Vol.2022-ARC-248 No.38
Vol.2022-SLDM-198 No.38
Vol.2022-EMB-59 No.38
2022/3/10

ROS 2ノード軽量実行環境mROS 2における任意型メッセージの通信処理方式

檜原 陽一郎[1]　中村 宏[1]　高瀬 英希[1,2]

概要：近年，ロボットシステムの社会的な需要の高まりを受けROSの活用事例が増加しており，その次世代版となるROS 2にも注目が集まっている．しかし，ROS 2は基本的に汎用OSが搭載された実行環境を必要とするため，応答性やリアルタイム性およびリソース使用量に関して課題があった．これを解決すべく，ROS 2を用いるシステムに組込み技術を導入することに注目が集まっている．我々は，仲介の仕組み無しにROS 2ノードと等価な出版購読型の通信を実現できる組込み向けの軽量実行環境であるmROS 2の研究開発を進めている．これまでのmROS 2は，基本型による通信のみに対応しており，さらにライブラリ内部のコードを型ごとに修正する必要があるため，本環境の利用時の開発コストが大きくなっていた．また，基本型以外の型は文字列型に変換して通信する必要があり，非効率な通信を強いられる．そこで本研究では，mROS 2における任意のメッセージ型による通信の実現手法を提案する．提案手法では，通信時の処理をメッセージ型に関して共通の処理及び固有の処理に分離し，固有処理を行うファイルをメッセージ型ごとに生成する．通信フローに生成された情報を組み込んだ上で，任意のメッセージ型による出版購読の通信時を効率化する通信処理方式を提案する．提案手法をSTM32 NUCLEO-F767ZIボード上に実装し，通信性能およびプログラムサイズにおいて既存手法に対する有効性を確認した．

1. はじめに

近年，様々な分野においてロボットシステムの需要が高まっている一方で，ロボットシステム開発に必要な知識は広範化し，開発工程は複雑化している．このような課題を解決するひとつの手段として，ROS（Robot Operating System）[1]の活用事例が増加している．

ROSは，ロボットシステム開発のために必要となる通信ミドルウェア，ライブラリ，ツール群およびユーザと開発者を繋ぐオープンなコミュニティを提供している．ROSの大きな特徴として，機能単位であるノードの組み合わせによってロボットシステムを構築できるという点が挙げられる．このような設計にすることで疎結合なシステムになり，システム規模の拡大にも容易に対応可能となる．また，世界中で公開されている既存のパッケージを再利用することが可能であるという利点もある．ROSの活用事例は多岐に渡り，Amazon Roboticsの物流補助ロボット，Sonyのaibo，自動運転システム用オープンソースソフトウェアAutowareなど様々なロボットの研究に用いられている．

現在では，次世代版ROSであるROS 2 [2]に注目が集まっている．ROS 2は基本的に汎用OSが搭載された実行環境を必要とする．しかし，汎用OSを搭載するには豊富なリソースを必要とするため，システムのコストが高くなり，消費電力も大きくなる．加えて汎用OSのタスクスケジューリングの性質上，応答時間が長くなることがある．このため，システムにおいて利用可能なリソースが限定される場合や応答性およびリアルタイム性を要求する場合には，ROS 2を用いることが困難であった．

このような課題を解決するひとつの手段として，ROS 2を用いるシステムの一部に組込み技術を導入することに注目が集まっている．組込みデバイスには，搭載するために必要なリソースは限定的であるが，リアルタイム性の確保に向いたOSであるRTOS(リアルタイムOS)を搭載できる．そのため，システム機能の一部を組込みデバイス上で実行することで，リソース使用量を抑制しつつもリアルタイム性を高くすることが期待できる．これを実現するために，組込みデバイス上でROS 2ノードを実行することを目的としたROS 2ノード軽量実行環境が複数提案されている．本研究はその中でも，文献[3]において提案されたmROS 2に着目する．mROS 2は，ROS 2と同じ通信プロトコルであるRTPS（Real Time Publish Subscribe protocol）[4]を用いているため，ROS 2ノードおよびmROS 2ノード

[1] 東京大学
The University of Tokyo
[2] JST さきがけ
JST PRESTO

Vol.2022-ARC-248 No.38
Vol.2022-SLDM-198 No.38
Vol.2022-EMB-59 No.38
2022/3/10

が直接通信できる．そのため，ブリッジが必要なその他の既存環境と比較して通信のオーバーヘッドが小さい．しかし，これまでの mROS 2 は基本型による通信のみに対応しており，さらにライブラリ内部のコードを型ごとに修正する必要があるため，本環境の利用時の開発コストが大きくなっていた．また，基本型以外の型は文字列型に変換して通信する必要があり，非効率な通信を強いられていた．

本研究の目的は，mROS 2 において，ROS 2 の扱うことができる任意のメッセージ型による通信を可能とする手法を提案し，実装することにより，通信性能を向上させることならびに開発コストを削減することである．本研究の成果により，mROS 2 の適用範囲が大きく拡大し，ROS 2 および mROS 2 を併用した，リアルタイム性や応答性およびリソース使用量の面において優れたロボットシステムの実現に貢献することが期待される．

