に対して作用する。そのため、下位処理による動作の実行を妨げず、システムの即応性を担保できる。これらのアーキテクチャを実現するには、複数の制御器を柔軟に実装可能かつマルチスレッドスケジューリング可能なソフトウェアが必要となる。

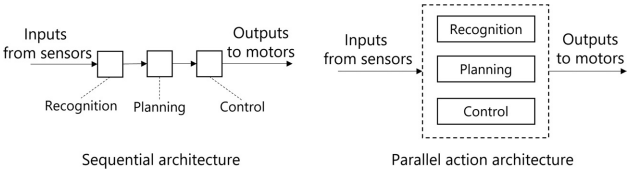

図 1　従来型と並列型行動アーキテクチャによる実行方式

一方、ロボットのソフトウェア開発の側面に目を向けると、専門的知見を要求する機能の実装が開発のボトルネックとなることが多い。例えば、モータ制御をはじめとする周期処理の実装にはリアルタイム性能の考慮が要求される。また、並列型行動アーキテクチャの場合、データ競合やデッドロックを避けたスレッドセーフな実装や、並列処理に起因する不具合の解析や実装の修正が必要となり、これらにも専門的知見が必要となる。さらに、ロボットシステムとしての改良を繰り返すうちにハードウェアや OS（Operation System）などの実行環境が変更されることによる手戻りが発生することもある。

ロボット開発効率化を目的とするソフトウェアのコンポーネント化やオープン化が進んでいるが、並列処理のスケジューリングに主眼をおいたミドルウェアはなく、並列型行動アーキテクチャを持つロボットの効率的な開発は難しい。さらに、一般にソフトウェアの開発効率化を図るには、保守性や移植性などのソフトウェア品質を高めることも要求される。

以上の経緯から、筆者らは上記アーキテクチャの実現とロボット開発効率化を両立するロボット技術開発用ミドルウェアを開発した。本ミドルウェアは、並列型行動アーキテクチャを有するロボットのソフトウェア開発において課題となるリアルタイム性の実現、データ競合の防止、実行環境の変更を容易にする。

リアルタイム性の実現には、実行周期や優先度、CPU コアの割り当てをはじめとする種々の設定が必要であり、OS や実行環境に依存した組み込み開発特有の専門性が要求される。また、ラピッドプロトタイピングで開発するには、これらの設定が容易に変更可能であることが望ましい。そこで、本ミドルウェアは、これらを容易に設定できるマルチスレッドスケジューリング機能を提供した。この機能はリアルタイム処理の概念をモデル化し、ミドルウェアのインターフェースで各種設定を実行可能にする。これにより、OS や実行環境に関する知識がなくてもリアルタイム処理の実装や設定変更が可能となり、開発中の仕様変更に柔軟に対応できるようにした。

データ競合の防止の実現には、スレッド間の指令送受信機能および出版 - 購読型モデルを基本としたデータ共有機能をミドルウェアの機能として提供した。これらの機能にスレッド間アクセス時のデータ競合の対策を集約することにより、開発者はデータ競合を考慮することなく、スレッド間アクセスを実現できる。

実行環境の変更には、開発ターゲットに依存しない OS 環境でのクロス開発を可能にすることで対応した。このアプローチにより、ターゲット環境とは異なる環境での実装やデバッグに好適なハードウェア・OS 環境上での不具合解析が可能となる。この実現は、ミドルウェアに OS の抽象化層を設けることにより達成している。

本稿では、従来手法との比較を交えて、その内容と本ミドルウェアの有用性について述べる。また、本ミドルウェアとその開発環境の適用事例を用いて、性能検証結果と開発効率に関する考察を述べる。

2. 既存のロボットミドルウェア

広く使用されている同種のミドルウェアのうち代表的な ROS と RT ミドルウェアについて紹介し、各ミドルウェアの課題について述べる。

2.1 ROS

ROS（Robot Operating System）[4] とは、Willow Garage 社が開発したロボットシステム開発用のオープンソースミドルウェア、またそのミドルウェアを使った開発環境ツール

群及び開発成果のエコシステムである。現在は非営利団体 Open Robotics が維持・管理している。様々な機能をノードと呼ばれる単位で実装し、ノード同士が P2P で通信することでロボットシステムを実現することができる。この疎結合によるシステム構築が可能な点と、開発環境ツール群が充実していること、エコシステムによるモジュールの公開・共有の仕組みがあること、ライセンスが条件を満たせばソースコードを公開する必要がない BSD であることにより、現在研究開発の場で最も使われている。

また、現在 ROS2[5] と呼ばれる次世代バージョンの開発が進んでいる。これは、ROS 開発当初に設定された要件定義が陳腐化してきたため、要件を見直して新たに開発されたものである。旧版との主な差異としては、複数台のロボット制御・組み込みシステム・リアルタイム制御への対応が挙げられる。