本論文の構成は次のとおりである．まず 2 章では, ROS 2, mROS 2 の概要および関連研究について述べる．3 章では提案手法である，mROS 2 における任意のメッセージ型による通信方式について詳説する．4 章では，提案手法を STM32 NUCLEO-F767ZI ボード上に実装し，通信性能およびプログラムサイズの面から有効性を評価する．最後に 5 章において，本論文のまとめと今後の展望を示す．

2. 準備

2.1 ROS 2

ROS [1] は，ロボット開発のために必要となる通信ミドルウェア，ライブラリ，ツール群およびユーザと開発者を繋ぐオープンなコミュニティである ROS コミュニティを提供する，オープンソースのロボットシステム開発支援プラットフォームである．現在，その次世代版となる ROS 2 に注目が集まっている．

ROS 2 が提供する代表的な通信機能として，出版購読型通信モデルに基づいたノード間通信がある．ROS 2 では，出版購読型通信における通信ミドルウェアの仕様として DDS [5] を採用している．DDS とは，OMG(Object Management Group) によって標準化されている，分散システムにおける通信ミドルウェアの仕様である．DDS の通信プロトコルには，RTPS [4] が採用されている．DDS に準拠する通信ミドルウェアは複数あるが，例えば ROS 2 では Fast DDS [6] などが用いられている．さらに ROS 2 では，使用する通信ミドルウェアを用途に合わせて切り替えることができる．

ここで，文献 [7] において示されている，出版購読型通信モデルを用いた自律走行システムの構築例を図 1 に示す．楕円がノードを，四角形がトピックを表す．このシステムは，各種のセンサデータを出版する `Sensor` ノードから，各処理を行う様々なノードを経由し，最終的に `Control` ノードが制御命令を発行するという形になっている．

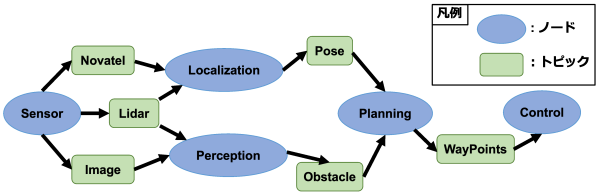

図 1 ROS 2 を用いた自律走行システムの構築例.

```
Image.msg
uint8 is_bigendian
uint32 step
uint8[] data
```

図 2 任意型定義ファイルの例.

また ROS 2 では，トピックごとに通過するメッセージの型を指定することが可能である．指定可能な型は，文字列型や数値型などの基本型だけでなく，それらを組み合わせた型である任意型も指定できる．ROS 2 において任意型を用いるには，その型を定義するファイルを作成する必要がある．その例を図 2 に示す．図 2 に示すように，画像データを表現する `Image` 型は `uint8` 型，`uint32` 型および `uint8` 型の動的配列を内部変数に持ち，それぞれ `is_bigendian`，`step` および `data` という変数名が付いている．この定義ファイルを元に，ROS 2 のビルドツールによって `Image` 型用ヘッダファイルを生成し，通信フローに組み込むことで，`Image` 型による通信が可能となる．

2.2 mROS 2

mROS 2 は，文献 [3] において提案された，オープンソースの組込み向け ROS 2 ノード軽量実行環境である．

mROS 2 のソフトウェア構成を述べる．ユーザ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，リアルタイム OS およびハードウェア抽象化ライブラリによって構成される．通信ライブラリは，rclcpp から最小限の機能を選定し，互換性を保つように設計している．通信プロトコルスタックは，組込みデバイス向けに RTPS の自律通信機能が実装され，Fast DDS との疎通も確認されている embeddedRTPS [8] を用いている．またリアルタイム OS には，組込みシステム向けの設計がなされた TOPPERS/ASP3 カーネル [9] を採用し，ハードウェア抽象化ライブラリには，STM32 マイコン向けの STM32Cube ライブラリ [10] を用いる．

ROS 2 および mROS 2 は，いずれも通信プロトコルに RTPS を用いる．そのため，mROS 2 ノードは，ROS 2 ノードと直接通信可能である．このことにより，mROS 2 以外の組込み向け ROS 2 ノード実行環境において問題となっている，ブリッジの仲介による通信のオーバヘッドおよび単一障害点の発生などの問題が解決できる．

一方で，これまでの mROS 2 は基本型による通信のみに

対応しており，さらにライブラリ内部のコードを型ごとに修正する必要があるため，本環境の利用時の開発コストが大きくなってしまっていた．また，基本型以外のメッセージに関しては文字列に変換して通信する必要があった．そのため，実用的なロボットシステム開発において，度重なる型変換処理に起因する通信のオーバーヘッドの発生や開発コストの増大という課題を抱えていた．

2.3 関連研究

既存の ROS 2 ノード軽量実行環境としてまず挙げられるのが，micro-ROS [11] である．micro-ROS は，通信ミドルウェアとして Micro XRCE-DDS [12] を用いている．Micro-XRCE-DDS により汎用デバイスおよび組込みデバイス間の通信を行う際は，仲介する役割を担う Agent を用いる必要がある．そのため，通信のオーバーヘッドが発生することに加え，複数の組込みデバイスによる通信を行う際に，Agent が単一障害点となるなどの問題が起こる．また，rclcpp を用いた開発を行うことはできない．

その他の既存環境としては，ros2arduino [13] がある．ros2arduino は，arduino などのローエンド組込みデバイスをターゲットとし，OS レスの環境においても動作する．しかし，micro-ROS と同じく Micro XRCE-DDS を用いているため，Agent 無用の通信を行うことはできない．また，ros2arduino アプリケーションプログラムは，ROS 2 ノードプログラムとの互換性があまり高くなく，学習コストが大きいという問題点もある．さらに，ROS 2 Foxy 以降への対応予定はないということが明言されている．

また，文献 [14] において，mROS 2 と同じく通信プロトコルに RTPS を用いた環境が提案されている．この研究では FreeRTOS [15] および FreeRTPS [16] を統合した環境を提案し，実装および評価を行っている．しかし，2017 年 10 月を最後に ROS Wiki および実装の更新が止まっている．

3. 任意のメッセージ型の通信処理方式

本章では，mROS 2 において，ROS 2 の扱うことができる任意のメッセージ型による通信を実現する手法を提案する．

3.1 設計要件

設計要件を定義する．まず，ROS 2 の扱うことができる任意のメッセージ型とは，以下を内部に保持する型である．

- 基本型
- 既に定義された任意型
- 上記いずれかの配列

提案手法では，上記の型による通信を可能とした上で，通信性能，プログラムサイズおよび ROS 2 ノードプログラムとの互換性の 3 点を考慮した設計を行う．通信性能に関して考慮すべき要素としては，応答性およびリアルタイム性があり，通信遅延時間およびその変動がそれらに対応する．そこで提案手法では，従来の mROS 2 と比較して通信遅延時間およびその変動が小さくなることを目指す．プログラムサイズが肥大化すると，mROS 2 を搭載することのできる組込みデバイスが，十分なメモリを備えたデバイスに限定される．このため提案手法では，従来の mROS 2 と比較して，可能な限りプログラムサイズが大きくならないことを目指す．ROS 2 の開発者が可能な限り小さい負担で mROS 2 を利用できるよう，学習コストを抑制することも重要である．提案手法では，mROS 2 において任意型メッセージを通信に用いる際のフローが，ROS 2 におけるフローと可能な限り高い互換性を持つようにすることも目指す．

3.2 設計方針

3.1 節の設計要件を踏まえ，その実現方針を示す．

メッセージを通信する際に mROS 2 が行う処理は，メッセージ型に依存しない共通処理および依存する固有処理に大別される．提案手法では，共通処理および固有処理を分離し，共通処理が固有処理を呼び出すという形で設計を行う．共通処理および固有処理を疎結合にすることにより，通信するメッセージ型ごとに固有処理を行うヘッダファイルを生成し，通信フローに組み込むだけで通信を行える．加えて，通信に使用するメッセージ型のヘッダファイルのみを通信フローに組み込むことが可能となるため，プログラムサイズを抑えることが期待できる．

また，メッセージ型固有の処理を行うヘッダファイルの生成フローは，基本型の場合および任意型の場合に分けて考える．基本型の場合は，予め実行する処理が決まっているためヘッダファイルを事前に用意することが可能である．それに対し任意型の場合は，その型の構造に依存して実行すべき処理が決まるため，任意型の定義ファイルを元にヘッダファイルを生成する必要がある．

3.3 メッセージ型固有ヘッダファイルの生成フロー

本節では，メッセージ型固有の処理を行うヘッダファイルの生成フローに関して詳細に説明を行う．

まずは，基本型の場合について述べる．提案手法において mROS 2 がサポートする基本型は，`bool` 型，`byte` 型，`char` 型，`float32/64` 型，`header` 型，`int8/16/32/64` 型，`string` 型，`uint8/16/32/64` 型である．

3.2 節で述べたように，基本型固有の処理を行うヘッダファイルは，事前にそれぞれ用意しておく．これにより基本型による通信を行う際は，使用する型のヘッダファイルを通信フローに組み込むだけで，その型による通信が可能となる．

続いて，任意型の場合について述べる．提案手法においては，基本型以外は全て任意型として扱うこととする．つ

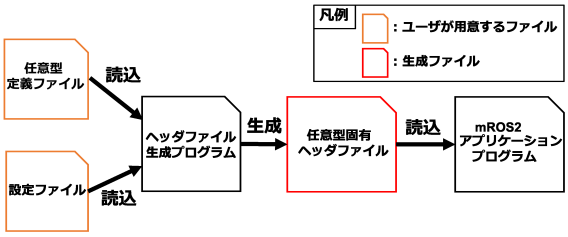

図 3 任意型用ヘッダファイルの生成フロー．

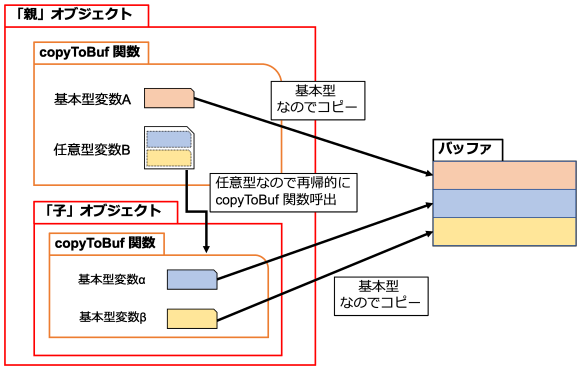

図 4 copyToBuf 関数において行う処理．

まり，基本型の配列も任意型として扱う．任意型の場合は，それを使用するアプリケーションの要求に依存するため，2.1 節の図 2 に示したものと同等の任意型定義ファイルを元に生成する．具体的なフローは，図 3 に示すように，次のとおりである．