2.2 RT ミドルウェア

RT ミドルウェア（RT-Middleware: RTM）[6] とは、産業技術総合研究所（産総研）などによって開発された、ロボットやロボット技術（Robotics Technologies: RT）を用いたシステムを構築するためのソフトウェアプラットフォームである。

RTM では各機能要素をコンポーネント化したソフトウェアモジュールを組み合わせてロボットシステムを構築するようになっている。このソフトウェアモジュールは RT コンポーネント（RT-Component: RTC）と呼ばれ、そのインターフェース仕様は国際標準化団体 OMG（Object Management Group）に公式標準仕様として規格化されている[7]。

また、RTM は（独）新エネルギー・産業技術総合開発機構（NEDO）の次世代ロボット知能化技術開発プロジェクトにおける開発基盤として採用されている。

OpenRTM-aist[8] は、産総研が実装・配布している RT ミドルウェアの実装を含むソフトウェアプラットフォームで、RTC の実行を管理するためのミドルウェア、RTC を実装するためのフレームワーク、ソフトウェア開発環境などで構成されている。RTC のフレームワークや管理機能が提供されていることから、ユーザはコアロジックの開発に専念できる。

2.3 既存ミドルウェアの課題

ROS に関しては、ノードと呼ばれる処理単位の独立性が高く、並列型行動アーキテクチャ実現のための処理の並列実行という点では適していると言える。しかし、ノード自体の処理やノード間の通信がリアルタイム処理に対応していないため[9]、即応性の高いロボットの制御を目指す本件に対しては不適である。ROS2 ではリアルタイム制御への対応を謳っているが、通信部分のリアルタイム性の向上がなされるにとどまっているのが現状である[10]。したがって、リアルタイム性の保証ができないため、即応性の高いロボットシステムの実現という筆者らの目的に対して十分ではない。これは、ROS は元々ロボット自体の制御ではなくロボットを構成要素の一つとするロボットシステムの開発を目的としたフレームワークであることが原因であると考えられる。

RT ミドルウェアの実装である OpenRTM-aist に関しては、RTC と呼ばれる処理単位の独立性が ROS 同様に高く、またノードの処理・通信に関してもリアルタイム処理に対応しており、筆者らが目指す即応性の高いロボットの実現という目的に対して好適な設計であると言える。しかし、並列型行動アーキテクチャのような構成を実現するには、標準の OS（Ubuntu）をリアルタイム化するためにカーネルにパッチ適用したうえで、リアルタイムに実行させる全ての RTC に対して以下の作業が必要になる。

- 周期実行部分の前後でリアルタイムプロセス/非リアルタイムへの変換処理を実装
- 実行周期と優先度ごとのコンテキストへの割り当て設定
- データ送受信方式を、周期実行に遅延が発生しないように自身の処理や送受信先の周期に応じて設定

このように実装されたシステムは、RTM が本来想定する、独立したモジュール群によりネットワーク的に構成されたシステムとは異なる。そのため、これらの設定作業は標準のフレームワークと開発環境ではサポートされておらず、ミドルウェアのユーザ自身ですべて対応する必要があり負担である[11]。

3. 提案手法

3.1 本ミドルウェアへの要求事項

本節では、前章で述べた問題の解決を目的とする主要要求事項を示す。また、ラピッドプロトタイピングの実現に必要となるミドルウェア全体の要求事項を品質特性ごとにまとめる。

3.1.1 主要要求事項

1）マルチスレッドスケジューリング

リアルタイム性能要求の高いスレッドに対して明示的に高い優先度やシステムリソースを割り当て可能とするマルチスレッドスケジューリング機能を提供する。これにより、リアルタイム性能要求の高いスレッドに対するそれ以外のスレッドからの影響を低減する。

2）データ競合の防止

スレッド間の指令送受信機能および出版-購読型モデルを基本としたデータ共有機能をミドルウェアの機能として提供する。これらの機能にスレッド間アクセス時のデータ競合の対策を集約し、スレッドセーフかつノンブロッキングでデータ共有を行う仕組みを持たせる。

3）マルチプラットフォーム対応

ミドルウェアに OS の抽象化層を設けることにより、開発ターゲットに依存しない OS 環境でのクロス開発を可能にする。このアプローチにより、ターゲット環境とは異なる実装/デバッグに好適な OS/ハードウェア環境上での不具合解析を可能にする。

なお、主要要求事項 1）のスレッド周期実行処理、主要要求事項 2）のデータ共有の仕様を決めるに伴い、弊社マシンオートメーションコントローラ NJ/NX CPU ユニットの仕様であるタスク、変数同期機能[12] の振る舞いを参考にした。

3.1.2 ミドルウェア全体の要求事項

本ミドルウェアの要求事項を ISO/IEC25010 のソフトウェア品質特性モデル[13] に分類する。本ミドルウェアは、ラピッドプロトタイピングを実現し技術開発の生産性向上を重視しているため、機能適合性、信頼性、セキュリティのような製品に求められる特性は考慮していない。なお、表 1 には開発効率化に必要となる開発補助機能、開発環境に対する要求も含めて記載している。

表 1　本ミドルウェアの要求事項

品質特性	要求
性能 効率性	リアルタイム処理、非リアルタイム処理が混在したマルチスレッドスケジューリングができること
保守性	最低限の変更で様々なキネマティクスのロボットに対応できること
互換性	外部システムと接続ができること
移植性	マルチプラットフォーム（複数の OS 環境）で動作可能であること
使用性	高級プログラミング言語を用いた開発ができること
	認識、計画、制御などの処理をすべて同一システムに作り込めること
その他	開発環境に CI（Continuous Integration）/CD（Continuous Delivery）をサポートすること

3.2 機能説明

本節では、前節で述べた要求事項を満たす本ミドルウェアの主要機能と、ラピッドプロトタイピング実現に必要なソフトウェア要件をまとめた補助機能について説明する。なお、主要機能 1）、2）は性能効率性、3）は移植性に関する品質の要求事項を満たす。

3.2.1 主要機能

1）マルチスレッドスケジューリング機能

本機能は、各処理の実行周期・優先度とスケジューリング方式に基づき、処理の実行順序をスケジューリングする。周期・優先度・スケジューリング方式などのパラメータは所望のシステムに応じて開発者が任意に設定できる。これは従来手法と異なり、実行スケジューリングのフレキシビリティに重点をおいた機能仕様である。

本機能により、リアルタイム性が要求されるモータ制御部のような処理と、それ以外の処理を並列実行するシステムを構築できる。ここでのリアルタイム性とは、一定周期で期待した時間に遅延なく処理が行われることであり、ロボット制御の重要な処理に必要な要件である。例えば、モータ制御部で遅延が発生するとモータの指令値に対する追従が遅れ、ロボットの位置精度や軌跡追従性能が低下する。

図 2 にマルチスレッドスケジューリング機能の概要図を示す。指定のスケジューリング方式に基づき定周期スレッド（図中 Periodic thread）と非定周期スレッド（図中 Non-periodic thread）が生成される。また、並列実行される各実行処理（図中 proc と表記）は、開発者指定に基づいてスレッドに割り付けされる。

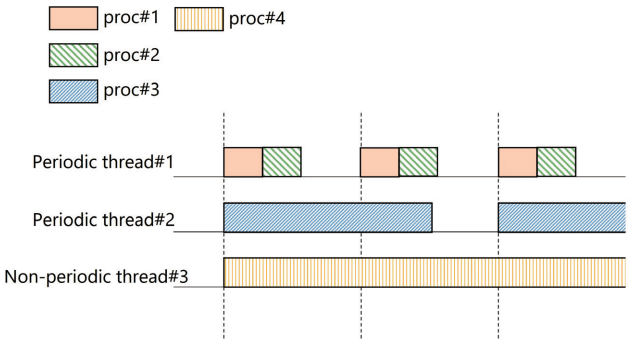

図 2　マルチスレッドスケジューリングの概要

2）データ競合を防止する機能群

下記の 2 つの機能によりスレッド間のデータ競合を防止する。

① スレッド間の指令送受信機能

スレッドまたは、別プロセスからミドルウェアの内部モジュールへのアクセスを可能にするため、汎用インター

フェースによるアクセス（コマンド機能）を実現する。ミドルウェアがコマンド受信スレッドを常駐させ、コマンド受信時に各モジュールのコマンドの実行スレッドを起動することで、コマンドの実行順序の保証とスレッド実行を確実に行えるようにする。また、コマンドの送信方式を同期/非同期の 2 種類を用いることで、ロボット動作のような順次実行が必要なコマンドと、ロボット内部状態表示のような順序性が不要なコマンドの両立を可能にする。

- 同期型：コマンド送信後、呼び出し先がコマンドを受け付けて結果を受信するまで呼び出し元の処理がブロックされる。呼び出し元の処理とコマンドの実行を同期したい場合に使用する。実行時間が長いコマンドには向かない。また、処理がブロッキングされるため、定周期スレッド内での同期コマンド送信は禁止とする。
- 非同期型：コマンド送信後、呼び出し先がコマンドを受け付けた後は呼び出し元の処理はノンブロッキングとなる。結果は指定したコマンド送信時に登録した関数にコールバックされる。実行時間が長いコマンドや定周期スレッド内での使用を想定している。

また、ミドルウェアの内部モジュールが定周期スレッド上でコマンドを受信するため、スレッドセーフキューを用いてコマンドを一次受けする仕組みをミドルウェアに搭載する。これにより、定周期スレッドのクリティカルセクションを最小化し、リアルタイム性能への影響を抑える。

② データ共有機能

スレッド間でデータ一貫性を保持しつつ、データ変更タイミングの保証と他スレッドからのアクセス保護を行う。各スレッドでデータの同時性を持たせるため、ROS や RT ミドルウェアでも使用されている出版-購読型モデルを使用する（図 3）。具体的には各スレッドで使用するデータに対し、公開（Publish）するのか、他スレッドのデータを購読（Subscribe）するのかを定義し、スレッド実行後に Publish、スレッド開始前に Subscribe している。それにより、各スレッドの実行前にはデータが更新され、各スレッド実行中は Subscribe しているデータは同一であることが保証できる。