(1) 任意型の定義を定義ファイルに記述する．
(2) 通信に使用する任意型の定義ファイルへのパスを記述した設定ファイルを用意する．
(3) 設定ファイルをヘッダーファイル生成プログラムに読み込ませ，ヘッダファイルを生成する．

以上の生成フローは，ROS 2 におけるヘッダファイル生成フローとの互換性が高くなるように設計を行っている．

3.4 メッセージ型固有の通信処理

本節では，メッセージ型固有のヘッダファイル内で行う通信処理の内容を述べる．

3.4.1 メッセージ送信時の処理

メッセージ送信時にヘッダファイル内で行う処理は，2 つに大別される．1 つ目は，メッセージオブジェクトをシリアライズしてバッファにコピーする処理である．2 つ目は，メッセージサイズを計算する処理である．

提案手法では，メッセージオブジェクトのメンバ変数の値を，順にバッファにコピーをすることによりシリアライズを行う．これを行う関数を，メッセージオブジェクトのメンバ関数である copyToBuf 関数として定義する．copyToBuf 関数は，図 4 に示すとおり，次の処理を行う．

(1) 引数として，コピー先のバッファ領域の先頭アドレスを指すポインタを受け取る．以降，このポインタを移動させながら，ポインタの指す領域を開始地点として，バッファへのデータのコピーを行う．
(2) メッセージオブジェクトのメンバ変数の型に応じて，次のいずれかの処理を行う．
- 文字列型以外の基本型ならば，メンバ変数の値をバッファにコピーし，ポインタを移動させる．
- 文字列型または動的配列型ならば，まずはそのサイズを計算し，その値をバッファにコピーする．続いてサイズの情報を元に，メンバ変数の値をバッファにコピーする．ポインタも移動させる．
- 静的配列型ならば，既にサイズが分かっているため，そのサイズの情報を元に，メンバ変数の値をバッファにコピーする．ポインタも移動させる．
- 既存の任意型ならば，「子」オブジェクトの copyToBuf 関数を再帰的に呼び出す．その際引数としては，その時点におけるポインタを渡す．また，「子」オブジェクトの copyToBuf 関数内で動かしたポインタの分だけ，「親」オブジェクト側のポインタも移動させる．

(3) 返り値として，copyToBuf 関数内でポインタをどれだけ移動させたかを返す．

上記の説明および図 4 に示すように，任意型の中に任意型が含まれている場合においても，「親」オブジェクトの copyToBuf 関数内で「子」オブジェクトの copyToBuf 関数を再帰的に呼び出すことにより，いずれは基本型または配列型の処理に到達する．従って，任意のメッセージ型に対して正しく処理を行うことが可能となる．

メッセージの送信時には，バッファのどの領域までを送信するかを把握するために，メッセージサイズの情報も必要となる．これを計算する関数を，メッセージオブジェクトのメンバ関数である getTotalSize 関数として定義する．getTotalSize 関数は，copyToBuf 関数においてどれだけポインタが移動したかを返す．

3.4.2 メッセージ受信時の処理

メッセージ受信時にヘッダファイル内で行う処理は，バッファからデータをコピーし，それをデシリアライズしてメッセージオブジェクトのメンバ変数に値を格納するという処理である．これを行う関数を，copyFromBuf 関数というメッセージオブジェクトのメンバ関数として定義する．このフローは，図 5 に示すとおり，3.4.1 節で示した copyToBuf 関数と全く逆の処理を行うだけである．

3.5 通信時の動作フロー

本節では，前節までの内容を踏まえて設計を行った，メッセージ通信時の動作フローについて述べる．

3.5.1 ノードの登録

通信を行う前に，mROS 2 ノードの登録が必要となる．この時ノードごとに，通信に用いるメッセージ型を指定する．具体的にはノード登録を行う関数に型名を渡すこと

Vol.2022-ARC-248 No.38
Vol.2022-SLDM-198 No.38
Vol.2022-EMB-59 No.38
2022/3/10

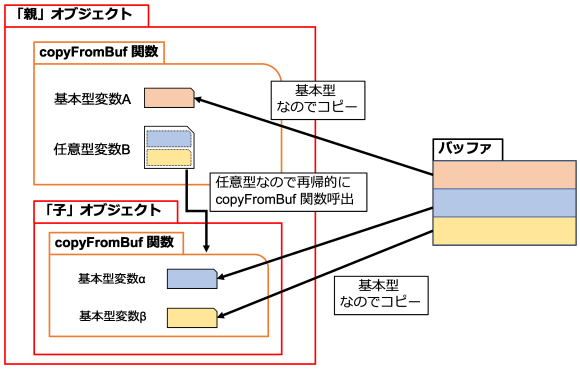

図 5 copyFromBuf 関数において行う処理.

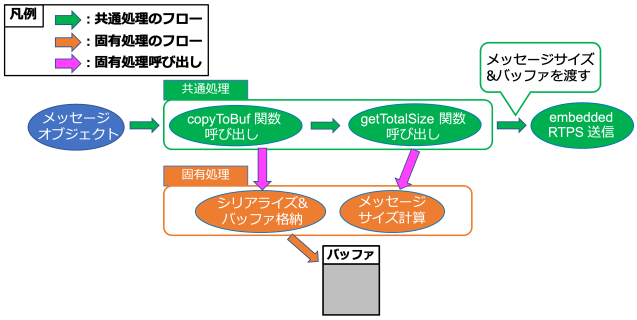

図 6 メッセージ送信時に行う処理.

で，その型による通信を行うための共通処理を行う関数，すなわち固有処理を呼び出すための関数が生成される．ただし，固有処理を行うヘッダファイルは，3.3 節にて述べたフローにより事前に用意する必要があることに注意する．

3.5.2 メッセージの送信

メッセージ送信時の具体的なフローは，図 6 に示すように，次のとおりである．

(1) メッセージオブジェクトを，共通処理が受け取る．
(2) 共通処理が，copyToBuf 関数を呼び出す．
(3) copyToBuf 関数が，メッセージオブジェクトのシリアライズおよびバッファへの格納処理を実行する．
(4) 共通処理が，getTotalSize 関数を呼び出す．
(5) getTotalSize 関数が，メッセージサイズを計算する．
(6) バッファおよびメッセージサイズの情報を embeddedRTPS の送信タスクに渡し，メッセージを送信する．

3.5.3 メッセージの受信

メッセージ受信時の具体的なフローは，図 7 に示すように，次のとおりである．

(1) embeddedRTPS 受信タスクがメッセージを受信する．
(2) 共通処理において，受信データを入れるためのメッセージオブジェクトを生成する．
(3) 共通処理が，copyFromBuf 関数を呼び出す．
(4) copyFromBuf 関数が，バッファにあるデータのデシリアライズおよびメンバ変数への格納処理を実行する．
(5) 共通処理が，コールバック関数を呼び出す．

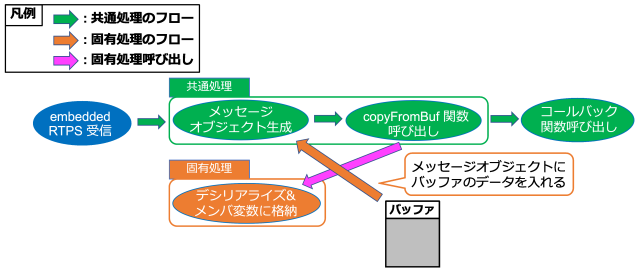

図 7 メッセージ受信時に行う処理.

4. 評価

前章の提案に基づいた機能拡張を mROS 2 に施した[*1]．その上で，任意のメッセージ型による通信を正しく行えるかどうかの確認を行った．その結果，3.3 節にて述べた，mROS 2 のサポートする基本型については，全ての型において通信が正しく行えることを確認した．任意型についても，確認した限りは通信を正しく行えている．さらに，通信性能およびプログラムサイズの観点から，提案手法の有効性を定量的に評価した．

4.1 評価環境

評価対象の環境は，mROS 2 の v0.2.1，v0.2.3 および提案手法とする．v0.2.1 は，機能拡張を施していない従来の環境である．v0.2.3 は，基本型に関してのみ提案手法に基づく実装を施した環境である．提案手法は，任意のメッセージ型による通信を可能とする機能拡張を施した環境である．なお，本研究成果は mROS 2 の v0.2.4 として公開予定である．

実験には，mROS 2 ノードを動作させるための組込みデバイスおよび ROS 2 ノードを動作させるためのノート PC を使用する．それぞれの環境の詳細を表 1 に示す．なお実験時には，イーサネットケーブルによりノート PC および組込みデバイスを接続した．

4.2 通信性能

本節では，通信遅延時間の評価結果を示す．計測に用いるシステムは，メッセージの送信/受信を行う ROS 2 ノード，および，受信したメッセージをそのまま送り返す mROS 2 ノードから構成される．このシステムでは，ROS 2 送信ノードがメッセージを送信し，それを mROS 2 ノードが受け取って直ちに返送し，最後に ROS 2 受信ノードがそれを受信するという流れで通信を行う．時刻の記録は，ROS 2 の提供するシステムクロック機能を用いて，ROS 2 送信/受信ノードにおいて行う．評価は，文字列型による通信お

[*1] ただし本論文執筆時点では，任意型の配列を含む任意型メッセージ，および，250B 以上のサイズを持つメッセージによる通信に関しては対応していない．これらに関しては早急に対応予定である．

Vol.2022-ARC-248 No.38
Vol.2022-SLDM-198 No.38
Vol.2022-EMB-59 No.38
2022/3/10

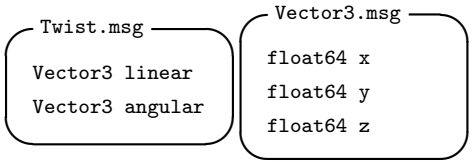

図 8 Twist 型のメッセージ構造.

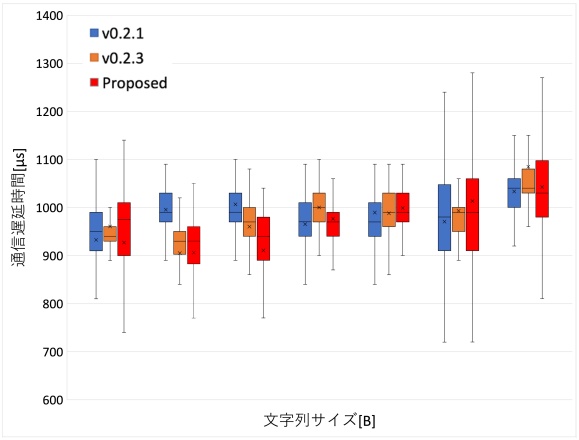

図 9 文字列型による遅延時間評価結果.