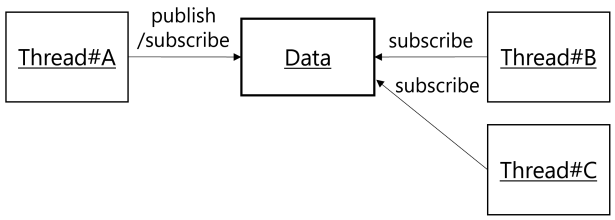

図 3　データ共有の概念図

3）マルチプラットフォーム対応

OS（Operating system）やハードウェアとミドルウェア間の依存を減らす。一般にロボットの技術開発着手時には OS／ハードウェア環境が定まっていないことが多い。また、開発中にこれらの環境変更があることも多い。そのため、実装工程では開発ターゲットに依存せず、実装に好適な OS／ハードウェア環境で開発し、実機検証工程ではターゲットとする OS／ハードウェア環境へ柔軟に変更可能であることが望ましい。そこで、Operating System Abstraction Layer（OSAL）層を設け OS 固有の処理を抽象化することで、柔軟な OS の変更を可能とする。また、ハードウェアも同様に抽象化層を設けることで、H/W なしでも検証を可能にする。

3.2.2 補助機能

本節では、補助機能を本ミドルウェアへの要求品質特性ごとに説明する。

1）保守性

図 4 にソフトウェアアーキテクチャ図を示す。ユーザインターフェースやデバッガに相当する Application 部、今回提案するミドルウェアである Robot Framework 部、OS 部、Hardware 部で構成する。Robot Framework には、ロボット固有の機能を実装した Function Modules、スケジューリングや通信、リソース管理などの基本機能をまとめる Runtime、OSAL を含む。認識（Recognition）・計画（Planning）・制御（Control）などのロボット固有の機能は Function Modules 内で個別にモジュール化・ライブラリ化し、プラグイン可能な構造とする。これにより、所望のシステムに対し最小のモジュール構成で動作可能な構造とし、保守性の向上と計算リソースの削減を実現する。

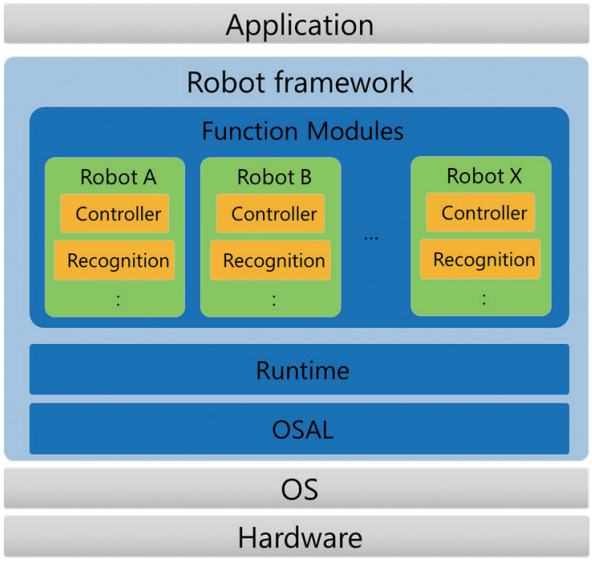

図 4　ソフトウェアアーキテクチャ図

2）互換性

ラピッドプロトタイピングを促進するため、技術開発対象外となる機能は外部システムを使用することにより実装工数を削減する。そこで、3rd-party ソフトウェアといった外部システムが提供する既存機能を活用するため、外部システムとの通信機能を提供する。本ミドルウェアでは、ROS との通信機能を提供することで、ROS をサポートしている様々なシステムを利用できるようにした。

3）使用性／その他

クロス開発環境および CI/CD 環境をサポートする。ロボット技術開発でラピッドプロトタイピングを実現するためには、アルゴリズム開発と実機検証のサイクルをハイサイクルで実施することが求められる。そこで、アルゴリズム開発工程向けに、高級言語が使用可能な Windows 環境やオープンソースで研究に好適な Ubuntu 環境を使用した開発環境をサポートする。開発者は本環境で実装とシミュレーションを行い、開発成果は CI 環境により自動テストが行われる。一方、実機検証工程ではターゲット環境用のバイナリを CD 環境により自動生成する。これらの結果、シミュレーションや試作機での事前検証と実ロボットにおける開発者の手間やミスを削減し開発効率向上を図る。

4. 検証結果

本章では、本ミドルウェアとその開発環境の適用事例を用いて、提案するロボットミドルウェアの性能検証結果およびその開発効率に関する考察を述べる。

4.1 検証環境

4.1.1 ロボットシステム

提案するロボットミドルウェアを用いて構築したモバイルマニピュレータシステムを例に示す。図 5 はその外観である。2 自由度の走行台車上に、8 自由度ロボットアームを搭載している。台車とアーム先端にカメラを使い、外界・物体を認識する。