および図 8 に示す任意型である Twist 型による通信に関して行った.

4.2.1 文字列型による通信

文字列のサイズを 1 B から 64 B まで変化させながら，文字列型による通信の遅延時間を計測した．ここでの遅延時間とは，ROS 2 送信ノードからメッセージが送信される直前の時刻および mROS 2 ノードから返送されたメッセージを ROS 2 受信ノードが受信した直後の時刻の差分とする．各文字列サイズにおける計測結果を，図 9 の箱ひげ図および表 2 に示す．表 2 に示す値は，小数点第一位を四捨五入している．

表 1 評価環境.

組込みデバイス	STM32 NUCLEO-F767ZI [17]
プロセッサ	Arm Cortex-M7
最大動作周波数	216 MHz
RAM	512 KB
OS	TOPPERS/ASP3
mROS 2	v0.2.1/v0.2.3/提案手法
汎用デバイス	ノート PC
プロセッサ	intel CORE i7 8565U CPU
最大動作周波数	1.80 GHz × 8 core
RAM	16 GiB
OS	Ubuntu 20.04.3 LTS
ROS 2	Foxy

表 2 1–64B の文字列型による通信遅延時間の平均値 [μs].

	1 B	2 B	4 B	8 B	16 B	32 B	64 B
v0.2.1	932	995	1006	965	989	971	1033
v0.2.3	961	905	960	1000	988	993	1085
提案手法	927	906	911	977	999	1013	1043

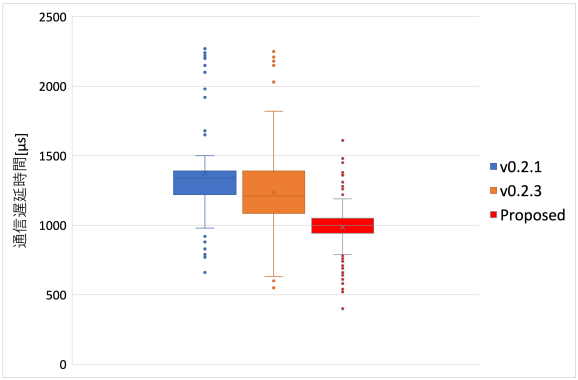

図 10 型変換の処理を除く Twist 型の通信遅延時間.

図 9 および表 2 に示すとおり，各バージョン間における顕著な遅延時間の差は見られないことが確認できる．このことから，共通処理および固有処理を分離するという設計に起因するオーバヘッドはほぼ発生しないことがいえる.

4.2.2 Twist 型による通信

Twist 型は，図 8 に示すとおり，内部に 2 つの Vector3 型という任意型を含み，さらにその Vector3 型は，内部に 3 つの float64 型変数を持つ．なお以降，v0.2.1 および v0.2.3 のことを旧バージョンと呼ぶ.

mROS 2 の旧バージョンにおいては，任意型である Twist 型による通信を行えない．そこで旧バージョンにおいては，まず ROS 2 送信ノードにおいて Twist 型メッセージを文字列に変換して送信する．続いて，mROS 2 ノードにおいて受信した文字列を Twist 型メッセージに変換し，再び文字列に変換して返送する．最後に，ROS 2 受信ノードにおいて受信した文字列を Twist 型メッセージに変換するという形で，擬似的に Twist 型による通信を再現する．なお，ROS 2 ノードにおける型変換には to_string 命令を用いる．mROS 2 ノードにおいては to_string 命令を利用できないため，型変換処理を行う自作の関数を用いている.

Twist 型による通信に関しては，3 つの実験を行った．ROS 2 ノードにおける型変換を含めない場合および含める場合の通信遅延時間の計測，および，mROS 2 ノードにおける処理時間の内訳の計測である.

まず，ROS 2 ノードにおける型変換を含めない場合の遅延時間の計測結果を図 10 および表 3 に示す．ここでの遅延時間とは，ROS 2 送信ノードにおいてメッセージを送信する直前から，mROS 2 ノードによるメッセージの返送を経て，ROS 2 受信ノードにおいてメッセージを受信する直後までの時間とする.

図 10 および表 3 に示すとおり，提案手法における遅延

表 3 型変換の処理を除く Twist 型の通信遅延時間 [μs].

	平均値	標準偏差
v0.2.1	1373.81	308.60
v0.2.3	1234.56	274.67
提案手法	984.46	173.53

Vol.2022-ARC-248 No.38
Vol.2022-SLDM-198 No.38
Vol.2022-EMB-59 No.38
2022/3/10

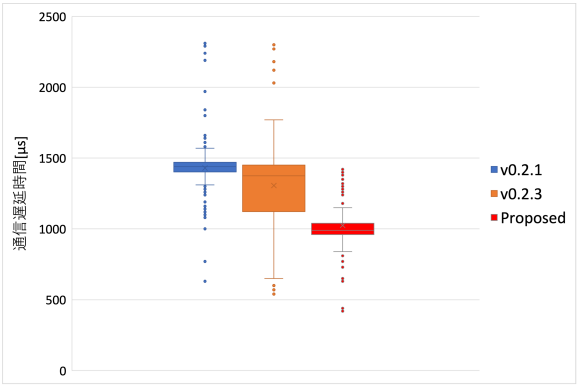

図 11 型変換の処理を含む Twist 型の通信遅延時間.

時間は，旧バージョンと比較して 200 μs から 300 μs 程度小さくなっている．この理由としてまず考えられるのは，mROS 2 ノードにおける型変換のオーバーヘッドである．後述するが，これによる差は 70 μs 程度であるということがわかっている．つまりそれ以外にも要因があると考えられる．まず考えられる要因としては，メッセージサイズの差異による影響が挙げられる．旧バージョンにおいては文字列型による通信を行い，提案手法においては実質的には float64 型を 6 つ持つ Twist 型による通信を行うが，文字列型による通信を行う際は，文字列サイズの情報も合わせて通信する必要があり，Twist 型メッセージと比較して 4 B 程度メッセージサイズが大きくなる．しかし，その程度の差が遅延時間にここまでの影響を及ぼすとは考えにくい．それ以外の要因については現状追究できていない．また，提案手法においては遅延時間の変動も小さくなっているが，その要因についても現状追究できていない．

続いて，ROS 2 ノードにおける型変換を含める場合の遅延時間の計測結果を図 11 および表 4 に示す．ここでの遅延時間とは，ROS 2 送信ノードにおいて送信する Twist 型メッセージを文字列に変換する直前から，mROS 2 ノードによる返送を経て，ROS 2 受信ノードにおいて受信した文字列を Twist 型に変換する直後までの時間とする．

図 11 および表 4 に示すとおり，提案手法における遅延時間は，旧バージョンと比較して 300 μs から 400 μs 程度小さくなっている．本実験では ROS 2 ノードにおける型変換の時間も含めているため，図 10 および表 3 の結果と比較して遅延時間の差が大きくなっている．また表 3 および表 4 を比較すると，ROS 2 ノードにおける型変換のオーバーヘッドが合計 60 μs 程度となることもわかる．

最後に，mROS 2 ノードにおける処理時間の内訳の計測も行った．計測は，TOPPERS/ASP3 カーネルの提供するタイマ用 API である fch_hrt 関数を各地点において呼び出して時刻を記録し，その差分を取ることにより行った．本実験において時刻の記録を行った地点は以下のとおりであり，各区間における処理時間の計測結果を表 5 に示す．

(1) メッセージを受信した直後.
(2) メッセージのデシリアライズを行い，コールバック関数に処理を渡した直後.
(3) メッセージを文字列型から Twist 型に変換し，再度文字列型に変換した直後（提案手法では行わない）.
(4) メッセージをシリアライズし，送信した直後.

表 5 に示すとおり，区間 (2)–(3) において，旧バージョンおよび提案手法における処理時間の顕著な差が見られる．これは，提案手法においては型変換を行う必要がないためである．また，区間 (1)–(2) においても処理時間が若干短くなっている．これは，旧バージョンにおいては文字列型を，提案手法においては実質的には float64 型を 6 つ持つ Twist 型をデシリアライズすることになるが，文字列型の通信には文字列サイズ情報も含む必要がある分，デシリアライズにおける処理が複雑になるためであると考えられる．

4.3 プログラムサイズ

本節では，Twist 型による通信を行うための mROS 2 プログラムのサイズを評価する．

本実験では，3 つの場合におけるプログラムサイズを計測した．1 つ目は，Twist 型による通信を行うための mROS 2 プログラム全体のサイズである．2 つ目は，Twist 型による通信を行うための mROS 2 アプリケーションプログラムのみのサイズである．3 つ目は，mROS 2 プラットフォームのプログラムサイズである．これは，1 つ目および 2 つ目のサイズの差分である．なおプログラムサイズは，arm-none-eabi-size を用いて，各ファイルの各領域ごとのサイズを取得する．その各領域に関する説明を行い，それぞれの結果を表 6，表 7 および表 8 に示す．ただし表 8 に示す値は，表 6 の dec 領域のサイズから，表 7 の dec 領域のサイズを除いたものである．

- text 領域: ROM 領域のプログラムの合計サイズ.
- data 領域: ROM 領域にある初期値で初期化され，RAM 領域にコピーされる変数の合計サイズ.
- bss 領域: RAM 領域の未初期化変数の合計サイズ.
- dec 領域: 上記 3 つの領域の合計サイズ.

表 4 型変換の処理を含む Twist 型の通信遅延時間 [μs].

	平均値	標準偏差
v0.2.1	1429.64	212.17
v0.2.3	1305.70	302.962
提案手法	1024.50	154.38

表 5 Twist 型通信時の mROS 2 ノード内処理時間内訳 [μs].