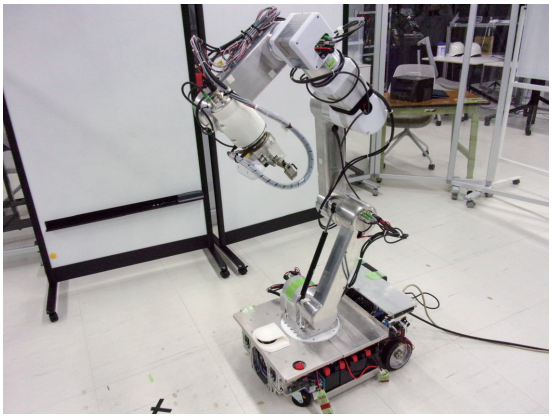

図 5　開発事例：モバイルマニピュレータ外観

本ロボットシステムの内部構成を図 6 に示す。カメラ（Vision Sensors）、外界を認識する PC（Environment Recognition PC）、車輪とアームを動かす制御 PC（Control PC）、モータ（Motors）、外部センサ（Sensors）で構成されている。SLAM（Simultaneous Localization and Mapping）[14] による自己位置推定（Self-Location Estimation）、経路生成（Path Planning）、経路追従（Navigation）、制御（Control）、

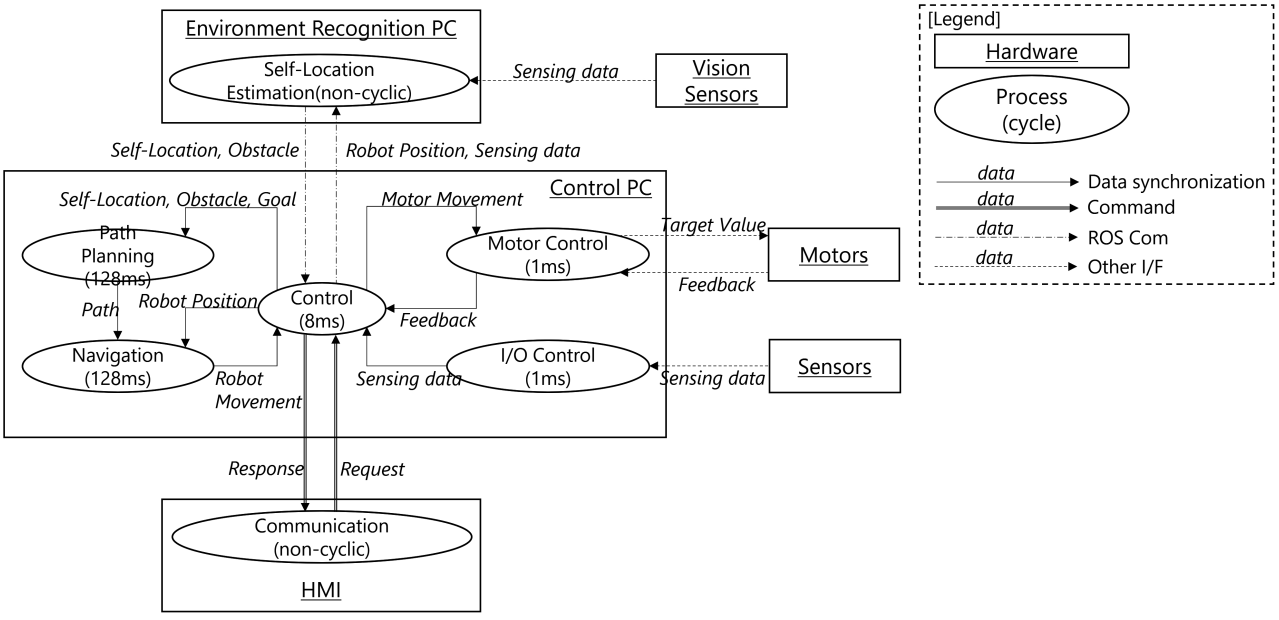

図 6　開発事例：モバイルマニピュレータシステム

I/O 制御（I/O Control）、モータ指令通信処理（Motor Control）が並列に実行され、所望の移動動作を実現する。

自己位置推定以外の処理はリアルタイム OS の Control PC で定周期処理として実行される。このうち、Path Planning と Navigation は数理最適化を含むために実行処理時間変動が大きいので、実行周期の長い 128［ms/cycle］スレッドとして定義される。一方、Control・I/O Control、Motor Control はリアルタイム性の要求が強い処理であるため、それぞれ実行周期の短い 1［ms/cycle］スレッド、8［ms/cycle］スレッドとして定義される。これら定周期のスレッドはミドルウェアのデータ共有機能を介してデータをやり取りしている。スケジューリングは設定ファイルにて、周期、優先度、実行スレッドを指定できるようにしている。