	区間 1–2	区間 2–3	区間 3–4	全体
v0.2.1	5.14	71.58	20.76	97.47
v0.2.3	5.17	70.87	20.61	96.64
提案手法	3.28	0.24	20.40	23.92

表 7 に示す結果を見ると，提案手法と比較し旧バージョンにおけるアプリケーションサイズは，特に text 領域において大きくなっていることがわかる．これは，旧バージョンにおいては型変換の関数を自作し，それを用いて型変換を行うという処理の記述が必要であるからだと考えられる．しかし，表 6 および表 8 に示す結果から，旧バージョンおよび提案手法の間の mROS 2 プログラムサイズの差は，そのほとんどが mROS 2 プラットフォームサイズにおける差によるものであることがわかる．これは提案手法における設計により，不要な部分のプログラムが削減できたことが 1 つの要因と考えられる．

5. おわりに

本研究では，ROS 2 ノード軽量実行環境 mROS 2 において，ROS 2 の扱うことができる任意のメッセージ型による通信を行うための手法を提案した．具体的には，mROS 2 における通信処理を，メッセージ型に関して共通の処理および固有の処理に分離し，固有処理を行うファイルはメッセージ型ごとに準備または生成する．そのファイルを通信フローに組み込むことで，任意のメッセージ型による通信が可能となった．

さらに，提案手法に基づいた機能拡張を mROS 2 に施し，動作確認および性能評価を行った．動作確認では，基本型，`Twist` 型などの任意型による通信が正しく行えていることを確認した．また性能評価では，通信性能およびプログラムサイズにおいて有効性を確認した．

本研究により，mROS 2 の通信性能の向上ならびに開発コストの削減を達成できた．これにより mROS 2 の適用範囲が大きく拡大し，ROS 2 および mROS 2 を併用した，リアルタイム性や応答性およびリソース使用量の面において優れたロボットシステムの実現への貢献が期待される．

mROS 2 の任意型メッセージ通信に関して早急に解決すべき課題は 2 つある．1 つ目は，任意型の配列を含む型による通信を可能にすることである．2 つ目は，250 B 以上のメッセージサイズによる通信を可能にすることである．

表 6 `Twist` 型通信時の，mROS 2 プログラム全体サイズ [B].

	text	data	bss	dec
v0.2.1	99971	16996	118800	235767
v0.2.3	99987	16996	118800	235783
提案手法	90419	16632	118800	225851

表 7 `Twist` 型通信時の，mROS 2 アプリケーションサイズ [B].

	text	data	bss	dec
v0.2.1	2865	4	52	2921
v0.2.3	2865	4	52	2921
提案手法	1265	4	32	1332

表 8 `Twist` 型通信時の，mROS 2 プラットフォームサイズ [B].

v0.2.1	232846
v0.2.3	232862
提案手法	224519

謝辞 本研究の一部は，JST さきがけ JPMJPR18M8 の支援ならびに国立研究開発法人情報通信研究機構の委託研究（04001）により得られたものである．

参考文献

[1] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y.: ROS: an open-source Robot Operating System, *Proc. of ICRA workshop on open source software*, No. 3.2, pp. 1–6 (2009).
[2] Woodall, W.: ROS on DDS (online), `https://design.ros2.org/articles/ros_on_dds.html` (2022.01.29).
[3] 高瀬 英希，祐源 英俊：mROS 2：組込みデバイス向けの ROS 2 ノード軽量実行環境，情報処理学会研究報告，発表予定 (2022).
[4] Object Management Group: About the DDS Interoperability Wire Protocol Version 2.5 (online), `https://www.omg.org/spec/DDSI-RTPS/` (2022.01.29).
[5] Object Management Group: About the Data Distribution Service Specification Version 1.4 (online), `https://www.omg.org/spec/DDS/` (2022.01.29).
[6] DDS API - Fast DDS 2.5.0 documentation (online), `https://fast-dds.docs.eprosima.com/en/latest/` (2022.01.29).
[7] Jiang, Z., Gong, Y., Zhai, J. et al. Message Passing Optimization in Robot Operating System, *Int J Parallel Prog* 48, pp. 119–136 (2020).
[8] Kampmann, A., Wüstenberg, A., Alrifaee B. and Kowalewski, S: A Portable Implementation of the Real-Time Publish-Subscribe Protocol for Microcontrollers in Distributed Robotic Applications, *Proc. of 2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 443–448 (2019).
[9] TOPPERS/ASP3 kernel (online), `https://toppers.jp/asp3-kernel.html` (2022.01.29).
[10] STMicroelectronics: STM32Cube Development Software (online), `https://www.st.com/ja/ecosystems/stm32cube.html` (2022.01.29).
[11] micro-ROS | ROS 2 for microcontrollers (online), `https://micro.ros.org/` (2022.01.29).
[12] eProsima Micro XRCE-DDS (online), `https://micro-xrce-dds.docs.eprosima.com` (2022.01.29).
[13] ROBOTIS-GIT/ros2arduino (online), `https://github.com/ROBOTIS-GIT/ros2arduino` (2022.01.29).
[14] da Silva Medeiros L., Julio R.E., Almeida R.M.A.., Bastos G.S.: Enabling Real-Time Processing for ROS2 Embedded Systems, *Koubaa A. (eds) Robot Operating System (ROS), Studies in Computational Intelligence*, vol. 778, pp. 477–528, Springer, Cham (2019).
[15] FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions (online), `https://www.freertos.org/` (2022.01.29).
[16] ros2/freertps (online), `https://github.com/ros2/freertps/` (2022.01.29).
[17] STMicroelectronics: NUCLEO-F767ZI (online), `https://www.st.com/en/evaluation-tools/nucleo-f767zi.html` (2022.01.29).