自己位置推定は実行処理時間が実行周期よりも大幅に長くなることや、センサとのインターフェースの必要上非リアルタイム OS の Environment Recognition PC 上で非定周期の処理として実行される。この処理は ROS で実装されており、ミドルウェアの ROS 通信機能を通じて制御 PC と非同期通信を行っている。

また、本ロボットシステムは HMI（Human Machine Interface）からの指令を受けて動作する。HMI からの動作指令はミドルウェアのコマンド機能を使用している。

4.1.2 開発環境

マルチプラットフォーム対応による開発効率向上のため、図 7 のような CI/CD を実行する開発環境を構築した。

開発したアルゴリズムや機能は Function Modules として GitLab 上で管理し、Push したソースコードを自動でビルド・テストを行う。本ミドルウェアで対応している実行環境に合わせた実行バイナリに対し自動テストを行うことで、設計品質を確保している。

CI/CD のビルド、テスト環境は AWS（Amazon Web Service）[15] 上の仮想マシンと開発成果をデプロイする実機を併用して構築している。仮想マシンでは Docker を動かし、CI 環境のコンテナを構築・起動している。

4.2 検証結果・考察

本ミドルウェアに要求される品質特性ごとに結果および考察を記載する。

4.2.1 性能効率性

構築したシステムを実環境で動作させた際の性能効率性に関する考察を以下に示す。測定環境として、OS: VxWorks、CPU: Intel Core i7-1165G7 2.8 GHz、メモリ : 64 GB を搭載した制御 PC を使用した。

1）マルチスレッドスケジューリング

構築したシステムに対し、モバイルマニピュレータ移動中のスレッド実行処理を図 8 に示す。本情報は VxWorks の IDE（Integrated Development Environment）である Wind River Workbench を用いて取得した。

図中では、各スレッドの実行状態を緑色の太線にて表しており、待機状態は波線で表している。図中左にはスレッド名と各スレッド Priority を明記しており、Priority の値が小さいスレッドが優先的に実行される。上から順に、定周期スレッドを生成するための 1 ms 割込み、（Interrupt200）、Motor Control、I/O Control、Control、Non-cyclic Thread1、2、…、5 スレッドが表示されている。Non-cyclic Thread は 3.2.1 2）①に記載のスレッド間の指令送受信機能やデータ共有機能を実行する度にスレッドが生成されるため、同時に数十スレッドが生成される場合がある。本図より実行処理時間変動が大きい複数の非周期スレッド（Non-cyclic Thread1、2、…、5）とリアルタイム性の要求が強い Motor Control、I/O Control、Control スレッドが期待の実行順序、実行優先度で、安定して周期実行されることを確認した。なお、表記の都合上 Planning と Navigation スレッド（128 ms 周期）については記載していない。

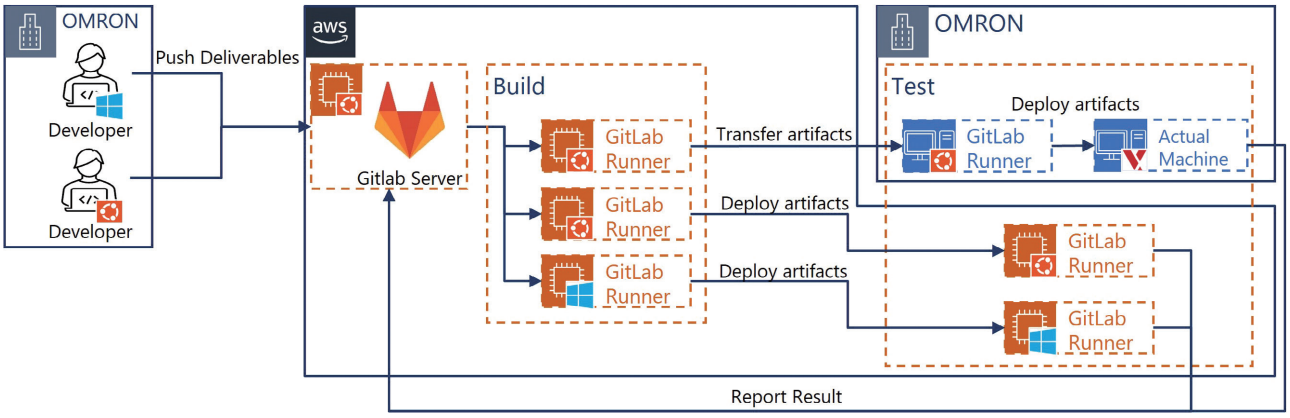

図 7　開発環境

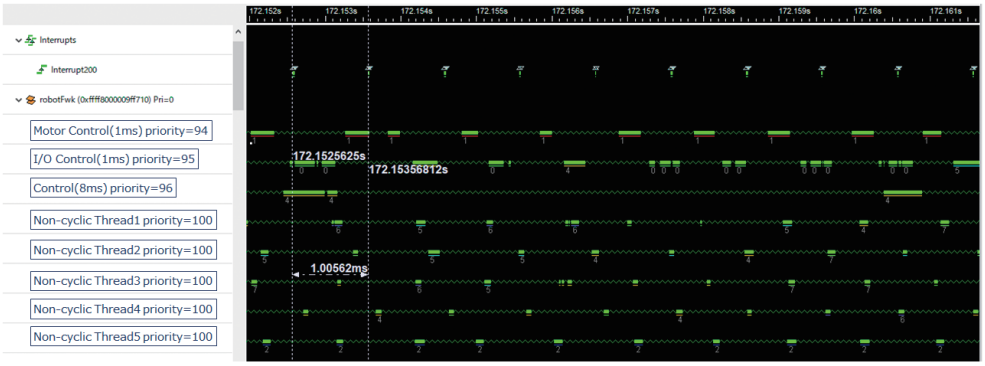

図 8　並列スレッドスケジューリング

2）リアルタイム性

リアルタイム性の要求が強い Motor Control、I/O Control、Control スレッドの実行時間統計情報（最小、最大、平均、標準偏差）を表 2 に示す。

表 2　実行時間統計情報

スレッド名（周期［ms］/Priority）	最小［ms］	最大［ms］	平均［ms］	標準偏差［ms］
Motor Control（1/94）	0.066	0.117	0.074	0.006
I/O Control（1/95）	0.163	0.229	0.172	0.011
Control（8/96）	0.211	0.369	0.276	0.031

本ロボットシステムでは、定周期処理は 1 ms の割込みから指定した間隔でスレッドを実行させることで実現している。割込み処理の統計情報は最小 0.991［ms］、最大 1.009［ms］、平均 1.0［ms］、標準偏差 0.001［ms］と安定した割込みを実現している。表 2 で示す通り、各スレッドは最大値が周期未満となっており、周期実行時間を超えるようなスパイクは発生していない。標準偏差を見ても Motor Control 0.006［ms］、I/O Control 0.011［ms］、Control 0.031［ms］と Priority の高いスレッドの方が少ないばらつきになっている。以上からリアルタイム性が求められるスレッドが安定的に周期実行されていることを確認した。

4.2.2 保守性

本ロボットシステムでは図 4 に示した通り、ロボット固有の機能は、Function Modules 内で実装、ライブラリ化している。一方、共通の機能は本ミドルウェアの必要なライブラリをロードすることで実現している。これにより、ロボット設計者は該当するロボットの Function Module 開発に専念することができた。その結果、モバイルマニピュレータを開発したテーマでは双腕ロボットも同時に開発することができ、本ミドルウェアが様々なロボットの共通フレームワークになりうることを確認した。

4.2.3 互換性

図 6 で示した通り、自己位置推定処理は制御 PC とは別の PC で実行されている。この処理や外部センサの制御に関しては、画像処理のライブラリやセンサのドライバが豊富な ROS を使うことで、短期間で開発することができた。ROS との通信機能を使用することで、この機能をロボットシステムに簡単に取り込むことができ、開発の効率化を実現できることを確認した。

4.2.4 移植性

コマンド機能によって Runtime や Function Modules 内のモジュールそれぞれの機能に対してアクセス可能であるため、開発フェーズに合わせた検証用ツールを使用して開発を効率化することができた。例えば Runtime の開発フェーズではデータ共有機能で public/subscribe されたデータの確認、ロボットの制御機能の開発フェーズではモータ入出力値の監視、計画機能の開発フェーズではロボットシミュレータによるロボット挙動の確認などが可能だった。

また、コマンド機能がマルチクライアント可能な設計になっていることから、その時々で必要なツールを複数同時に使用することができる。これにより、各機能の開発者が自分の作業のために開発した検証ツールを他者が再利用することが容易になり、開発が進んでシステムが複雑化した後に発生した問題の分析も効率的に行うことができた。

これらの検証用ツールは各開発者が開発したものを CI 環境で共有することで別の開発者が改良するという良いサイクルが発生した。その結果、ロボットシミュレータツールは高機能化が進み、仮想空間に任意のロボットを可視化・検証できるようになるまでになった。ロボットの可視化モデルは汎用的なロボット記述フォーマットである URDF（Unified Robot Description Format）[16] で記述できるようになっているため、今後の別の開発テーマにも流用可能であり、ミドルウェアとともに展開予定である。

4.2.5 使用性／その他

1）GitLab での CI（自動ビルド・テスト）

本 CI 環境により、設計者が任意の OS で開発した Function Modules をターゲット環境の実機上での自動テストと実行バイナリの自動生成を可能にした。これにより、シミュレーションで検証を行った後、実機で実際に動かすまでをリードタイムなしでシームレスにできた。結果として、各設計者の検証が、リアルタイム性を考慮した実機動作まで行えるようになり、実ロボットでも使用可能なアルゴリズムや機能の開発を加速できた。

2）AWS を活用したサーバリソースの最適提供

仮想マシンでの運用により、複製、初期化が容易になり、これまで発生していた、サーバやネットワーク負荷の調整、管理負荷が低減した。その結果、開発状況に応じたビルドサーバ・テストサーバの増減や異常時の復旧を 30 分以内で可能にした。

5. むすび

人と同じ空間で柔軟に動き、非定型な作業を自動化するロボットの早期実現には、動的な変化への柔軟性や即応性を備えたロボットをラピッドプロトタイピングで効率的に開発することが課題となる。

本論文では、この課題に対して、並列型行動アーキテクチャを有するロボットをラピッドプロトタイピングで開発可能な技術開発用ミドルウェアを提案した。

具体的には、提案ミドルウェアの特徴と従来ミドルウェアに対する優位性を述べた。本ミドルウェアは、ROS/ROS2 では実現が難しい複数スレッドのリアルタイム性を保証したスケジューリングを実現している。また、データ共有方法やスケジューリングについて個別に考慮する必要がない点で、OpenRTM-aist よりも高い開発効率をユーザに提供できていると言える。

また、本ミドルウェアとそのソフトウェア開発環境の適用事例を用いて、性能検証結果およびその開発効率に関する考察を述べた。

今後は、並列型行動アーキテクチャを搭載するロボットの社会実装を目指し、本ミドルウェアの更なる改善とともに、世の中に革新的なロボットを送り出し続ける所存である。

参考文献

1）オムロン. “中外製薬とオムロン, オムロン サイニックエックスが目指す人が活きる「次世代ラボオートメーション」.” オムロン EDGE&LINK. https://www.omron.com/jp/ja/edge-link/news/688.html（Accessed: Jun. 5, 2024）.

2）R. Brooks, “A robust layered control system for a mobile robot,” *IEEE J. Robot. and Automat.*, vol. 2, no. 1, pp. 14–23, 1986.

3）山本孝信, 杉原知道, “積層変調アーキテクチャに基づく二脚ロボットの SEAN システム,” *ロボティクス・メカトロニクス講演会講演概要集*, pp. 1A1-D05, 2021.

4）Quigley Morgan et al., “ROS: an open-source Robot Operating System,” *ICRA workshop on open source softw.*, vol. 3, no. 3.2, p. 5, 2009.

5）S. Macenski et al., “Robot Operating System 2: Design, architecture, and uses in the wild,” *Sci. robot.*, vol. 7, no. 66, 2022.

6）N. Ando et al., “RT-middleware: distributed component middleware for RT（robot technology）,” in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2005, pp. 3933–3938.

7）*Robotic Technology Component Specification*, formal/08–04–04, 2008.

8）N. Ando et al., “A software platform for component based rt-system development: Openrtm-aist,” in *Int. Conf. Simul., Model., and Program. Auton. Robots*, 2008, pp. 87–98.

9）安藤慶昭, “ロボットミドルウェアの実時間処理・排他処理の比較,” *日本ロボット学会誌*, vol. 34, no. 6, pp. 366–369, 2016.

10）J. Kay. “Introduction to Real-time Systems.” ROS 2 Design. https://design.ros2.org/articles/realtime_background.html（Accessed: Jun. 5, 2024）.

11）菅谷みどり 他, “IXM: ロボット制御ソフトウェア向けプロセス間通信ミドルウェア,” *情報処理学会論文誌*, vol. 58, no. 10, pp. 1578–1590, 2017.

12）オムロン株式会社. *NJ/NX シリーズ CPU ユニット ユーザーズマニュアル ソフトウェア編（SBCA-467Z）*, pp. 5–46–5–90.

13）*Systems and software engineering*, ISO/IEC 25010, 2011.

14）H. Durrant-Whyte, T. Bailey, “Simultaneous localization and mapping: part I,” *IEEE Robot. & Automat. Mag.,* vol. 13, no. 2, pp. 99–110, 2006.

15）AWS. “オムロン, 研究開発用の HPC 基盤を AWS 上に構築, 最適なコンピューティングリソースを活用し, 革新的技術開発をリード.” AWS 導入事例：オムロン株式会社 | AWS. https://aws.amazon.com/jp/solutions/case-studies/omron-case-study/（Accessed: Jun. 19, 2024）.

16）ROS.org. “urdf.” urdf - ROS Wiki. https://wiki.ros.org/urdf（Accessed: Jun. 19, 2024）.

執筆者紹介

松永 大介　MATSUNAGA Daisuke
技術・知財本部
ロボティクス R&D センタ
Voyager Project 部
専門：ソフトウェア工学

山本 智哉　YAMAMOTO Tomoya
技術・知財本部
ロボティクス R&D センタ
Voyager Project 部
専門：ソフトウェア工学

藤井 春香　FUJII Haruka
技術・知財本部
ロボティクス R&D センタ
Voyager Project 部
専門：ソフトウェア工学

小島 岳史　KOJIMA Takeshi
技術・知財本部
ロボティクス R&D センタ
Voyager Project 部
専門：ソフトウェア工学